

DYNAMIC MULTILEVEL GRAPH LAYOUT AND VISUALISATION

CARL JONATHAN CRAWFORD

A thesis submitted in partial fulfillment of the requirements of the
University of Greenwich for the degree of Doctor of Philosophy

June 2016

DECLARATION

“I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of Doctor of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others”.

.....

Carl Crawford

.....

Christopher Walshaw

.....

Alan Soper

ACKNOWLEDGEMENTS

A big thank you to everyone that helped me on my way.

Large or little, I couldn't have done it without you.

ABSTRACT

This thesis addresses the issue of efficient dynamic graph drawing for large scale connected graphs with around 10,000 vertices. It contains three main contributions.

Firstly, an efficient method for approximating the n-body calculations used in Force Directed Placement (*FDP*) is described, exploiting use of a multilevel scheme to approximate distance between groups of vertices much like the Barnes Hut Octree. The method suggests better representation of the graphs underlying relationships. In experiments this algorithm, referred to as Multilevel Global Force (MGF), reduces running time by an average of 40% compared to the popular Barnes Hut Octree approximation method.

Secondly, optimisation methods used in static graph drawing (such as multilevel and approximation schemes) are adapted for use in dynamic graph drawing, simultaneously improving the quality of layouts produced and reducing the complexity such that large graphs with thousands of vertices can be drawn in interactive time.

Thirdly, several techniques are introduced for incorporating graph changes into the dynamic graph drawing to differing extents, allowing the viewer to decide whether the ongoing layout should preserve the original layout or prioritise the graph changes.

The works are combined to form an efficient multilevel dynamic graph drawing algorithm.

CONTENTS

1	Introduction	33
2	Literature Review	36
2.1	Overview	36
2.2	Definitions	36
2.3	Introduction.....	38
2.4	Static Graph Drawing	39
2.4.1	Multilevel refinement	43
2.4.2	Approximation.....	47
2.4.3	Graph Aesthetics	49
2.4.4	Overview of Static Graph Drawing.....	54
2.5	Dynamic Graph Drawing.....	54
2.5.1	Metrics for Graph Stability.....	58
2.6	Summary	60
3	Concepts.....	62
3.1	Overview.....	62
3.2	Problem Description	63
3.3	Story of Algorithms	64
3.4	Implementation Methodology.....	66
3.4.1	Configurations	66
3.4.2	Limitations.....	67
3.4.3	Benefits.....	67
3.5	Measuring Layouts	68
3.5.1	Local and Global Aesthetics.....	69
3.5.2	Aesthetics of Static Graphs	71

3.5.3	Metrics for Graph Stability.....	72
3.5.3.1	Layout Generation and Layout Adjustment.....	73
3.5.4	Summary of Aesthetics and Layout Quality.....	73
3.6	Multilevel Global Force (<i>MGF</i>)	74
3.6.1	Calculating Approximation	75
3.6.2	Structure of Approximation.....	78
3.6.3	Summary of Multilevel Global Force.....	80
3.7	Dynamic Graph Drawing.....	80
3.7.1	Visualisation	81
3.7.2	The Dynamic Graph	82
3.7.2.1	Graph Modification Operations	82
3.7.3	Dynamic Matching	85
3.7.4	Dynamic Modified Spring Embedder.....	88
3.7.5	Multilevel Layout	90
3.7.6	Approximation using <i>MGF</i>	92
3.7.7	Summary of Dynamic Graph Drawing.....	93
3.8	Summary.....	94
4	Multilevel Aesthetics.....	96
4.1	Analysis of Numerical Results	97
4.1.1	Global Layout Analysis	97
4.1.2	Local Layout Analysis of Partitions	98
4.1.3	Multilevel Analysis of Layout Generation	99
4.2	Subjective Analysis.....	100
4.2.1	Static Graphs	100
4.2.2	Development of Layout.....	103

4.3	Summary	105
5	Experimentation and Results: Static Graph Drawing	106
5.1	Test Data – Static Graphs	107
5.1.1	Static Test Graphs.....	107
5.1.2	Leafy Test Graphs	108
5.1.3	Layout Quality and Metrics.....	108
5.2	Multilevel Global Force (<i>MGF</i>)	109
5.2.1	Comparisons to state-of-the-art	109
5.2.1.1	Numerical Results.....	110
5.2.1.2	Subjective Analysis.....	113
5.2.2	Optimising the Multilevel Structure.....	115
5.2.2.1	Multimatching.....	116
5.2.2.2	Brute Matching	116
5.2.2.3	Pattern Processing.....	116
6	Experimentation and Results: Dynamic Graph Drawing.....	118
6.1	Test Data – Dynamic Graphs.....	119
6.1.1	Types of Operations	119
6.1.2	Frequency of Operations	120
6.2	Dynamic Spring Embedder.....	121
6.2.1	Configuration.....	121
6.2.2	Comparison of Spring Embedder Algorithms.....	121
6.2.2.1	Analysis of Numerical Results	121
6.2.2.2	Subjective Analysis.....	124
6.2.3	Layout Adjustment	126
6.2.3.1	Analysis of Numerical Results	127
6.2.3.2	Subjective Analysis.....	131

6.3	Multilevel Dynamic Modified Spring Embedder	133
6.3.1	Analysis of Numerical Results	133
6.3.1.1	Final Layouts	133
6.3.1.2	Metrics for Graph Stability	134
6.3.2	Subjective Analysis	136
6.4	Dynamic Modified Spring Embedder with Multilevel Global Force Approximation 140	
6.4.1	Configuration.....	140
6.4.2	Analysis of Numerical Results	140
6.4.2.1	Metrics for Graph Stability	141
6.4.3	Subjective Analysis	143
6.5	Dynamic Matching	145
6.5.1	Multilevel Matching Update.....	146
6.5.1.1	Quantitative Analysis.....	146
6.5.1.2	Subjective Analysis.....	149
6.5.2	Multilevel Global Force Updates	152
6.5.2.1	Quantitative Analysis.....	152
6.5.2.2	Subjective Analysis.....	156
6.6	Summary.....	160
7	Evaluation.....	162
7.1	Multilevel Aesthetic Analysis.....	162
7.2	Multilevel Global Force (<i>MGF</i>) Approximation	163
7.2.1	Running Time and Complexity	163
7.2.2	Layout Quality.....	164
7.2.3	Configuration.....	165
7.2.3.1	Limiting approximation	165

7.2.3.2	Coarsening Tolerance	165
7.2.3.3	Multimatching, Primitives and <i>MGF</i> Structure	166
7.2.3.4	Pattern Placement	168
7.2.3.5	General Graphs	169
7.2.3.6	Leafy Graphs.....	169
7.2.3.7	Overall Usage	170
7.2.3.8	Primitive Graphs	171
7.2.4	Summary of <i>MGF</i>	171
7.3	Dynamic Graph Drawing Methods.....	172
7.3.1	Dynamic Spring Embedder	173
7.3.1.1	Configuration of Displacement.....	173
7.3.1.2	Comparison of Spring Embedders	173
7.3.1.3	Layout Adjustment	174
7.3.2	Multilevel Generation.....	175
7.3.3	Multilevel Global Force	176
7.3.4	Dynamic Matching	177
7.3.4.1	Graph Structure.....	177
7.3.4.2	Frequency of Amendments	177
7.3.4.3	Fragmentation	178
7.3.5	Multilevel Updates	178
7.3.5.1	Multilevel Global Force Updates.....	179
7.4	Summary.....	181
8	Conclusions.....	184
8.1	Research Findings.....	184
8.1.1	Implementation.....	184

8.1.2	Algorithm Performance	185
8.1.3	Multilevel Global Force	186
8.1.4	Dynamic Graph Drawing.....	187
8.1.5	Dynamic Matching	188
8.1.6	Metrics for Graph Aesthetics and Graph Stability	188
8.2	Research Contributions	189
8.3	Limitations	190
8.4	Future Work.....	191
8.5	Concluding Paragraph.....	192
9	References	193
10	Appendix	200
10.1	Comparison of Running Time for Multilevel Spring Embedder with Multilevel Global Force and Multilevel Spring Embedder with Octree	200
	Multilevel Spring Embedder with Multilevel Global Force	201
10.1.1	Multilevel Spring Embedder with Octree Running Time	201
10.2	Coarsening Tolerance	202
10.2.1	Analysis of Numerical Results	202
10.2.2	Subjective Results	204
10.3	Correspondence with Daniel Danko	207
10.4	Configuring Multilevel Global Force	208
10.4.1	Numerical Results	208
10.4.2	Subjective Analysis	210
10.5	Configuring Dynamic Spring Embedder: Multilevel Global Force	213
10.6	Multilevel Aesthetics: Analysis of Layout Generation.....	223
10.7	Approximation Limit and Distance Limit in Calculating Repulsive Forces	227

10.7.1	Numerical Results	228
10.7.1.1	Approximation Limit.....	228
10.7.1.2	Comparison to R.....	230
10.7.2	Subjective Results	232
10.8	Brute Matching	234
10.8.1	Numerical Results	235
10.8.2	Subjective Analysis	236
10.9	Leaf Placement Scheme.....	238
10.9.1.1	Numerical Results	239
10.9.1.2	Subjective Analysis	243
10.10	Primitive Graph Coarsening.....	247
10.10.1	Numerical Results.....	249
10.10.2	Subjective Analysis.....	250
10.11	Configuring Dynamic Spring Embedder.....	253
10.12	Comparison of Layout Adjustment for Dynamic Spring Embedder and the Eades Spring Embedder Implementation	259
10.13	Dynamic Matching: Multilevel Updates	264
10.13.1	Small Graphs.....	264
10.13.1.1	Single, Middle and Full Update Comparison.....	264
10.13.1.2	Full and Rematch Update Comparison	269
10.13.2	Medium Graphs	274
10.13.2.1	Single, Middle and Full Update Comparison.....	274
10.13.2.2	Full and Rematch Update Comparison	279
10.14	Dynamic Matching: Multilevel Global Force Updates	284
10.14.1	Small Graphs.....	284
10.14.1.1	Single, Middle and Full Update Comparison.....	284

10.14.1.2	Full and Rematch Update Comparison	289
10.14.2	Medium Graphs	294
10.14.2.1	Single, Middle and Full Update Comparison.....	294
10.14.2.2	Full and Rematch Update Comparison	299
10.14.3	Large Graphs.....	303
10.14.3.1	Single, Middle and Full Update Comparison.....	303
10.14.3.2	Full and Rematch Update Comparison	307
10.15	Comparison of Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder	311
10.15.1	Comparison of Metrics for Graph Stability for <i>FRD</i> and <i>FRD</i> ML algorithms	311
10.15.2	Comparison of <i>FRD</i> and <i>FRD</i> -ML layouts (Subjective Analysis).....	314
10.16	Comparison of Graph Stability for Dynamic Spring Embedder and Dynamic Spring Embedder with Multilevel Global Forces – Metrics for Graph Stability.....	319
10.17	Animation and typical Frame rates for video mediums	322
10.18	Dynamic Matching: Impact of Graph Type on Multilevel Updates	322
10.19	Isomorphic Drawings	324
10.19.1	Numerical Results.....	325
10.19.1.1	Comparison of <i>MGF</i> to Octree.....	325
10.19.2	Isomorphic Layouts using <i>MGF</i>	325
10.19.3	Subjective Analysis.....	326
10.20	Multimatching	329
10.20.1	Numerical Results.....	329
10.20.1.1	Multimatching using the Degree of Vertices	330
10.20.2	Subjective Results.....	331
10.20.2.1	Multimatching Effect	331
10.20.2.2	Multimatching using the Degree of Vertices	333

10.21	Graphs	335
10.21.1	Leafy Graphs.....	335
10.21.2	Dense Graphs	335
10.21.3	Sparse Graphs	335
10.21.4	Regular Graphs	336
10.21.5	Known Layouts for Test Graphs.....	336
10.22	Implementation: Additional Information and Pseudocode	338
10.22.1	Graph Drawing and Experimental Framework.....	339
10.22.1.1	Data Structures	341
10.22.1.2	Multilevel Structure.....	342
10.22.1.3	Chaco and Operations Files.....	342
10.22.2	Multilevel Aesthetics	344
10.22.2.1	Generating Approximate Positions	344
10.22.3	Multilevel Global Force (<i>MGF</i>)	345
10.22.3.1	Multilevel Scheme.....	345
10.22.3.2	Spring Embedder with Multilevel Global Force	349
10.22.3.3	Multilevel Static Graph Drawing Algorithm	352
10.22.3.4	Primitives	353
10.22.4	Dynamic Graph Drawing.....	359
10.22.4.1	Dynamic Spring Embedder Adaptation	360
10.22.4.2	Multilevel Layout Generation	361
10.22.4.3	Dynamic Spring Embedder with Multilevel Global Force	363
10.22.4.4	Dynamic Matching.....	364
10.22.5	Summary	371
10.23	Multilevel Aesthetics: Detailed Results	371

10.23.1	Comparison of G0, and GN and partition layout quality.....	371
10.23.1.1	Layouts for the Modified Spring Embedder without Multilevel Refinement	371
10.23.1.2	Layouts for Modified Spring Embedder with Multilevel Refinement.	372
10.24	Dynamic Spring Embedder: Development of Layouts	373
10.24.1	Layout Generation	373
10.24.2	Layout Modification	375
10.24.2.1	Growth.....	375
10.24.2.2	Shrink	376
10.25	Comparison of Dynamic Spring Embedders.....	377
10.26	Comparison of Change in Edge Crossings for Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder	382
10.27	Multilevel Dynamic Spring Embedder: Development of Layouts.....	384

TABLES AND FIGURES

Figure 2.1. Example graph coarsening with matchings' highlighted by dashed lines (left) and a mapping of matchings' as a tree structure (right)	44
Figure 2.2 Interpolation and Refinement of graph layout between three graphs in a multilevel scheme using Force Directed Placement. From right to left, a coarsest graph with layout interpolates the positions of vertices to the finer graph, which is refined with FDP. The process is then repeated for the next two graphs.	45
Figure 2.3. Space decomposition (quadtree) of a simple graph, recursively splitting an area into four equal parts until each vertex exists in its own section (left), and accompanying tree structure recording the splits for use in calculation global interactions (right). Example usage of the tree for calculating repulsive forces	49
Table 2.1 A table of suggested aesthetic criteria which affect the readability of graph layouts as provided by Tamassia et al (1988), further information regarding the origins of these algorithms can be found in the publications	51
Table 2.2 Identification of literature empirically evaluating aesthetics expected to affect the readability of two dimensional graphs	52
Figure 3.2. A small general graph comprising of three densely connected clusters loosely connected to one another (left), showing an interpreted view of the "global layout" whereby the densely connected clusters are viewed as individual vertices with edges and reducing the complexity of the diagram.	70
Figure 3.3. The same graph of Figure 3.2 but coarsened using a multilevel scheme to form an approximation of the structure, simplifying it to provide a simplified layout similar to the authors "global layout". Dotted lines identify those edges which are collapsed to form the vertices in the coarser graph (right).	70
Figure 3.4. Two layouts of a 55x55 vertex grid, the layout on the left shows a significant fold with high numbers of edge crossings, whereas the right exhibits a lesser fold with far fewer edge crossings.	72

Figure 3.5. Given an example graph with initial random layout, it is expected the changes in edge crossings will gradually be reduced over time as Force Directed Placement pull vertices into more optimal positions, and if so, such reduction can be visualised and compared as a chart to determine if the efficiency of the drawing algorithm.73

Figure 3.7. An example of changes in positions between graphs, showing the difference between outdated positions generated by Force Directed Placement and an approximation of the position of vertices77

Figure 3.8. An example of Multilevel Global Force approximation in approximating global repulsive forces (left) and the usage mapped to the *MGF* tree (left). Repulsive forces are calculated between a vertex v , and coarse vertices (approximate positions) a , b and c . The result is reduced number of calculations.78

Table 3.1. Operations available for dynamic graphs in this research.....83

Figure 3.10. The graph 3025 with few (left) and many (right) edges removed, showing preservation of the global layout and degradation of local layouts.....84

Figure 3.11. The graph 3025 with the addition of two edges connecting opposite corners of the graph a large change in global layout. The graph appears three dimensional due to the folds, however, remains 2D.....84

Figure 3.12. Defragmentation of matching as a result of edge and vertex removal.....86

Figure 3.13. Given an example graph (in this case, part of the London Underground), and generation of a multilevel scheme using edge contraction (left), a resulting *MGF* tree is generated automatically (right).....87

Figure 3.14. No multilevel update for the addition of two vertices and two edges to the graph in Figure 3.13, showing the immediate matching of the two edges and their addition into the multilevel scheme by sharing the parent of the anchoring vertex87

Figure 3.15. Half-update of the multilevel scheme for the addition of two vertices and two edges to the graph in Figure 3.13, showing the incorporation of the vertices as typical

matchings up to the 3rd (mid) level, whereby they are joined with the parent of neighbouring matches (identified through traversal of the tree from the anchored vertex)87

Figure 3.16. Full update of the tree, and coincidentally the same structure as generated by an entirely new matching, of two vertices and two edges added to the graph in Figure 3.13. The tree is updated until an unmatched coarse vertex is identified with which the new matchings can be matched (found at the 4th level). If no matching is available, a new coarsest graph is generated and the matching is matched with itself.....87

Figure 3.17. Interpolation of vertex movement between two graphs in the multilevel scheme, incorporating the global layout achieved in coarse graphs (top) into finer graphs (bottom). Shapes are used to identify parent and child vertices between the two graphs such that two triangular vertices are represented by one triangular vertex in the coarse graph.91

Figure 3.18. Interpolation of movement as provided in Figure 3.17, incorporating the global layout achieved in coarse graphs (top) into finer graphs (bottom), however, due to positions of vertices in finer graphs not matching those in coarser graphs, interpolation of vertex movement results in poor placement in the coarser graph. Shapes are used to identify parent and child vertices between the two graphs such that two triangular vertices are represented by one triangular vertex in the coarse graph.....91

Table 4.1 Comparison of the number of edge crossings exhibited in the layouts, and approximation of layouts, for several test graphs. Two algorithms are used to generate the layouts; Fruchterman-Reingold Spring Embedder and Multilevel Fruchterman-Reingold Spring Embedder, with the latter utilising a multilevel scheme to improve global layout, reducing the number of edge crossings in the approximate layouts.....97

Table 4.2. Comparison of the quality of local layout, measured as the edge crossings exhibited in partitions of a graph generated by a multilevel scheme. Two algorithms are used to generate layouts for a selection of the test graphs, the Fruchterman-Reingold Spring Embedder and the Multilevel Fruchterman-Reingold Spring Embedder, expecting both to have similar local quality.....98

Table 4.3 Comparison of the range of edge crossings exhibited in partitions of a graph generated by a multilevel scheme. Two algorithms are used to generate layouts for a

selection of the test graphs, the Fruchterman-Reingold Spring Embedder and the Multilevel Fruchterman-Reingold Spring Embedder, expecting both to have similar local quality. ...98

Figure 4.1. Change in edge crossings for the layouts of the graph regular-medium provided by three algorithms, Eades Spring Embedder, Dynamic Fruchterman-Reingold Spring Embedder and a Multilevel Dynamic Fruchterman-Reingold Spring Embedder.99

Figure 4.2. Change in edge crossings for the development of approximate layouts of the graph regular-medium provided by three algorithms, Eades Spring Embedder, Dynamic Fruchterman-Reingold Spring Embedder and a Multilevel Dynamic Fruchterman-Reingold Spring Embedder. 100

Figure 4.3. A layout for the graph 3025 (left) generated by a Multilevel Fruchterman-Reingold Spring Embedder, and an approximation of the layout generated by a multilevel scheme (right) representing the global structure of the graph. 101

Figure 4.4. A layout for the graph 3025 (left) generated by a Multilevel Fruchterman-Reingold Spring Embedder exhibiting a severe twist in the layout, and an approximation of the layout generated by a multilevel scheme (right) representing the global structure of the graph showing a similar twist. 102

Figure 4.5. A layout for the graph 3025 (left) generated by a Fruchterman-Reingold Spring Embedder exhibiting poor global layout (folds and twists in the layout), and an approximation of the layout generated by a multilevel scheme (right) representing the global structure of the graph showing similar bends and twists of the layout. 102

Figure 4.6. Two layouts for the graph 3025, generated using the Fruchterman-Reingold Spring Embedder (left), and by the Multilevel Fruchterman-Reingold Spring Embedder (right), colored to show the partitions used to measure local layout. 103

Table 4.4. Figures showing the improvement of layout for the graph regular-medium, with multilevel interpretation of global layout showing the development of approximate global layout for the Multilevel Dynamic Fruchterman-Reingold Spring Embedder. 104

Table 5.1. A collection of static graphs used for experimentation of static graph drawing methods. The number of vertices, edges and the source of each graph are provided. Actual layouts for these graphs are provided in Appendix 10.21.5 107

Table 5.2. A collection of graphs known to have one or more "primitive leaf" vertices for experimentation with pattern recognition algorithms. The number of vertices, edges and leaf vertices for each graph are provided..... 108

Table 5.3. Comparison of running time for Multilevel Spring Embedder with Multilevel Global Force (*MGF*) or Octree (OT) approximation. The results show a significant 41% decrease in running time from usage of *MGF*, in addition to reduced maintenance running time between *MGF* Update and Octree Generation. Running time is shown in milliseconds. 111

Table 5.4. Comparison of edge crossings and range in edge length for layouts generated by Multilevel Spring Embedder with Multilevel Global Force (*MGF*) and Octree (OT) approximation. The results show that layouts generated using Multilevel Global Force (*MGF*) have an increase in edge crossings of 13.51%. 111

Table 5.5. Comparison of running times for the algorithms tested here (Multilevel Global Force (*MGF*) and Octree (OT)) to those published by Hu (2005). Running times are provided in seconds due to the formatting provided by Hu..... 113

Figure 5.1. Layout generated for 4elt using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left), Octree approximation (centre) and n-body force calculation without approximation (right). 113

Figure 5.2. Layout generated for 3025 using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right) 114

Figure 5.3. Layout generated for sierpinski10 using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right) 115

Figure 5.4. Layout generated for dime20 using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right). Note the difference of the Peripheral Effect, whereby edges for the Octree are more uniform leading to a more

expansive and “fluffy” layout, whereas the layout achieved with *MGF* shows noticeably more compression..... 115

Table 6.1. A collection of static graphs to be used as the base dataset for dynamic graphs for experimenting with dynamic drawing algorithms. The number of vertices and edges, the average degree, and a description regarding the structure of the graphs are provided..... 119

Figure 6.1. Layout provided to the graph regular-large using *FRD MGF* with single level multilevel update. The layout shows that added vertices and edges are not provided with low energy positions before the next set of operations occur, causing a build-up represented as a dense area on the graph (right). 120

Table 6.2 Number of edge crossings exhibited in layout after 300 iterations of *FDP*, comparing the Modified Spring Embedder of Fruchterman and Reingold (*FR*), the Eades Spring Embedder (*SE*), and the proposed Dynamic Spring Embedder (*FRD*) 122

Figure 6.2. Change in edge crossings exhibited in layouts provided to the collection of test graphs by Eades Spring Embedder (*SE*), Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder Cooling Schedule (*FRDC*) over 300 iterations of *FDP* application..... 122

Figure 6.3. Change in average vertex movement between frames, exhibited in layouts provided to the collection of test graphs by Eades Spring Embedder (*SE*), Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder Cooling Schedule (*FRDC*) over 300 iterations of *FDP* application 123

Figure 6.4. Layouts provided to the graph regular-small provided by the Eades Spring Embedder (left), Fruchterman-Reingold Spring Embedder (middle) and Dynamic Spring Embedder (right), after 300 iterations of *FDP* application 124

Figure 6.5 Reducing the Peripheral Effect of a layout by modifying the configuration of the algorithm..... 125

Figure 6.6. Layouts provided to the graph sparse-small provided by the Eades Spring Embedder (left), Fruchterman-Reingold Spring Embedder (middle) and Dynamic Spring Embedder (right), after 300 iterations of *FDP* application 125

Figure 6.7. Layouts provided to the graph <i>dense-small</i> provided by the Eades Spring Embedder (left), Fruchterman-Reingold Spring Embedder (middle) and Dynamic Spring Embedder (right), after 300 iterations of <i>FDP</i> application	125
Figure 6.8. Emergence of layout for the graph <i>regular-small</i> using the proposed Dynamic Spring Embedder. From left to right, layout after (A) 10 iterations, (B) 50 iterations, (C) 100 iterations, (D) 200 iterations, and (E) 300 iterations.	126
Figure 6.9. Change in edge crossings exhibited during layout generation for all test graphs during application of <i>shrink</i> operations, comparing the Dynamic Spring Embedder (<i>FRD</i>) and the Eades Spring Embedder (<i>SE</i>).....	127
Figure 6.10. Change in average vertex movement exhibited during layout generation for all test graphs during application of <i>shrink</i> operations, comparing the Dynamic Spring Embedder (<i>FRD</i>) and the Eades Spring Embedder (<i>SE</i>).....	128
Figure 6.11. Change in edge crossings exhibited during layout generation for all test graphs during application of <i>growth</i> operations, comparing the Dynamic Spring Embedder (<i>FRD</i>) and the Eades Spring Embedder (<i>SE</i>).....	129
Figure 6.12. Change in average vertex movement exhibited during layout generation for all test graphs during application of <i>growth</i> operations, comparing the Dynamic Spring Embedder (<i>FRD</i>) and the Eades Spring Embedder (<i>SE</i>).....	129
Figure 6.13. Change in edge crossings exhibited during layout generation for all test graphs during application of <i>maintain</i> operations set, comparing the Dynamic Spring Embedder (<i>FRD</i>) and the Eades Spring Embedder (<i>SE</i>)	130
Figure 6.14. Change in average vertex movement exhibited during layout generation for all test graphs during application of the <i>maintain</i> operations set, comparing the Dynamic Spring Embedder (<i>FRD</i>) and the Eades Spring Embedder (<i>SE</i>).....	130
Figure 6.15. Application of <i>shrink</i> operations – removal of vertices and edges – to the graph <i>regular-small</i> . From left to right, an initial layout is provided to the base graph (A), after which vertices are removed in staggered phases (B, C, D) and a final layout generated after	

all operations have finished in (E). Note that the layouts in (D) and (E) are the same, however, shrinking and compression occurs as a result of reduced repulsive forces. 131

Figure 6.16. Layouts for regular-small after application of *shrink* operations, as drawn using Dynamic Spring Embedder (left) and Eades Spring Embedder (right)..... 132

Figure 6.17. Application of *growth* operations – addition of vertices and edges – to the graph regular-small. From left to right, an initial layout is provided to the base graph (A), after which vertices are added in staggered phases (B, C, D, and E). Layout is achieved almost instantly as other vertices are being added, leading to a smooth visualization of the extension. 132

Figure 6.18. Layouts for regular-small after application of *growth* operations, as drawn using Dynamic Spring Embedder (left) and Eades Spring Embedder (right)..... 132

Figure 6.19. Layout generated for regular-small after application of the *maintain* amendments, whereby vertices become repelled so much they move large distances, resulting in the extended edge lengths and poor vertex placement seen in the image 133

Table 6.3. Comparison of edge crossings and range in edge lengths for layouts to a selection of graphs, generated by *FRDML* and compared to *FRD* to identify if layout quality improves for larger graphs (as a result of global layout being better achieved). 134

Figure 6.20. Change in edge crossings as layout is generated for general graphs of different sizes, exhibiting little difference between using Dynamic Spring Embedder (*FRD*) and Multilevel Dynamic Spring Embedder (*FRD ML*) 135

Figure 6.21. Comparison of the change in edge crossings exhibited in layouts for different sizes of graph when drawn using Multilevel Dynamic Spring Embedder 135

Figure 6.22. Change average vertex movement as layout is generated for general graphs of different sizes, exhibiting little difference between using Dynamic Spring Embedder (*FRD*) and Multilevel Dynamic Spring Embedder (*FRD ML*)..... 136

Figure 6.23. Comparison of layouts for the graph regular-large, generated using Dynamic Spring Embedder (left) and the Dynamic Spring Embedder with Multilevel scheme (right).
..... 137

Figure 6.24. Comparison of layouts for the graph sparse-large, generated using Dynamic Spring Embedder (left) and the Dynamic Spring Embedder with Multilevel scheme (right).
..... 137

Figure 6.25. Comparison of layouts for the graph regular-small, generated using Dynamic Spring Embedder (left) and the Dynamic Spring Embedder with Multilevel scheme (right).
..... 138

Table 6.4. Comparison of the improvement of layout for the graph regular-large over 300 iterations using the Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder , showing the improvement on global layout as a result of multilevel layout refinement... 139

Table 6.5. Comparison of running time required to complete one iteration of Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) for various test graphs 140

Table 6.6. Comparison of edge crossings and range in edge lengths exhibited by layouts generated by Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) algorithms for various test graphs 141

Figure 6.26. Change in edge crossings as layout is generated for general graphs of different sizes, exhibiting little difference between using the Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) .. 142

Figure 6.27. Change in average vertex movement as layout is generated for general graphs of different sizes, exhibiting a 20% decrease from using Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) in comparison to Dynamic Spring Embedder (*FRD*) 143

Figure 6.28. Comparison of layouts for the graph regular-small, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right), showing a fold in the layout as a result of the approximation 143

Figure 6.29. Comparison of layouts for the graph *regular-large*, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right)..... 144

Figure 6.30. Comparison of layouts for the graph *sparse-large*, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right)..... 144

Figure 6.31. Comparison of layouts for the graph *dense-medium*, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right)..... 145

Figure 6.32. Comparison of the change in edge crossings for incorporation of *shrink* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 147

Figure 6.33. Comparison of the change in edge crossings for incorporation of *growth* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 147

Figure 6.34. Comparison of the change in edge crossings for incorporation of *maintain* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 148

Figure 6.35. Comparison of the change in average vertex movement for incorporation of *maintain* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used 149

Figure 6.36. Comparison of layout generated for the graph *regular-medium* using Multilevel Dynamic Spring Embedder, using single-level (left) and mid-level (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout 150

Figure 6.37. Comparison of layout generated for the graph *regular-medium* using Multilevel Dynamic Spring Embedder, using high-level (left) and rematch (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout. 150

Figure 6.38. Comparison of layout generated for the graph *dense-small* using Multilevel Dynamic Spring Embedder, using single-level (left), mid-level (center) and high-level (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout 151

Figure 6.39. Comparison of layout generated for the graph *sparse-medium* using Multilevel Dynamic Spring Embedder, using single-level (left) and rematch (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout. 151

Figure 6.40. Comparison of the change in edge crossings for incorporation of *shrink* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 152

Figure 6.41. Comparison of the change in average vertex movement for incorporation of *shrink* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 153

Figure 6.42. Comparison of the change in edge crossings for incorporation of *growth* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 154

Figure 6.43. Comparison of the change in average vertex movement as a result of incorporating *growth* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used..... 155

Figure 6.44. Comparison of the change in edge crossings movement as a result of incorporating *maintain* operations into Multilevel Global Force approximation of the

dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used.....	155
Figure 6.45. Comparison of the change in average vertex movement as a result of incorporating <i>maintain</i> operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used.....	156
Figure 6.46. Comparison of layouts for sparse-medium using Dynamic Spring Embedder with Multilevel Global Force approximation after application of <i>maintain</i> operations, using mid-level (left) and high-level update methods, showing noticeable difference in the spread of vertices as a result of the methods.....	157
Figure 6.47. Layouts for regular-medium, showing the minor change in vertex positions as a result of the Multilevel Global Force structure being altered during the rematch update method. Edges are coloured based on their length, with the warping shown as edge lengths changing (and colours changing).....	157
Figure 6.48. Comparison of layout generated for the graph regular-medium using Dynamic Spring Embedder with Multilevel Global Force approximation, using single-level (left) and mid-level (right) update methods for incorporating amendments into the multilevel scheme and associated approximation.....	158
Figure 6.49. Comparison of layout generated for the graph regular-medium using Dynamic Spring Embedder with Multilevel Global Force approximation, using high-level (left) and rematch (right) update methods for incorporating <i>growth</i> amendments into the multilevel scheme and associated approximation.....	158
Figure 6.50. Comparison of layout generated for the graph sparse-small using Dynamic Spring Embedder with Multilevel Global Force approximation, using single-level (left) and rematch (right) update methods for incorporating <i>maintain</i> operation amendments into the multilevel scheme and associated approximations.....	159
Figure 6.51. Comparison of layout generated for the graph <i>dense-medium</i> using Dynamic Spring Embedder with Multilevel Global Force approximation, using high-level (left),	

rematch (center) and single-level (right) update methods for incorporating <i>growth</i> operation amendments into the multilevel scheme and associated approximation	160
Figure 7.1 Subtle warping of the graph 3025 as a result of regenerating the multilevel scheme for 're-matching' updates.....	180
Figure 10.8 Generalisation of edge crossings in layouts of multiple graphs for different approximation methods	209
Table 10.1 The number of edge crossings exhibited in a layout for a collection of graphs for differing values of the C parameter used in Modified Spring Embedders. The figures show that values in the range of 0.0005 and 0.05 provide fewest edge crossings for the development environment	209
Figure 10.9 Impact on changing the value of C on graph layout of 3elt	211
Figure 10.10 Approximation with differing scaling values (C) on the forces, showing the difference in layout as a result of the approximated forces, highlighting the need for configuration.....	212
Figure 10.11 Layout of data showing part of the graph outside of the approximation area	212
Figure 10.12 Comparison of the change in edge crossings for the graph sparse-medium over the course of 300 iterations using Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD).....	223
Figure 10.13 Comparison of edge crossings for the graph sparse-medium over 150 frames of iteration, after an initial 150 frames of force directed placement using Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD).....	224
Figure 10.14 Comparison of edge crossings in layouts for a coarsened abstraction of sparse-medium, showing the global layout as generated for Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD)	224

Figure 10.15 Comparison of the change in edge crossings for the graph *dense-small* over the course of 300 iterations using Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD).....225

Figure 10.16 Comparison of edge crossings in layouts for a coarsened abstraction of sparse-medium, showing the global layout as generated for Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD)226

Table 10.2 Table showing the maximum degree of a selection of test graphs, showing the choice of matching value for Brute Matching and the resulting number of graphs in the Multilevel Scheme using the value. The running time for generating the multilevel scheme is also provided.....235

Table 10.8 Which of the Primitive Graphs each of the test graphs are coarsened to, showing that many graphs coarsened down into the star primitive, with few coarsening into Ring or used standard FDP. None coarsened into a Chain.247

Table 10.9 The number of graphs generated in the multilevel scheme and the size of the coarsest graph, showing that even though primitives were used, the coarsest graph is still coarse247

Figure 10.41 Change in edge crossings 300 iterations of force directed placement of graph regular-small, showing the difference of the value tr (repulsive force displacement)253

Figure 10.42 Change in average vertex movement 300 iterations of force directed placement of graph regular-small, showing the difference of the value tr (repulsive force displacement)254

Figure 10.43 Change in edge crossings 300 iterations of force directed placement of graph sparse-small, showing the difference of the value tr (repulsive force displacement)255

Figure 10.44 Change in average vertex movement 300 iterations of force directed placement of graph sparse-small, showing the difference of the value tr (repulsive force displacement)255

Figure 10.45 Change in edge crossings 300 iterations of force directed placement of graph dense-small, showing the difference of the value tr (repulsive force displacement)	255
Figure 10.46 Change in average vertex movement 300 iterations of force directed placement of graph dense-small, showing the difference of the value tr (repulsive force displacement)	256
Table 10.12 Layouts given to the graphs regular-small, sparse-small and dense-small using tr values of '0.1, 0.3, 0.5, 0.7 and 0.9', showing the impact of changing the equilibrium of forces used in force directed placement. Layouts for sparse-small show that the force largely changes the clustering of leaf like vertices, with many of the graphs showing readable layout. Layouts for regular-small show compression and warping as forces are unable to escape each other, with a value of 0.5 providing highest quality layouts. Dense small shows to have very erratic movement for lower values of tr , with a n equilibrium preferred for generating symmetrical/equal layouts (lower values for tr show high movement as vertices swap between far and close from the centre of mass.....	258
Figure 10.47 Change in edge crossings for sparse type graphs as growth operations are applied and reverberated through the multilevel scheme using the four update methods.	322
Figure 10.48 Change in edge crossings for regular type graphs as growth operations are applied and reverberated through the multilevel scheme using the four update methods.	323
Figure 10.49 Average vertex movement measured in dense-type graphs during application of growth operations, showing that for each of the four update methods, the movement exhibits the same peaks	324
Figure 10.60 Actual layout for the graph 4elt	336
Figure 10.61 Actual layout for the graph data.....	336
Figure 10.62 Actual layout for the graph dime20	337
Figure 10.63 Actual layout for the graph mesh100.....	337
Figure 10.64 Actual layout for the graph sierpinski10.....	338

Figure 10.65 System Architecture detailing the four main components of the drawing algorithms generated here, indicating some of the functionalities available and providing context as to how components may be switched out to provide accurate comparison of techniques	340
Figure 10.66 An example of a dynamic drawing algorithm using the framework, indicating usage of operations, multilevel scheme, <i>MGF</i> approximation and the intended use of animation for display purposes and the frames. Force Directed Placement is left ambiguous, suggesting the drawing method is likely to change	341
Figure 10.67 Vertex data structure	342
Figure 10.68 Example of Chaco and XYZ files for a simple graph of four vertices	343
Table 10.19 Protocol and expected parameters of input for Graph Modification Operations	343
Figure 10.69 Generation of approximate positions of vertices in the multilevel scheme via post order traversal	344
Figure 10.70 Coarsener controller for generating graphs for a multilevel scheme	346
Figure 10.71 Coarsen method used to generate a coarse abstraction of a given graph	347
Figure 10.72 Function for matching vertices and identifying a maximal independent edge subset (MIES), a sub-process of coarsening	348
Figure 10.73 Function for incorporating edges of a finer graph G_i into a coarse approximation of that graph G_{i+1} , a sub-process of coarsening	349
Figure 10.74 Spring Embedder with Multilevel Global Forces for approximating repulsive forces	351
Figure 10.75 Inclusion of an approximation into repulsive force calculation using <i>MGF</i>	352
Figure 10.76 Multilevel Static Graph Drawing Algorithm design	353

Figure 10.77 Incorporation of primitive processing into the coarsening example provided in Figure 10.70.....	354
Table 10.20 Methods for identifying primitives through pattern recognition.....	355
Figure 10.78 Method for pattern coarsening of leaf vertices and chains (primitives).....	356
Table 10.21 Methods for layout calculation for primitive graphs and shapes	358
Figure 10.79 Alteration of Multilevel Static Drawing algorithm to include Primitive processing.....	359
Figure 10.80 Dynamic Spring Embedder, an adaptation of the Fruchterman-Reingold (1991) static spring embedder for use in Dynamic graph drawing.....	361
Figure 10.81 Multilevel algorithm for layout generation and adjustment using the Dynamic Spring Embedder Adaptation	361
Figure 10.82 Method for calculating initial positions of vertices in a multilevel scheme, ensuring vertices are near their coarser representations in the multilevel scheme.....	362
Figure 10.83 Alteration of repulsive force calculation in the Dynamic Spring Embedder for use of Multilevel Global Force.....	363
Figure 10.84 Alteration to post order update of approximate vertex positions.....	364
Table 10.22 Implementation of graph modification operations for dynamic graphs.....	365
Table 10.23 Adaptation of graph modification operations for use as multilevel updates.....	367
Figure 10.85 Caretaker method used to maintain the multilevel scheme after operations have been performed.....	368
Figure 10.86 Caretaker sub process - check whether any matches are available for any unmatched vertex.....	368

Figure 10.87 Caretaker sub process - check if a vertex has been provided with a parent, if not, check any adjacent vertices for a parent with only one child, or create a new vertex in the next coarsest graph to act as a parent.....369

Figure 10.88 Caretaker sub process - identify any vertices in the coarse graphs which no longer represent vertices in the original, and remove them.....369

Figure 10.89 PreserveEdges function used to check the preservation or the removal of edges throughout the multilevel scheme370

Figure 10.90 Join method for use with multilevel update methods, ensuring graph modifications are joined to the existing multilevel scheme371

1 Introduction

In our modern data driven society, how do we keep up with understanding and analysing such vast amounts of information? Since the turn of the millennium, the World Wide Web has exploded in popularity, not just for offering countless sources for facts and figures but as the basis of ever-expanding social networks, passing data through millions of nodes on a daily basis. Like this, much of the data that supports the everyday workings of modern lives has been increasing, and with it, the demand for visualisation in meaningful and expressive ways.

With the amount of data growing, the methods for visualising the data continue to develop, graph drawing being no exception. The research compiled in this thesis aims to investigate;

How can the complexity and cost of dynamic graph drawing methods be reduced such that large dynamic graphs with around 10,000 vertices can be drawn as a smooth animation, with minimal cost to the quality of layouts?

To answer the question, the history of graph drawing algorithms is researched in the **Literature Review**, annotating the evolution of graph drawing algorithms and helping understand the difficulties and weaknesses researchers have overcome in the area. In particular, interest is given to Force Directed Placement (FDP), one of the more popular methods for drawing graphs which offers a variety of optimisation methods.

The review investigates and implements such research aiming to circumvent the expensive calculations associated with the FDP methods and improving the readability of graph layouts, particularly for larger graphs. Initially, interest is given to Static Graph Drawing from which many Dynamic Graph Drawing algorithms have evolved, and in which many optimisations exist which adapt the methods to huge graphs with millions of vertices (in particular, multilevel schemes and approximation).

Dynamic graph drawing algorithms are next visited, with interest in the problems encountered with complexity and the solutions used to overcome them. Comparisons are made to Static Graph Drawing to show how common weaknesses have been overcome and how they can possibly be resolved in dynamic graph drawing.

Surprisingly, there has been little to no research in applying the solutions of static graph drawing to dynamic graph drawing in the context of drawing large graphs. Implementation of a few select published algorithms, notably the Multilevel Spring Embedder and Barnes Hut Approximation, help cement a foundation for optimising dynamic graph drawing algorithms. These investigations and implementations are described in the contributions (**Concepts**, Chapter 3), and are used to illustrate the concepts for a more efficient approximation method (**Multilevel Global Force**, Section 3.6) and the adaptation of static graph drawing methods to Dynamic Graph Drawing (**Dynamic Modified Spring Embedder**, Section 3.7) which is used to bring the solutions of one area to another, overcoming weaknesses such as minima and complexity.

Being able to apply the solutions for one problem to another is only half of the work however. Dynamic graphs are different to static graphs in that they change over time, and the contributions above need to respect this. **Dynamic Matching** (Section 3.7.3) is a potential solution described here, detailing how the dynamic changes are applied and how they can be interpolated through multilevel and approximation schemes to differing extents, offering visualisation of these changes which can emphasise the changes to a graph or preservation of the initial layout.

Various attempted optimisations of these contributions are also described in contributions, and an analysis and comparison of them is described in the **Experimentation and Results** (Chapters 4 5 and 6).

Measuring Layouts is inherently difficult however. Graph drawings are subject to each individual's own perceptions which are unlikely to be universal, and so to provide a more literal comparison, aesthetics described in Literature as constituting “good layout” (which optimises the amount of data that can be extracted from a drawing) are used as metrics for measuring the readability.

Through these metrics, the contributions are experimented and numerical results collected as part of a large **Experimental Framework** (Section 3.4) providing evidence for **Evaluations** (Chapter 7) of the suggested concepts and additional optimisations.

Comparison is of importance, especially in the context of state-of-the-art methods described in Literature, and so each of the works are initial compared to an existing algorithm to better show their strengths and weaknesses.

These thoughts are summarised in the **Conclusions** (Chapter 8) of the thesis, indicating which of the algorithms can be used to optimise Dynamic Graph Drawing algorithms for calculating layout for larger graphs in a readable and helpful visualisation, and the potential use cases where the algorithms and optimisations are best suited.

2 Literature Review

2.1 Overview

This chapter reviews literature in the area of graph drawing and sets the scope of interest for the contributions presented in this thesis. In order to properly introduce the area and the state-of-the-art methodologies used within the field, the literature review is split into the following sections:

The discussion starts with some **Definitions** of common terminology (Section 2.2).

An **Introduction** (Section 2.3) is given to present the area and provide context for the literature, narrowing the scope of interest to that which is applicable to the contributions made here. The research method used to identify appropriate literature is also described.

Static Graph Drawing (Section 2.4) looks at the evolution of layout generation algorithms, from their introduction for visualising small data collections to the current state-of-the-art algorithms used to display huge datasets (greater than 100,000 vertices), identifying the problems faced, how they are overcome and the optimisation methods used.

Dynamic Graph Drawing (Section 2.5) focuses on layout adjustment techniques for collections of data which change over time, aiming to optimise the visualisation of changes to a graph while maintaining a readable layout. A description of the typical problems encountered and the methods of visualisation used to maximise readability are also given.

Data visualisation is the primary goal of Graph Drawing, therefore a brief detailing of **Graph Aesthetics** (2.4.3) and **Metrics for Graph Stability** (2.5.1) are provided within the subsections above to identify the criteria constituting “good” and “bad” graph layout.

A brief **Summary** (Section 2.6) concludes the notes in the subsections above.

2.2 Definitions

Some definitions of common terminology are given below.

A Graph is a geometric representation of a relational dataset, typically created with the aim of providing improved understanding of data compared to observation of the raw data. Graphs which do not change over time are referred to as **static graphs**, in contrast, **dynamic graphs** change over time and require continuous layout adjustment to visualise changes. Although drawing algorithms are often optimised for one or the other, there is overlap between the areas, with much of the literature for dynamic graphs originating in static graph drawing.

Throughout this thesis, graphs are referred to in two parts;

- The **graph** is a model of the relational data, describing information about the vertices and the edges connecting them and does not require a layout (such that discussion regards the data and relationships within)
- The **graph layout** is the visual representation that is given a graph to help a reader extract information from the data, whereby vertices are given some position in a plotting and drawn with line segments between them (representing edges)

A general graph is a graph which has no attributed “type”, that is to say it does not follow drawing conventions of any specific type of graph (such as planar or hierarchical graphs). A general static graph, G , is defined as a set of vertices, V , and a set of edges, E , whereby each edge connects two vertices.

In the works here, a graph layout is a physical record of the positions each vertex has been given for a layout, for example, in a Cartesian coordinate system, a layout will be a list of x , y and z positions describing each vertex.

For the purposes of this thesis the layout of a graph refers to its visualisation, with vertices represented as point positions in some drawing area, and edges drawn as straight line segments between connected vertices. During description and evaluation, layout is referred to in two ways;

- **Local layout** refers to the layout of vertices and edges at a detailed level, with more regard to the placement and proximity of vertices and edges (given a map for example, this would be analogous to concentrating on individual buildings and roads between them)

- **Global layout** refers to the layout as a whole, perceiving the graph as the sum of its parts with more regard to overall shape and structure (using the same map example, this would be analogous to viewing the city as a whole with interest in neighbourhoods more than individual houses)

Dynamic graphs follow the same definitions with the addition of a time component, allowing for changes to the graph over time. The visualisation of the layout adjustment is usually drawn as an on-line process (see for example Veldhuizen, 2007), in that changes are observed as they happen through use of animation or time slices.

The term **meaningful** is used to describe graph layout which is deemed appropriate enough to represent the information provided, as opposed to a layout which does not help the understanding of the data visualised. Although subjective, extensive research has been undertaken (see below) to determine which attributes of layouts make for more meaningful visualisations. Aesthetic criteria have evolved from these and are optimised in many drawing algorithms, aiming to improve the readability of graph layouts (see Section 2.4.3).

For comparison of drawing methods, it is commonplace to identify:

- the complexity of suggested algorithms
- analysis of the running time
- subjective analysis of the layouts

Throughout the document graphs may be considered large or huge, which refer to the graph having greater than about 10,000 and 100,000 vertices respectively.

2.3 Introduction

Graph drawing is the task of calculating a layout for a Graph which would otherwise not have a visual representation. Force Directed Placement is one of the most popular approaches to graph drawing (see Kaufmann and Wagner (2001), Battista et al (1994)) and is the primary method investigated here. Other techniques such as Spectral, Tree and Orthogonal layout methods are also widely used, often optimising for specific criteria such as the ordering of vertices or use of multi-line edges (see Tamassia et al, 1988).

Originally used in VLSI and chip placement, Force Directed Placement was first introduced into static graph drawing and quickly expanded to dynamic graph drawing (Misue et al, 1995), providing a bridge between research fields and applying known methods to a largely unvisited area.

The method models a graph as a mechanical system under stresses which iteratively move the vertices of a layout into positions which reduce the overall stress of the system.

The intention of the research reviewed here is to investigate static graph drawing algorithms with particular reference to the algorithmic modifications required to deal with larger graphs, identifying some of the problems faced with large datasets and the state-of-the-art optimisation techniques currently employed to overcome them. Graph aesthetics are also visited to define what a ‘good’ graph layout is.

Investigation of Dynamic graph drawing algorithms follows with interest in their differences to static graphs, why few methods have been applied to large datasets and what optimisations (if any) exist. Given the communities interest in preserving a reader’s perception of dynamic graphs, Metrics for Graph Stability are also investigated.

2.4 Static Graph Drawing

Static graph drawing refers to the drawing of unchanging static graphs, for which Force Directed Placement (*FDP*) is one of the most popular layout procedures (Kaufmann and Wagner (2001), Battista et al (1994)) providing high quality results, simple heuristics, and interactivity which can be applied to both static and dynamic graph drawing. The approach describes the modelling of forces which iteratively change the placement of vertices and reduces an energy function relating to the layout of a graph. A graph with high energy is expected to have poor layout, and through minimisation of this energy, the layout can be improved (Eades (1984), Kamada and Kawai (1989)). Minimisation of the energy function is an iterative process which typically involves moving vertices into positions which lower the graph energy.

The Spring Embedder (Eades, 1984) is one of the most common approaches for this, which models a graph as a mechanical system with vertices represented as freely moving rings

which repulse one another, and edges as springs which pull related vertices closer to one another. The two forces are:

$$\text{A spring force, } f_a(\Delta) = k \log \Delta$$

$$\text{and a repulsive force, } f_r(\Delta) = \frac{k}{\Delta^2}.$$

The spring force is an approximation on the properties of springs and keeps adjacent vertices uniform distances apart. If the distance, Δ , between connected vertices is too large, the spring will contract bringing the two vertices closer together, conversely, if too close together the spring force will push vertices apart. The repulsive force mimics an inverse square law applied between all pairs of non-adjacent vertices, pushing them apart. The closer the vertices are to one another, the more violent the separation. Both forces use an ideal edge length k and work against each other to move vertices into equilibrium positions between them.

The energy of the system is calculated as the total force between all vertices. From an initial random positioning, the system is “let go” and iteratively improved, using the forces to displace vertices and reduce the energy of the system. After some number of iterations, the energy of the system is expected to have reduced to such an extent that vertices have minimal further movement, at which point the layout can be said to have converged.

In practice, Eades determined that most graphs with less than 30 vertices are drawn well in under 100 iterations and remarked that many layouts can be generated with fewer. For many of the examples given, the layouts are shown to be aesthetically pleasing, exhibiting few edge crossings, uniform edge lengths and uniform density of vertices across the viewing area. Some examples show the difficulty of generating layout for dense graphs where the number of edges prevent expansion and so resulting in dense “busy” layouts with high numbers of edge crossings. In addition, larger graphs require significantly more time to suitably expand the layout, often becoming trapped in local minima as parts of the graph are unable to pass one another (Eades (1984), Fruchterman and Reingold (1991)).

Local minima refer to configurations of vertices where lower energy layouts exist but require vertices to move into higher energy positions before reaching them. The spring embedder

does not allow for “up-hill” moves as they increase energy, thus the layout becomes trapped in “local” minima and converges with a layout which may not be optimal (Davidson and Harel, 1996).

Various modifications to the heuristic exist. Kumar and Fowler (1994) detail a spring embedder for generating three dimensional drawings. Sugiyama and Misue (1995) describe the use of magnetic springs to control orientation of edges to achieve certain aesthetic criteria. Huang et al (2010) suggest inclusion of additional forces to expand minimum angles between coincident edges in order to provide compromise over achieving aesthetic criteria. Bannister et al (2013) provide an added constraint to incorporate social gravity in force calculation for visualisation of social networks.

Kamada and Kawai (1989) suggest an adaptation to the method, replacing repulsive forces with springs between all vertices with length equal to the “graph theoretic” distance between vertices. The total energy of the system is equal to the total energy of the springs. The minimisation of this energy is achieved through estimating a vertex’s position until the energy of forces acting upon it are reduced.

Each vertex, $v_m (x_m, y_m)$, is chosen based on its energy, Δ_m (as calculated below), and ends when this value becomes less than some tolerance (where E is the energy of the system):

$$\Delta_m = \sqrt{\left\{ \frac{\partial E}{\partial x_m} \right\}^2 + \left\{ \frac{\partial E}{\partial y_m} \right\}^2}$$

The method generates high quality layouts exhibiting symmetry, minimisation of edge crossings and a high level of congruence between isomorphic graphs (it is commented by the authors that Eades’ method does not provide such congruence), but suffers higher complexity than the Eades spring embedder with $O(n^3)$ calculations.

An alternate to the Spring Embedder is a method which simulates the heat treatment of materials seen through annealing. Davidson and Harel (1996) use simulated annealing to model graphs as amorphous solids, using temperature to represent the movement of particles in the material. The particles begin with high temperature and are "cooled" to reduce their movement and form an ordered crystalline structure synonymous with high quality layout.

The temperature provided by Davidson and Harel (1996) represents the energy of vertices in the layout, with higher energies allowing for larger movements. An initial temperature is generated through random placement of vertices, and vertices are iteratively given new positions called “configurations”. Each configuration of vertices is estimated and changes the temperature of the graph, if lower, the configuration is accepted, otherwise acceptance is probabilistic, allowing for higher temperature configurations to be accepted (offering chances to overcome local minima).

Vertex movement is limited by a cooling schedule, allowing for larger displacement in earlier iterations which is steadily reduced over time, “cooling” the layout.

The number of iterations and consequent running time is dependent on the values given to the cooling schedule, with rapid cooling finishing quickly but generating layouts with imperfections (such as edge crossings or coincident vertices), or slow cooling improving the aesthetic quality of the layout but taking longer.

The method is flexible, boasting a selection of cost functions to optimise different aesthetic criteria, which can be used for beautification and refinement of layouts (Harel and Sardas, 1995). Modifications for optimising three dimensional drawings is provided by Cruz and Twarog (1996).

Combination of Simulated Annealing and the Spring Embedder is presented by Fruchterman and Reingold (1991), using the control over vertex movement to overcome local minima in the spring embedder. The implementation allows for high energy vertex positions in earlier iterations of the algorithm, subsequently reducing the movement until it is less than some tolerance as described above (Davidson and Harel, 1996). In contrast, Eades method suppresses the movement of vertices by a percentage of the movement.

Further optimisation is achieved through limiting the distance at which repulsive forces take effect, and the use of a grid to approximate repulsive forces and further reduce the complexity involved. The algorithm is said to run very quickly and provide equivalent layouts to other methodologies, however, as the size of graphs increases, it becomes more difficult to generate high quality layouts as conflicting forces push and pull vertices into sub-optimal positions.

Harel and Koren (2002) provide an alternative to grid approximation methods through embedding a number of additional dimensions evenly throughout a graph and using the relative positions of vertices (through principal component analysis) to estimate their projected placement into a two or three dimensional plane. The use of high-dimensional embedding removes the n-body calculations, providing an opportunity to draw larger graphs. The number of dimensions used is dependent on the user with higher values extending the runtime and lower values decreasing layout quality; a value of 50 is used by the authors.

An alternate method of graph drawing using stress majorisation described by Gasner et al (2005). The method builds on work by Kamada and Kawai (1989), which iteratively solves a system of linear equations using the Conjugate Gradient (CG) method, to reduce the stress of the system. The approach is shown to reach energy minima much quicker than the Kamada and Kawai method, and offers drawings for large graphs with over 10,000 nodes quickly.

2.4.1 Multilevel refinement

Many of the models and algorithms described for static graph drawing are limited by the size of the graph they can visualise, facing problems in overcoming local minima in large datasets and exponentially increasing running time. In order to overcome such problems, multilevel and multiscale schemes can be used to simplify the structure of a graph, reducing its size such that the methods above can be applied to provide improved global layout.

A multilevel scheme is described by Walshaw (2003) stemming from the author's previous work in graph partitioning (Walshaw et al, 1997). The scheme takes a graph as its input and generates a collection of coarser interpretations through identification and collapse of a maximal independent edge subset (MIES) (Hendrickson and Leland, 1995) to which *FDP* can be applied. The independent edges are contracted to form "coarse" vertices with coincident edges preserved forming a coarser representation of the graph. The process is repeated until a graph with two vertices is generated, or the difference between levels (graph size) is less than some tolerance.

Given an original graph G_0 , the hierarchy of graphs can be defined as $G_L = \{G_0, G_1, G_2 \dots G_n\}$, each with a set of vertices and edges such that $G_1 = \{V_1, E_1\}$. Each graph which is

generated will have fewer vertices, with G_n having the least (see Appendix 10.22.3.1 for pseudocode).

Figure 2.1 provides an example of multilevel generation for an example graph G_0 :

- Independent edges (denoted as dotted line segments) are identified;
- These are then contracted, merging the connected vertices into one forming vertices of a new graph G_1 (reducing the number of vertices from 10 to 7);
- Edges of the initial graph are preserved and brought into the new graph, connecting the new vertices (approximating the relationships of the G_0);
- The process is then repeated on each subsequent graph generating a collection of increasing abstract representations of the original until a graph of 2 vertices is found.

The matching of vertices used to generate each level of the multilevel scheme can be represented as a tree showing which vertices in a finer graph are represented by a vertex in a coarser graph.

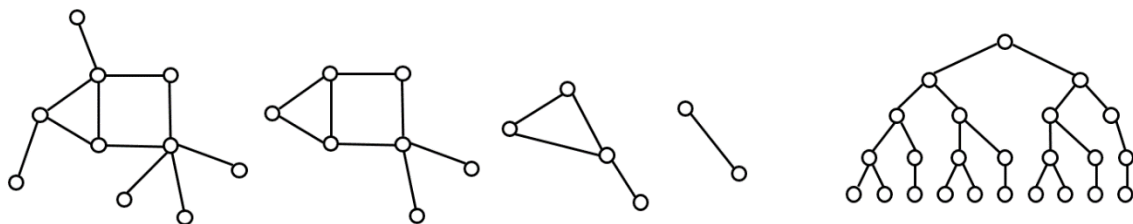


Figure 2.1. Example graph coarsening with matchings' highlighted by dashed lines (left) and a mapping of matchings' as a tree structure (right)

The process for generating a multilevel layout then works backwards from the coarsest graph:

- Force directed placement is applied to the coarsest graph, G_n ;
- The generated layout is interpolated to the next coarsest graph, G_{n-1} , by passing the position of a vertices to the matched children vertices it represents, giving the graph an initial layout;
- FDP is applied to G_{n-1} to refine the layout;
- The process is repeated on each subsequently finer graph until the original graph, G_0 , is given a layout, as illustrated in Figure 2.2.

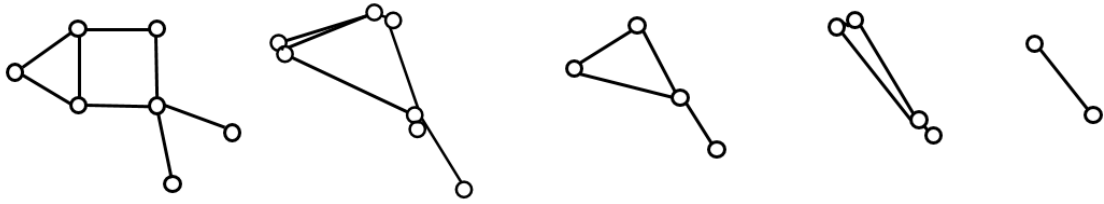


Figure 2.2 Interpolation and Refinement of graph layout between three graphs in a multilevel scheme using Force Directed Placement. From right to left, a coarsest graph with layout interpolates the positions of vertices to the finer graph, which is refined with FDP. The process is then repeated for the next two graphs.

For the application of *FDP* in each of the coarser graphs, the ideal edge length k must change in order to provide a layout which fits into the space currently occupied by the graph. Each vertex in a coarser graph may represent many vertices in finer graphs, and so edge lengths must be amended to avoid layouts being squashed. Walshaw suggests a simple heuristic to calculate k for each level:

$$k_l = \sqrt{\frac{4}{7}} \cdot k_{l+1}$$

Where k_l is the ideal edge length of the current finer graph, and k_{l+1} is the ideal edge length calculated for the previous graph. For the coarsest graph, the value is calculated as:

$$k_L = \frac{1}{|E_L|} \sum_{(u,v) \in E_L} \|(u,v)\|$$

The drawings generated exhibit high quality global layout and untangling, and reduce running time by reaching a high quality layout sooner (graphs with hundreds of thousands of vertices are reportedly drawn in several minutes). Both global and local layout are achieved well, effectively overcoming the problems with minima. An implementation of the algorithm running on a GPU can further reduce running time by a factor of up to 20 (Frishman and Tal, 2007).

A similar method of multi-scale layout is provided by Hadany and Harel (2001), describing an alternate method of generating the collection of coarser graphs. Identification of a

maximal independent edge subset is dropped in favour of a cost function determining which edges should be contracted through analysis of their attributes (clustering, degree and homotopic numbers). The authors aim to better preserve a graph's attributes and proportions, achieving more accurate representations of the original graph.

The force directed placement algorithm chosen by Hadany and Harel (2001) uses a relaxation scheme that employs a steepest descent minimisation procedure to find improved vertex positions and reduce the energy of the graph. Unlike Walshaw's approach, this is first applied to the original graph, and interpolated through the coarser graphs, after which it is interpolated back through the finer graphs again, refining the layout.

Further research from Harel and Koren (2001) details an alternate implementation of the algorithm with the priority on minimising running time instead of optimising layout.

An approach embedding multiple dimensions is provided by Gajer et al (2001), describing a method for coarsening a graph through identifying maximal independent vertex subsets, and using the positions of these vertices to place vertices intelligently into optimised initial positions, maximising the angle size between adjacent vertices. The authors describe that vertices will be placed in initial positions close to their final positions, such that they can be refined using Force Directed Placement.

A spectral method of drawing graphs is described and implemented by Koren et al (2002) through the use of multiscale matrices and eigenvector computation. ACE (Algebraic multigrid Computation of Eigenvectors) is an approach which, like previous papers, uses multiple levels of abstraction to solve an energy minimisation problem and interpolates a solution through the coarser levels, refining the solution whilst doing so.

ACE treats the minimisation of a graph's energy as an eigenvector problem, and uses the eigenvectors of the Laplacian and mass matrices of a graph to cluster vertices together in a multiscale fashion (using an interpolation matrix to move between levels). An iterative power method variant is used to find the lowest eigenvalue of a coarsened matrix, and the corresponding eigenvector. This solution is then used to partially solve the next level, which once solved becomes the partial solution to the next level and so on, being applied iteratively until a solution to the original problem is found.

An alternate method of multiscale layout is suggested by Hachel and Jünger (2005). The Fast Multipole Multilevel Method (FM3) uses a method of coarsening “solar systems” to coarsen vertex clusters, choosing a ‘sun’ with neighbouring vertices regarded as ‘planets,’ each with ‘moon’ nodes. Each solar system is then collapsed to form coarser nodes. The use of ‘solar systems’ allows for initial placement of the vertices relative to their solar system (local layout) which can be refined using *FDP*. Force directed placement is applied to refine the layout.

2.4.2 Approximation

Even with overcoming minima, Force Directed Placement still suffers from high running times associated with the iterative calculation of forces between all pairs of vertices (for example Eades, 1984), emulating an n-body problem and making the techniques especially costly for large graphs. In dynamic drawing, the result is an increase in the time required to generate each frame of a visualisation, resulting in a staggered animation of the drawing process.

Algorithms typically overcome this complexity by reducing the number of “global force” calculations through limiting the distance at which forces take effect and/or approximation. These approximations take the layout of a graph, and imagines some structure (typically a grid) over the layout, treating the vertices in one section as a singular vertex with average position and weight.

A popular method achieving this is the Barnes Hut Octree (Barnes and Hut, 1986), which has been effectively applied to multilevel force directed placement (Hu (2005), Hachel and Jünger (2005)), improving on a grid based system used previously (Fruchterman and Reingold (1991), Walshaw (2003)).

The octree is generated through a recursive strategy:

- Given a layout, calculate a bounding box that covers the entire layout;
- Split the bounding box into 4 or 8 sections (for 2D or 3D dimensions respectively);
- Iterating over each section, if there is a section with more than one vertex within, split that section into 4 or 8 sections;

- Continue on each section until each vertex occupies its own section or some limit is reached;

The splits are recorded in a tree representing the proximity of vertices in the drawing area as shown for an example graph in Figure 2.3. The generated octree structure is used by:

- For a vertex in the graph, which inhabits its own ‘section’ of the octree, calculate repulsive forces between itself and vertices in neighbouring sections (point *a* in Figure 2.3);
- If a neighbouring has more than 1 vertex, the average position of the vertices within the section is used and treated as a single point with which repulsive force is calculated;
- The repulsive forces are combined to give one direction;
- Once finished at this level, move to the parent section which contains the current section and its siblings;
- Repeat the process with the siblings of the parent section (points *b*, *c* and *d* in Figure 2.3);
- Continue until the bounding box (root of the Octree) is reached, then move on to the next vertex.

Usage results is a drop in complexity from n^2 to $n \log n$, reducing the running time of algorithms but requiring the space decomposition structure be generated in addition to the graph, requiring maintenance to maintain accuracy.

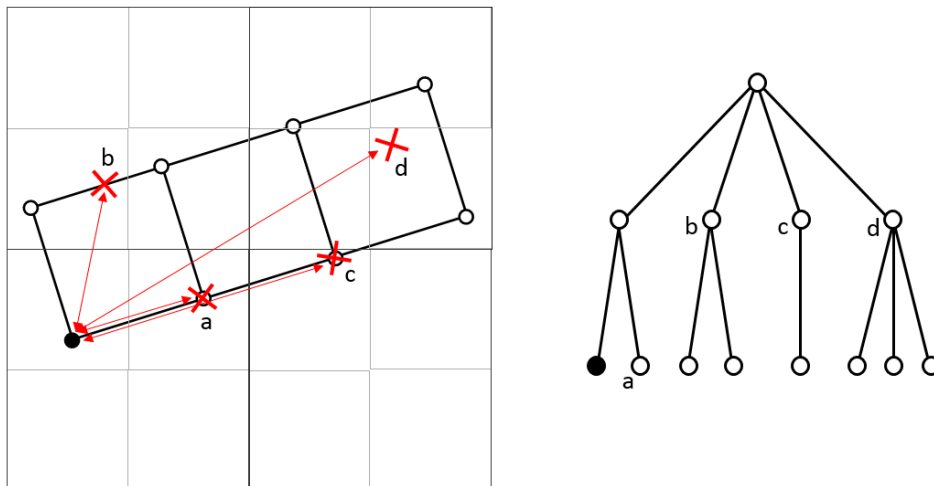


Figure 2.3. Space decomposition (quadtree) of a simple graph, recursively splitting an area into four equal parts until each vertex exists in its own section (left), and accompanying tree structure recording the splits for use in calculation global interactions (right). Example usage of the tree for calculating repulsive forces

In addition, Hu describes an issue named the “peripheral effect” observed in the drawings, whereby the outer edges of a graph are more compressed than the inner edges (as a result of the forces). A resolution is suggested which increases the powers within the repulsive force function: changing K^2/d to K^3/d^2 , and means that the spring forces are more likely to overcome the effect of repulsive forces, promoting uniform edge length.

Layouts for huge graphs are generated much more quickly than Walshaw and with comparable quality. In addition, Hu describes an issue named the “peripheral effect” observed in the drawings, whereby the outer edges of a graph are more compressed than the inner edges (as a result of the forces). A resolution is suggested which increases the powers within the repulsive force function: changing K^2/d to K^3/d^2 , and means that the spring forces are more likely to overcome the effect of repulsive forces, promoting uniform edge length.

2.4.3 Graph Aesthetics

Graph aesthetics are the attributes of a layout which impact the amount of information an observer can absorb, and is one of, if not the most important aspect of information visualisation (Tamassia et al, 1988). Many methods and algorithms exist to provide improved and adaptable ways of achieving optimal readability of datasets, however, due to

the subjectivity of layout analysis, measurable comparison of layout quality is difficult. This in turn makes analysis and comparison of drawing methods difficult.

Definitions of aesthetics and various graph drawing methods pertaining to maximise one or more of these are identified by Tamassia et al (1988). The research gives a review of various specialist and general graph drawing algorithms, with an analysis of the aesthetics that each optimises and a description of standards within the area. The aesthetics are collected from discussions in a wide range of earlier graph drawing publications (a taxonomy of which can be found in the publication).

A list of the aesthetics described are given below in Table 2.1, each with a category identifying whether the aesthetics affect **global** (entire graph) or **local** (some small part of a graph) layout, and which graph type each is most relevant to (**Flat** referring to general graphs and **Hierarchical** referring to tree type graphs).

Aesthetic	Category
Minimisation of the area occupied by the drawing	Global/Flat
Balance of the diagram with respect to the vertical or horizontal axis	Global/Flat
Minimisation of the number of bends along the edges	Global/Flat
Maximisation of the number of faces drawn as convex polygons	Global/Flat
Minimisation of crossings between edges	Global/Flat
Vertices with high degree in the centre of the drawing	Local/Flat
Minimisation of the differences among vertices dimensions	Global/Flat
Minimisation of the global length of edges	Global/Flat
Minimisation of the length of the longest edge	Global/Flat
Symmetry of children vertices in hierarchies	Local/Hierarchical
Uniform density of vertices in the drawing	Global/Flat

Verticality of hierarchic structures	Local/Hierarchical
--------------------------------------	--------------------

Table 2.1 A table of suggested aesthetic criteria which affect the readability of graph layouts as provided by Tamassia et al (1988), further information regarding the origins of these algorithms can be found in the publications

It is remarked that methods for drawing undirected general graphs, specifically force directed placement techniques, typically optimise: the uniform density of vertices; symmetry; minimisation of bends along edges; and minimisation of the area occupied by the graph.

Due to the subjective and contextual nature of analysing graph layouts, measurement and evaluation of each attribute's effect on readability is difficult, if not impossible. As such, empirical investigations are used to investigate their nature through subjects answering questions regarding graphs with differing layouts.

The procedures for many of these studies follow a similar pattern: subjects are given a series of questions regarding structure and connectivity within various layouts exhibiting differing levels of an aesthetic. The subjects' response times and correctness are measured, with correctness and fast response times believed to suggest increased readability. Similar conclusions are reached throughout many of the studies, with variations depending on the context (for example, when testing on UML diagrams specifically, the results show Orthogonality has more of an impact than in general graphs). A generalised cross analysis is provided in Table 2.1.

Aesthetic Criteria	Definition	Evaluations,	Effect on Readability
Minimisation of edge crossings	Reducing the number of edge crossings exhibited in a layout	Purchase et al (1995, 1996, 2002) Purchase (1997, 2000) Ware et al (2002)	High Impact
Minimisation of bends	Minimisation of the number of bends in edges (in layouts not using the straight line standard)	Purchase et al (1995, 1996) Purchase (1997, 2000)	High Impact
Continuity of edges	Reducing angular deviation in the edges of a path, drawing them as straight as possible	Ware et al (2002)	High Impact
Orthogonality	Maximising the orthogonality of a layout	Purchase et al (2002) Purchase (1997, 2000)	Medium Impact
Symmetry	Drawing symmetrical components of a graph as similarly as possible	Purchase et al (1995, 1996, 2002) Purchase (1997, 2000)	Low or No Significant Impact
Minimum angles	Increasing the minimum angles between coincident edges of a vertex	Ware et al (2002) Purchase (1997, 2000) Purchase et al (2002)	Low or No Significant Impact
Branches	The degree of vertices in a path which do not belong to the path	Ware et al (2002)	Low or No Significant Impact
Total geometric line length	The total line length of the graph	Ware et al (2002)	Low or No Significant Impact

Table 2.2 Identification of literature empirically evaluating aesthetics expected to affect the readability of two dimensional graphs

The results of the empirical studies show a preference towards minimisation of edge crossings for improving readability of graph layouts. Continuity is demonstrated to have a greater effect on reaction times (Ware et al, 2002), but is specific to optimisation of shortest paths (generating layouts specific to an action). Minimisation of bends is shown to have the second most noticeable effect on readability but only applies to orthogonal drawings.

Other aesthetics are shown to have very little effect. Bennet et al (2007) provides a more in-depth cross evaluation and analysis of core aesthetics, identifying heuristics which aim to achieve and test their effectiveness on:

- Visceral, behavioural and reflective levels
- Symmetry, orientation and contause
- Conflicting/balancing aesthetics

Purchase (2002) aids algorithm design with suggestions for metrics of aesthetics, and provides cost functions which can be incorporated into genetic and simulated annealing algorithms. The metrics for edge crossings are of high interest due to their described high impact on layouts, with metrics for Symmetry and Angles noted but lesser used due to their lower impact.

An alternate study uses eye movement to determine how a user reads a graph and reacts to aesthetic criteria (Huang and Eades, 2005). The study showed subjects tend to follow edges toward a target node and avoid areas densely populated with vertices and edges. Further to this and the use of questionnaires, Huang et al (2008) suggest metrics for measuring cognitive load of participants in order to aid evaluation of layouts, obtained through subjects reporting their “mental load” using a scale of 1 to 9, referring to low and high “mental effort” respectively.

It should be noted that the results of these empirical studies are for smaller graphs where identification and visualisation of these attributes are more noticeable. For large and huge graphs there is no known literature providing definitions or analysis of aesthetics, partly due to the difficulties in identifying criteria for high densities of edges and vertices in the viewing platform (Herman et al, 2000). However, attributes may still be applicable to local structures

within a graph (Pfaltz, 1972) and are expected to remain relevant to local layouts and approximated global layout quality.

2.4.4 Overview of Static Graph Drawing

Force directed placement offers various techniques for graph drawing by treating the graph as a mechanical system with some energy, the reduction of which is expected to relate to high quality vertex positions. Although providing high quality layouts, the methods suffer from high running times and local minima. To overcome these, various approaches have been applied to progress the field.

Multilevel and multiscale techniques provide a means of generating simplifications of a graph, allowing for Force Directed Placement to be applied on a global level, improving global layout. Local layout can be refined with further *FDP*. The methods are shown to work effectively to overcome minima and provide access to visualising huge graphs (>100,000 vertices). Approximation methods are used to overcome the high running times associated with *FDP*, reducing the number of calculations but retaining accuracy.

Further to the literature covered here, a number of surveys summarising the techniques and methods of graph drawing are used by the community. A broad collection of early graph drawing algorithms, and analysis of each, is given in Battista et al (1994). Kobourov (2004) provides a detailed review of force directed placement algorithms, detailing the most common and popular methods available, with a more recent review including multilevel schemes in Kobourov (2012). Hu (2011) also provides a recent review and analysis of general graph drawing algorithms for large graphs in the context of visualising large networks.

2.5 Dynamic Graph Drawing

Dynamic graphs are those which change over time, typically including the removal, addition or amendment of vertices and edges. These changes may be described as “offline” if known beforehand, or “online” if the changes are not known and unpredictable. The contributions here focus on “offline” datasets, however the described works are applicable to both.

The drawing methods for such graphs optimise the visualisation of the changes to the graph and evolution of the layout, typically through the use of frames or animation (Archambault et al, 2011).

Literature regarding dynamic graphs is often split by specific graph types, and uses heuristics relating to topology and other known/expected attributes of those graphs, including hierarchical graphs (North and Woodhull (2002), Alstrup et al (2005), Gorg et al (2005)), planar graphs requiring planarity testing for changes to the graph (Tamassia, 1989) and chordal graphs (Ibarra (2001), Hill et al (2005)). Cohen et al (1992) suggest a framework for drawing and maintaining dynamic graphs using a collection of such algorithms based on the authors' previous work.

Misue et al (1995) suggest a concept of the mental map and identify attributes which may help preserve it, applying them in a dynamic graph drawing framework. Force-Scan approaches use force based shuffling to separate and arrange coincident nodes, and a variety of visual mappings (fisheye, orthogonal fisheye and use of rectangular viewing areas (biform)) to transform sections of the graph such that changes are viewed and placed in such a way that the remaining layout is unchanged. The conclusions are that they enable the mental map to be preserved during layout adjustment.

Brandes and Wagner (1997) adapt static drawing algorithms to work in a dynamic setting, testing two popular layout algorithms: Eades' Spring Embedder (Eades, 1984) and Tamassia's minimum-bend orthogonal layout model for planar graphs (Tamassia, 1996). The authors suggest few changes are required to repurpose the algorithms, primarily discussing methods of stabilising the layouts and avoiding the large movements and changes that occur in static algorithms, preserving the readability and understanding of the graph.

The approach is of interest here due to the application of static methods to dynamic graphs, setting a benchmark for visualisation using shared methods.

Similarly, Diehl and Gorg (2002) introduce a method which generates a layout for a 'supergraph' using simulated annealing. The supergraph is generated from a series of graphs which show a story of a graph as changes are applied. The graph is given a 'base layout'

which is applied to all graphs in the series, and are refined using further simulated annealing methods. Like the methods above, the approach encourages preservation of a base layout.

A similar adaptation is described by North and Woodhull (2002), which suggests a dynamic graph drawing algorithm based on the static work of Sugiyama et al (1981), used for on-line hierarchical drawing (a modification of which is used in the Graphviz and Dynagraph packages from Ellson et al, 2004).

Following the use of Simulated Annealing, Lee et al (2006) present a method for generating dynamic graph layout which does not require prior knowledge of graph changes. Following from the work by Davidson and Harel (1996), the algorithm uses a collection of parameters to control the various aesthetics of the graph in order to maintain a user's mental map. Using a framework provided by Bridgeman and Tamassia (2002), the authors identify those parameters which will affect the readers' perception of the layout and set the appropriate cost functions in order to maintain it. The algorithm relies on redrawing the graph when changes have been applied, generating a sequence of frames.

Cohen (1997), and Baur and Schank (2008), describe an extension to the Spring Embedder method (Kamada and Kawai, 1989) and provide use of multidimensional scaling techniques (MDS) to reduce the "stress" of a system. Xu et al (2011) also make use of dynamic multidimensional scaling as a method of layout adjustment, but use the stress of a graph to identify the energy of a system as a means of layout analysis. The authors also suggest measuring the temporal difference in vertex positions (between frames) and the distances between nodes in groups as methods for layout analysis.

Although using different minimisation techniques, the methods follow the spring embedder principle and encourage applying spring embedder methods using in static graph drawing, to generating layout for the dynamic graphs.

Further extending the works in static graph drawing, Frishman and Tal (2008) suggest a method using multilevel force directed placement for quickly generating high quality layouts for frames of a dynamic graph (a frame referencing a change or sequence of changes to the layout). Sequential layouts are morphed to provide a smooth transition. Layout is preserved

through vertex pinning, whereby only vertices close to the changes to the graph are given adjusted layouts.

The use in this way identifies a leap to using multilevel methods for improving layout, and for quickly generating layout. Although generating high quality layouts, the use of applying multilevel FDP to generate layouts one at a time is not optimal, especially for fast changing graphs which demand for fluid layout adjustment.

Veldhuizen (2007) suggests a more elegant solution for visualising large dynamic graphs using a system of dynamic springs embedded into a multilevel scheme. The approach generates layout for all levels of a multilevel scheme simultaneously, ensuring positions of vertices in coarser graphs still represent the vertices in finer graphs by connecting them using springs (as a type of inter-graph connectivity). Damping and time dilation mechanisms are used to ensure graphs evolve at a similar rate.

The author also uses the Octree approximation to reduce the complexity of the algorithm and further reduce the running time.

Unlike other literature, Veldhuizen's algorithm is run continuously through animation (as opposed to frames) with changes to the graph being visualised immediately. The algorithm is also shown (via demonstration videos) to be very quick, showing the generation of layout for large graphs (1000 nodes) in just a few seconds.

A review of the current issues facing the visualisation of large networks is provided in Hu (2013), which introduces algorithms deemed state-of-the-art and details the biggest problem as size of the graphs involved and the computational time required to generate layouts.

Of the dynamic graph algorithms identified, most encompass the modification of static force directed placement methods to cope with the visualisation of dynamically changing data, with a high interest in graph stability for preserving a user's perception (such as vertex movement). Apart from schemes by Frishman and Tal (2008) and Veldhuizen (2007), all identified methods are applicable only to small graphs and are still limited by expensive calculations and local minima.

2.5.1 Metrics for Graph Stability

Although affected by the aesthetic criteria, dynamic graphs are interpreted differently to static graphs due to their changing layout and the added time component, resulting in a host of further attributes for determining the readability of a graph. As an example, typical use of a static graph may involve identifying the shortest path between two nodes, whereas in dynamic graphs, questions may relate more to the movement of vertices and the changes in connectivity.

Changes to a dynamic graph and corresponding layout will likely impact a viewer's perception of the visualisation. This perception is abstract will differ from person to person, however there are many attempts at defining it. A common theme across the research below is that a user will be looking at changes in the graph, and how those affect the impact on the existing layout. As such, this abstract way a user perceives a changing layout is regarded as a stability problem.

Böhringer and Paulisch (1990) refer to the problem of drawing dynamic graphs nicely as the stability problem, suggesting a system of constraints to control attributes relating to the shape of a graph and positioning of vertices, modified to achieve stability. Lai et al (1991) build upon this and suggest models for measuring changes to a user's perception of changing layout and define criteria to describe the shape of the layout:

- orthogonal ordering – positions of vertices relative to one another, for example, N S E W;
- preservation of locality and the shapes of vertex clusters;
- topology of a graph and transformation of shapes.

Later work by the authors provides a framework which apply these models (Misue et al, 1995), including a selection of layout adjustment methods (based on transformations and shuffling) and visualisation techniques to optimise the aesthetics.

The term “mental map” as used by Misue et al (1995) refers to a viewer's changing interpretation of a dynamic graph as it is amended. Similarly to static graph aesthetics, definition of the “mental map” is subjective, with authors providing their own variations on which attributes should be optimised to best preserve it.

Friedrich and Eades (2002) build on research by Papakosta and Tollis (1996) and suggest criteria relating to the motion of a graph as a means of improving readability, with interest in the perception of animation or movement of layouts as rigid three dimensional objects. The approach looks at similar shape properties within the graph as described by Lai et al (1991):

- minimise temporary edge crossings;
- maintain minimal distance between nodes which do not move uniformly;
- maximise structured movements (movement of vertex clusters as one).

A survey of dynamic graph drawing by Shannon and Quigley (2007) identifies the main problems faced with dynamic graph drawing algorithms as:

- how to model the graph (data structures);
- the sensitivity of the graph to change;
- and identification of which aesthetics are preferred by the viewer.

Discussion on the uses of dynamic graphs is provided by Alberts et al (1998), whom also suggest aesthetic criteria for preserving the a users perception while performing tasks such as checking connectivity, identifying minimum spanning trees or identifying single source shortest paths.

In terms of criteria which would best optimise the properties these tasks assess, studies (Table 2.2) show that for questions regarding the connectivity and shortest paths, minimisation of edge crossings and continuity of edges have greatest effect.

Papakostas and Tollis (1996) propose a method of preserving layout through limitation of vertex movement, suggesting readability is affected by the movement of vertices between frames, requiring the reader to keep track of multiple vertices moving at once. Purchase et al (2007, 2008) evaluates such limitation of vertex movement as a means of altering the readability of a dynamic layout and of preserving graph stability, showing that limiting vertex movement aids the readability of some graphs for some questions, supporting the suggestion that the a users perception of layout is dependent on the context of the data and is more down to users' suggestions as to which aesthetic to control.

Further empirical studies investigating the effect of a user's perception of a layout on memorability in dynamic graphs suggest that layout preservation has little effect on the readability of graphs on its own (Archambault et al (2011), Archambault et al (2011), Archambault et al (2012)).

Comparison of user generated layouts and automatic layouts is described by Dwyer et al (2009), showing that users pay more attention to the layout of cliques over other criteria. In contrast to the empirical studies of Purchase (1997, 2000) and Purchase et al (1995, 1996, 2002), the results show that 77 of 188 subjects paid most attention to cliques as opposed to 17 of 188 for edge crossings.

An addition to the concept of the graph stability is the "faithfulness" of a layout, in other words: does a visualisation perform the task required of it? Nguyen et al (2012) suggest the idea to "fill the missing link" of the stability problem, stating that aesthetic criteria are required but not sufficient for measuring the layout of a graph, and that the "faithfulness" adds another metric for evaluating algorithms which is better linked to the context of the drawing.

The literature identifies the use of graph stability as a means of describing the usefulness of layout for dynamic graphs, and although much research exists, there is an inherent difficulty in defining the metrics of graph stability and the attributes which contribute it, and no known literature regarding huge datasets.

2.6 Summary

Although the scope of the research here is primary algorithmic design, an introduction to the "aesthetics" of graph layout is provided, suggesting a means of identifying high quality layout through the attributes of the layout as a result of automatic layout algorithms.

- The aesthetic which is suggested to have greatest effect on the readability of a static layout generated by force directed placement is the minimisation of edge crossings, suggesting the use of edge crossings as a metric for identifying high quality layouts
- Identification of the aesthetics which have greatest effect on preservation of the users perception of a layout is inconclusive, with results suggesting the specific use and

context of the visualisation holds most importance, however cost functions can be used for optimising various aesthetic qualities (e.g. optimising shortest paths)

Graph drawing algorithms are reviewed in two stages, static and dynamic. The research into static graph drawing focuses on the development of early force directed placement schemes into the multilevel force directed placement algorithm of Hu (2006), providing background for the three components of typical algorithms:

- Spring Embedder (and general Force Directed Placement) – the metaphor used to generate layout
- Multilevel schemes – used to overcome limitations of Force Directed Placement and extend its functionality to larger datasets
- Approximation – used to reduce the complexity of n-body forces

Dynamic graph drawing identifies the techniques of layout adjustment used to visualise changes to a graph, and authors' methods of maintaining the user's perception of the graph such that the layout is meaningful. Many frameworks are suggested but follow similar concerns:

- Layout changes should not result in the loss of a user's understanding of the graph;
- Changes should be visualised in real time;
- Visualisation of changing layout is often achieved through the use of snapshots over time (Archambault et al, 2011)

Existing dynamic graph drawing literature is largely focused on the development of algorithms for smaller graphs and optimisation of graph stability. Few techniques exist for larger graphs due to difficulties providing a global and local layout (overcoming minima), readability of large collections of information (visualisation of data), and producing fluid reaction to changes to a graph quickly (maintain metrics for graph stability and running time). Frishman et al (2007) and Veldhuizen (2007) provide methods using a multilevel scheme to improve upon layout generation, however, the approaches are largely hardware orientated and feature little investigation of updating the multilevel scheme when a graph changes. Such issues are the aim of investigation here.

3 Concepts

3.1 Overview

This chapter details the research for new algorithmic approaches, categorised into sections detailing the contributions and methods used, cumulating in a framework for drawing large dynamic graphs efficiently.

Firstly, a **Problem Description** (Section 3.2) is provided, detailing the scope of research and the issues to be resolved through completion of the aims and objectives. Following this, the motivation behind the algorithm choices is described in **Story of Algorithms** (Section 3.3).

The **Implementation Methodology** (Section 3.4) then describes the Experimental Framework used to implement, develop and test the contributions of the thesis, followed by methods for **Measuring Layouts** (Section 3.5), describing the metrics used by the Experimental Framework for analysing and comparing static and dynamic graph layouts during experimentation.

Following these, a method for improving the efficiency of force directed placement is described, named **Multilevel Global Force** (Section 3.6) approximation (Crawford. 2013). The method exploits a multilevel scheme (Walshaw, 2003) to reduce the cost of approximating global forces in force directed placement (Fruchterman and Reingold, 1991). The technique reduces the number of calculations and provides approximation which better represents the structure of a graph. Additional methods for improving efficiency and better encapsulating the structural information of the graph are discussed.

Methods for applying optimisation of static layouts to dynamic graphs are described in **Dynamic Graph Drawing** (Section 3.7), detailing the methods of visualisation and the operations available to amend dynamic graphs. A modified dynamic spring embedder, based upon a static method described by Fruchterman and Reingold (1991), is introduced, providing improved layout convergence and better overcome minima. Adaptation of a multilevel scheme is also included with methods for incorporating amendments to the dynamic graph into the multilevel matching algorithm in order to incorporate local layout

changes into global layouts through Multilevel layout and MGF, described here as Dynamic Matching.

Finally, a **Summary** (Section 3.8) is provided to conclude the chapter in regard to the problem description.

3.2 Problem Description

Literature and research on static graph drawing for large and huge graphs is extensive and well documented with many techniques available for improving efficiency and readability of the layouts generated. However, research into dynamic graph drawing is largely focused on visualisation of smaller graphs (Tamassia et al, 1988) due to the complexities of existing algorithms and difficulty reading large quantities of changing data (Shannon and Quigley, 2007). Despite the rise in data available for visualisation in recent years (Hu, 2013), few algorithms have been developed for visualisation of large dynamic graphs. Additionally, analysis of layout quality has been largely focused on smaller graphs (see Section 2.4.3 and 2.5.1), with no identification of which aesthetic criteria affect readability of graphs with thousands of vertices.

The literature review indicates that methods for static and dynamic drawing are similar if not the same (Eades (1984), Davidson and Harel (1996)), suggesting the approaches used to optimise static graph drawing can be transferred to dynamic drawing (for example, Frishman et al, 2007).

Prior to the extension of such developments, the weaknesses and difficulties associated with drawing large dynamic graphs are identified, specifically those not covered by existing techniques. The problems as identified from the Literature Review are:

- Difficulty reading, understanding and following the evolution of large amounts of data which change over time;
- High complexity and costs associated with drawing techniques (making their usage unfeasible);
- Providing global and local layout simultaneously during the drawing process and overcoming minima;

- Little analysis of aesthetics of large and huge graphs which affect readability or the Mental Map.

Utilising these as a specification, the intended research aims are generated;

- To provide efficient force calculation of global interactions (**Multilevel Global Force**)
- Visualization and adjustment of high quality layouts for large graphs such that changes can be observed immediately (**Dynamic Graph Drawing**)
- Global layout and local layout generation simultaneously via multilevel integration (**Dynamic Graph Drawing**)
- Definition of aesthetics which affect readability of large and huge graphs (**Measuring Layouts**)

In summary, the aim of the work is to identify and develop methods to aid the drawing and visualization of large and huge dynamic graphs, using multilevel schemes to provide both global and local layout, and efficient approximation to reduce cost and improve efficiency of generating visualisations.

3.3 Story of Algorithms

One of the aims of this thesis is to find ways in which dynamic graph drawing algorithms can be optimised for larger graphs. As such, it is preferable to utilise an algorithm that has the ability to achieve such results.

The Spring Embedder is the chosen route for this investigation as it has already proven itself capable of being optimised for huge static graphs through means of Multilevel and approximation schemes (Walshaw (2003) and Hu (2005)). In addition, the algorithm has been introduced to dynamic graph drawing previously (Brandes and Wagner (1997)), making it a good candidate for experimentation.

A particular quality of the Spring Embedder is its iterative approach to generating layout, such that the algorithm (as described by Eades (1984)) can be run continuously, which when visualised appears as an animation showing progressive changes in vertex positions. Although other dynamic drawing algorithms exist, there is little research in enabling them

to cope with larger graphs with more than 8000 vertices, unlike the Spring Embedder which has previously been optimised for drawing larger static graphs (with up to millions of vertices, Walshaw).

Through implementation of the Multilevel scheme and Octree approximation (to better understand the algorithms), similarities between the two became apparent and Multilevel Global Forces (MGF) was born, offering further efficiency savings for the Spring Embedder and making the algorithm choices much more desirable. An implementation of the Barnes Hut Octree, and the Multilevel Spring Embedder were introduced into the Experimental Framework for comparison to MGF.

However, the Spring Embedder alone (as described by Eades (1984)) has difficulty with overcoming minima to untangle larger layouts, for which the solution in static graph drawing (introducing a cooling schedule to allow high energy movements, Fruchterman and Reingold (1991)) is not directly transferrable to dynamic graph drawing.

The Modified Dynamic Spring Embedder acts as the solution for this; developed through implementation and experimentation (following discussion with Danko, Appendix 10.3), the algorithm provides repulsive and spring forces to overcome minima. As the algorithm is an adaptation of the Modified Spring Embedder, the multilevel scheme and Multilevel Global Forces became available with few changes necessary.

To further reduce complexity, other methods for optimising the multilevel structure and vertex placement schemes are investigated.

In order to attain dynamic drawing however, the algorithms should support graph operations. This includes applying those changes to any multilevel and approximation schemes, with minimal impact to layout and running time. As such, four methods are suggested for updating the schemes with the graph changes to different extents; low, medium and high level updates, and total re-matching.

The extent of the update determines if visualising graph changes in layout is priority (high-level updates and re-matching) or if layout preservation is more a priority (low-level update). Mid-level updates are included to add a balance between both.

3.4 Implementation Methodology

In order to test and compare algorithms, and provide a fair and unbiased analysis of the contributions in respect to state-of-the-art methods, an Experimentation Framework for Graph Drawing is implemented. The framework acts enables implementation of different heuristics described in literature and combine them into different algorithmic configurations for comparison.

The aim is to remove environmental (processing power, memory, operating system) and implementation (data structures, executing environment) discrepancies which may contaminate results.

Such experimental frameworks are not new and can be found in other research in the area, and are used to provide context for the contributions (e.g. Misue et al (1995), Hu (2005), and Veldhuizen (2007)).

3.4.1 Configurations

An object orientated approach is used to break algorithms into single-purpose components which model a specific part of the drawing process such that they can be used independently. These can then rebuilt to include new or modified components for experimentation.

The shared resources (components, data structures and environment) should make comparison of the new algorithmic approaches fairer, with significant changes in running time and layout quality occurring as a result of algorithm changes.

In addition to the components, the Experimental Framework allows for applying various parameters and values to variables in the algorithms (for example, modifying cooling schedules). Experimentation can then identify which values can provide the best configurations for the algorithms in particular contexts.

Suggested configurations for the contributions are described in the Evaluations (Section 7).

3.4.2 Limitations

Although there is reuse of components, it's difficult to ensure they are not bias (for example, the Octree uses functions which are not used by other components and could slow down the component). Although algorithmic design is similar throughout, there is no perfect comparison.

The use of shared components limits the optimization that can be applied. Specialized algorithms which are single purpose can be optimized to a very fine detail, unlike an algorithm comprised of shared parts (optimization of which must apply to all algorithms using them).

Even if algorithms are implemented to the most efficient extent possible, comparison of algorithms are constricted by analysis; results are provided only for a selection of test data expected to provide a general behaviour, which is measured using 'aesthetics' which are not a perfect representation of a graph layout. As such, the recorded performance is unlikely to be 100% perfect. That being said, test data is taken from other graph-related areas (Partitioning, see 5.1), and includes data from real life sources.

3.4.3 Benefits

Like any undertaking, there are limitations, however there are a large number of benefits to be had from using an Experimentation Framework. The platform allows for: greater control over variables; high flexibility in algorithm and experimental design; automation of common tasks (particularly useful for repeating tests and collecting data).

Some of the more practical uses is the ability to access methods and functions, and ability to compare them with minor changes to the remaining algorithm, allowing for investigation of ideas and curiosities quickly.

Moreover, implementing the works of others gives better insight into the workings of the algorithms they describe, and may highlight some behaviours which are not openly spoken about in the publications (for example, the impact of bounding boxes in approximation,

Figure 3.1 for example). Being able to follow the works of others builds confidence in the contributions here and acts as a guiding hand in the development of new algorithms.

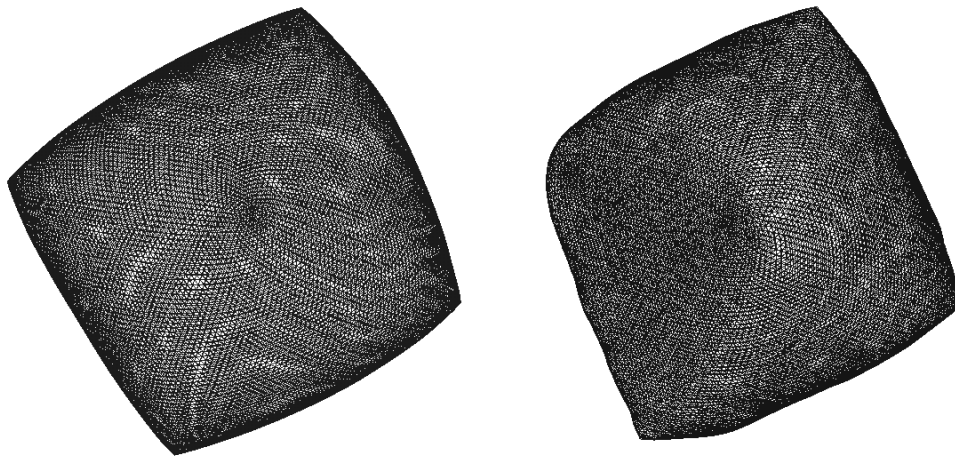


Figure 3.1. Layout generated for sphere using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right), exhibiting flattening of the graph as it encounters a boundary of the approximation structure

3.5 Measuring Layouts

Much of the literature in graph drawing provide definitions of aesthetics which are targeted in existing drawing algorithms (Tamassia et al, 1988), here the optimisation of aesthetic criteria is assumed through reuse of existing drawing methods. Optimisation of aesthetic criteria is not investigated here, instead using the metrics for comparing algorithm performance and parameter usage.

Although associated, static and dynamic layout analysis is different, analysis of static graphs focus on structural and geometric properties (such as shortest paths and connectivity), whereas dynamic layouts are reviewed in regard to the movement and change between consecutive frames. Literature however, only investigates these for smaller graphs. As such, a method for scaling the attributes to larger graphs through use of a multilevel scheme is described below, after which aesthetics of static and dynamic layouts assumed to affect readability are described in their respective sub-sections.

It should be noted, and has been noted previously by various authors (Table 2.2), that although the use of “aesthetics” here infers quality in layout, the effectiveness of a layout for representing data cannot be completely objective due to each individual’s interpretation

of the visualisation. Determining the accuracy of layout in regard to an individual's interpretation is considered outside the scope of the research.

3.5.1 Local and Global Aesthetics

Investigation of aesthetics which affect the readability of smaller graphs is well documented (see Table 2.2, Literature Review), however there is little investigation into readability of layouts for large and huge graphs. As a result, methods providing static layout for larger graphs often analyse layout quality with regard to the aesthetics of smaller graphs (Walshaw, 2003) or provide only a subjective comparison of layouts (Hu, 2005).

Although such aesthetics are effective at comparing layouts (Walshaw, 2003), larger graphs differ due to the vastness of the data making it difficult to observe and understand all the data at once, and so may not represent the readability as indicated by empirical testing (see Literature Review, Chapter 2). For example, tests such as identifying shortest paths (i.e. Purchase 2000) between two vertices support the use of edge crossings as a factor in readability, however identifying a shortest path between vertices in a large graph is much more difficult due to the higher volume of edges.

Dwyer et al (2009) showed that layouts of a graph drawn by humans have high focus on cliques more than edge crossings, suggesting that larger collections of data may be easier to analyse and read when broken into more manageable chunks for analysis. Further to this, for large collections of data it is expected a user will be more interested in the data as a whole (global) layout, only focusing on smaller areas of the layout for specific queries (local layout). An example of the author's interpretation of global and local layout of an example graph is provided in Figure 3.2 whereby vertices are grouped to form a global layout.

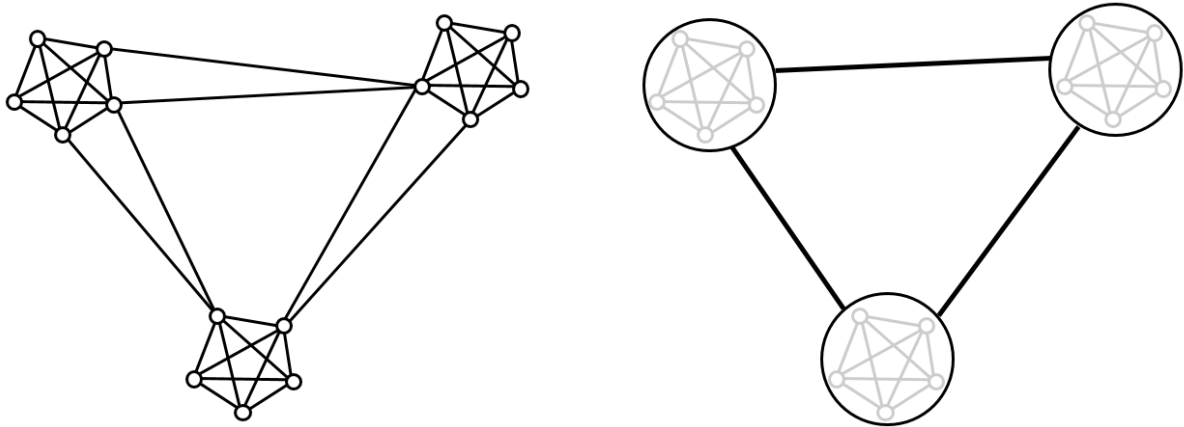


Figure 3.2. A small general graph comprising of three densely connected clusters loosely connected to one another (left), showing an interpreted view of the “global layout” whereby the densely connected clusters are viewed as individual vertices with edges and reducing the complexity of the diagram.

In graph drawing, multilevel schemes are used to scale force directed placement techniques to global layout through approximating the structure of the graph. Similarly here, a multilevel scheme can be used to identify an approximation of the global layout of a graph, providing a means of observing the global layout and applying the aesthetics/metrics used to analyse smaller graphs. This assumes global layout is approximated in coarser graphs and local layout is approximated in the finer graphs, such that aesthetic criteria can be measured for analysing different levels of readability in layouts. Figure 3.3 shows an approximation of a graph, showing a similar “global layout” to the author’s interpretation in Figure 3.2.

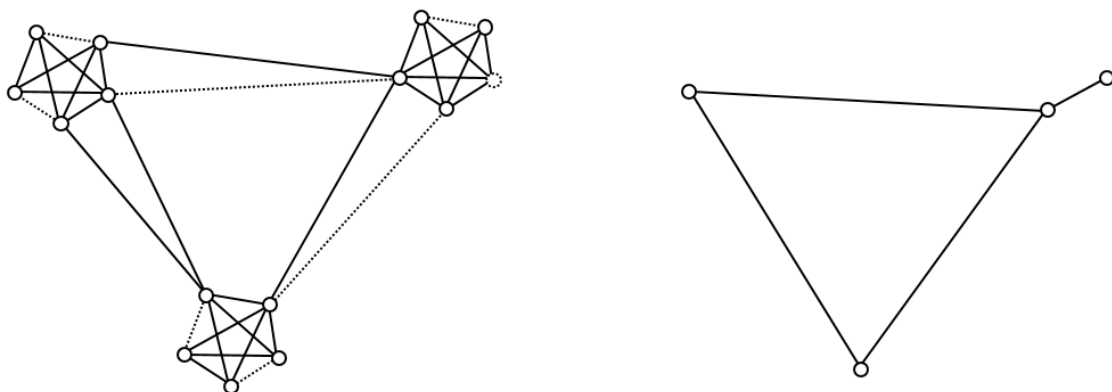


Figure 3.3. The same graph of Figure 3.2 but coarsened using a multilevel scheme to form an approximation of the structure, simplifying it to provide a simplified layout similar to the authors “global layout”. Dotted lines identify those edges which are collapsed to form the vertices in the coarser graph (right).

The examples of Figure 3.2 and Figure 3.3 are of a small graph whereby the local layout and global layout are observable from within the same visualisation. However, for larger graphs it is expected that local layout will require the user to explore specific parts of the layout and effectively split local and global layouts, suggesting analysis should follow such pattern of global and local analysis.

3.5.2 Aesthetics of Static Graphs

Research into graph aesthetics and readability provides the basis for optimisation of specific attributes of layouts in order to maximize the amount of information which can be read from them. Reviews of methods (Tamassia et al, 1988) show many algorithms optimise different criteria, with empirical studies identifying which criteria have greatest impact on readability (see Literature Review, Table 2.2).

Although many algorithms use subjective analysis (Eades (1984), Fruchterman and Reingold (1991), Davidson and Harel (1996), Hu (2005), Koren et al (2002)), the use of aesthetics for analysis and comparison of layout quality is chosen here, specifically: Edge Crossings (used by Walshaw, 2003) and Edge Lengths, chosen as they have previously been used in the area, they are suggested to have high impact on readability of layouts, and because they can be measured numerically (enabling more of a comparison of layouts than subjective views).

The nature of such attributes indicate similarity between layouts of isomorphic graphs, used to identify difference in layout quality between algorithms and parameters. Fewer edge crossings are expected to indicate more readable layouts (see Literature Review, Table 2.2) and ranges in edge lengths indicate the extent of the Peripheral Effect (Hu, 2005).

It should be noted that the ability to indicate readability in layouts is only assumed due to suggestions in literature (see Literature Review, Table 2.2), comparison also requires a subjective analysis to confirm the findings. As an example, given two 55x55 grids, one drawn with few edge crossings and the other with a fold in the body of the layout resulting in significantly more edge crossings (Figure 3.4), the difference is substantial and automatic analysis may reject the folded grid as a suitable layout, however, such a fold in layout may not alter the understanding of the graph as being a grid.

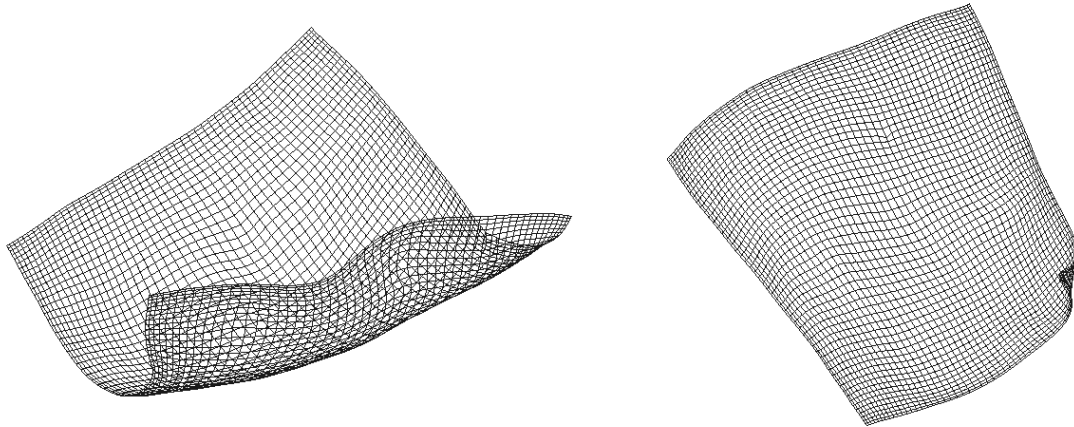


Figure 3.4. Two layouts of a 55x55 vertex grid, the layout on the left shows a significant fold with high numbers of edge crossings, whereas the right exhibits a lesser fold with far fewer edge crossings.

3.5.3 Metrics for Graph Stability

Dynamic layouts are typically visualised as frames or time-slices exhibiting layout changes via vertex movement. Frames are generated over time as layout changes, with differences between layouts in consecutive frames expected to relate to a user's interpretation of a layout. This interpretation is referred to as the metrics for graph stability (see Literature Review, Mental Map), with significant rapid change expected to result in the loss of a user's understanding. There are various definitions attempting to define user perception of layout across literature (see Literature Review), however for the purpose of analysis in this research the interpretation here is defined as the *difference in vertex movement* and *difference in edge crossings* between frames (as suggested by Purchase, 2007, 2008).

By recording the changes in vertex movement and edge crossings, it is believed analysis will be able to identify changes in the stability of the graph. Figure 3.5 shows an example which models this behaviour, describing a large decrease in edge crossings over the first 10 frames which is expected to be difficult for someone to follow. Larger vertex movements and changes in edge crossings between frames are expected to make the evolution of a layout harder to follow, with smaller movements making it easier to follow.

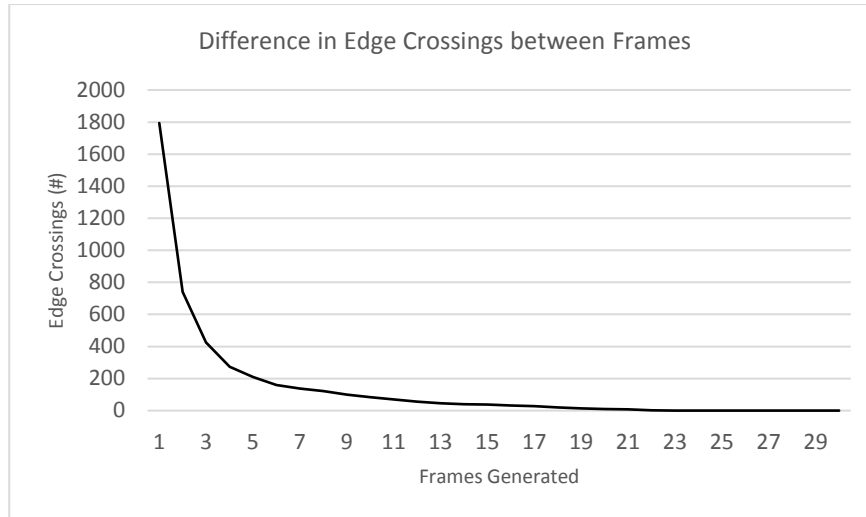


Figure 3.5. Given an example graph with initial random layout, it is expected the changes in edge crossings will gradually be reduced over time as Force Directed Placement pull vertices into more optimal positions, and if so, such reduction can be visualised and compared as a chart to determine if the efficiency of the drawing algorithm.

3.5.3.1 Layout Generation and Layout Adjustment

Measuring metrics for graph stability is expected to indicate the change in a layout overtime, the extent of which will likely change due to the number and frequency of operations to a graph (and an algorithms ability to respond and represent these changes in layout). Changes as a result of these operations is described as **Layout Adjustment**.

In addition to layout generation, it is of interest to the author to push drawing algorithms to an extreme to investigate what would happen to a layout after an extreme number and frequency of operations. Such examples can be generated through simple heuristics, however the data generated may not be a true representation of real world examples. As such, is used as an extreme of Layout Adjustment. The aim is to generate a layout for large static graphs (from an initial random layout) using a dynamic graph drawing algorithm, and measure the metrics for graph stability over time, identifying how well the algorithm can overcome minima and how quickly an optimal layout can be generated.

3.5.4 Summary of Aesthetics and Layout Quality

Graph aesthetics known to impact readability are used to measure the quality of layouts such that analysis and comparison between algorithms and parameters can be achieved. The aim

is to provide a means of quantifying the differences between algorithms with regard to how much they increase or decrease the readability of layouts.

Given the lack of literature governing aesthetics for large graphs, it is expected that aesthetics known to impact small graphs can be combined with use of a multilevel scheme to provide analysis of approximate global layouts of graphs (similar to its usage with force directed placement).

The metrics used here for measuring the quality of static layouts and individual frames of dynamic layouts are:

- Number of edge crossings;
- Uniformity of edge lengths.

Metrics for measuring graph stability are:

- Movement of vertices between frames;
- Difference in edge crossings between frames.

Two methods are suggested for observing performance on changing layouts:

- Layout Adjustment for layouts with operations being applied to a graph with initial layout provided
- Layout Generation whereby a graph is given an initial randomised layout, and is then improved until an optimal layout is found

3.6 Multilevel Global Force (*MGF*)

Both static and dynamic graph drawing share a history in force directed placement with early algorithms being used for both (Eades (1984), Davidson and Harel (1996)). Accordingly, and as part of the contribution presented in this thesis, developments in static graph drawing were analysed and a new method introduced to provide an efficiency saving for force directed placement techniques which may also be applied to dynamic graph drawing. To provide comparison to existing approaches, the method is described for use with static graph drawing, with further adaptation for dynamic drawing (Section 3.7.6).

As indicated by literature, one of the dominant problems facing force directed placement techniques is overcoming minima, that is, layouts which require temporary increase in energy to achieve lower energies (early algorithms do not allow such increases in graph energy, e.g. Eades (1984), Kamada and Kawai (1989)). Cooling schedules combined with powerful forces (Fruchterman and Reingold, 1994) are shown to be effective at overcoming this, but larger graphs tend to exhibit more minima in which to get stuck, and as a result global layout is often poor (high number of edge crossings).

Multilevel schemes have been shown to overcome this, providing global layout through application of Force Directed Placement to approximations of a graph containing fewer vertices (Walshaw (2003), Harel and Koren (2001)). Such schemes produce subsequently smaller and more approximate graphs on which layout algorithms can be applied, providing layouts which are interpolated and then refined using further force directed placement.

3.6.1 Calculating Approximation

Typically multilevel schemes are used to generate layout through the following steps (more detailed pseudocode can be found in Appendix 10.22.3):

- Calculate a collection of successively coarser graphs
- calculating a layout for the coarsest graph using Force Directed Placement (*FDP*)
- interpolating the layout into the next finer graph of the sequence
- Further refinement using FDP.

Figure 2.1 and Figure 2.3 shows that the both multilevel schemes and Octree approximation techniques can be mapped as trees to identify the splits and matches within, however their usage is different with one method approximating space and the other approximating graph structure.

By altering the multilevel scheme such that coarser vertices take the average positions of their finer counterparts after refinement, the multilevel scheme can be adapted to provide an approximation of both structure and layout of a graph.

Figure 3.6 illustrates the procedure for this on an example graph:

- The graph has already been processed and exists within a multilevel scheme, whereby the matched independent edges are denoted as dashed lines in each of the coarser abstractions;
- Starting with the original graph G_0 , for each of the independent edges which have been matched, the positions of the two connected vertices are averaged to give a centre of mass (similar to the Octree approximation);
- The centre of mass is then given to the parent vertex which represents the two vertices in the coarser graph (G_1);
- This process is then repeated on each of the subsequent coarser graphs, building a tree of graph layouts which approximate the layout of the original graph;

From this, the vertices in the coarser graphs have been given positions which approximate the positions of the vertices they represent in the original graph, effectively approximating the layout. Note that the shape of the coarse graph in **Figure 3.6**, which would normally have equal edge lengths, is elongated to represent the positions of vertices in the original graph.

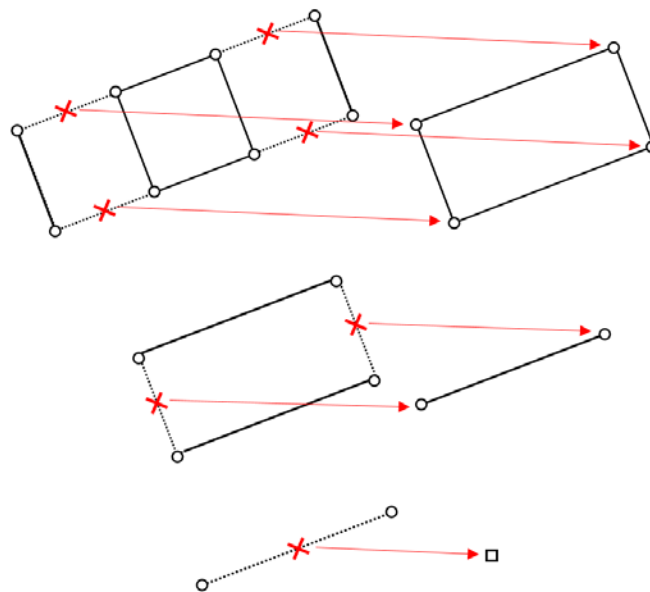


Figure 3.6. Combining the approximation of structure as generated by the multilevel scheme with an approximation of layout, with an initial graph of 8 vertices coarsened to form a graph of 4 vertices, approximating the positions (crosses) of the coarsened vertices (top). The process is repeated for each of the coarsened graphs (middle, bottom).

In order to use the scheme as approximation of vertex positions, the process is reversed such that vertices in coarser graphs take the average position of the vertices they represent (children), using a post order traversal of the *MGF* tree.

By updating the positions of vertices in coarser graphs, the layout generated by Force Directed Placement is overwritten. Those former positions cannot be used for approximation due to differences between layout generated through force directed placement and approximate positions, as shown in Figure 3.7. As a consequence of this, implementation and usage of Multilevel Global Force use an additional set of coordinates for coarser vertices due to the need to retain the layout generated for coarser graphs (a description is provided in 3.7.6).

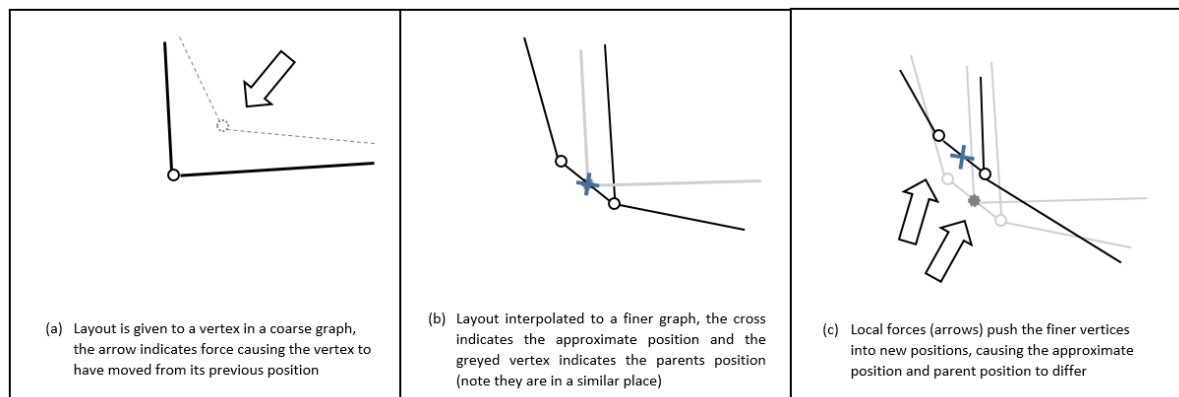


Figure 3.7. An example of changes in positions between graphs, showing the difference between outdated positions generated by Force Directed Placement and an approximation of the position of vertices

Once all coarser graphs have been updated with approximate positions, the tree becomes ready for use in force calculation. Usage is the same as the Octree (providing the ability to switch between them without alteration to force directed placement), whereby given a vertex v , repulsive forces are calculated between it and its siblings, its parents siblings and so forth until the root of the tree is reached, as shown in Figure 3.8.

Figure 3.8 shows traversal of the multilevel mapping, with repulsive forces approximated against vertices in the coarser graph: a , b , and c , acting upon a vertex, v . As vertices v and a share the same parent vertex, repulsive forces are calculated between them, after which no more sibling exist and traversal of the multilevel mapping tree begins. The parent of v has one sibling, b , representing and approximating the position of two vertices in the original

graph, and used to approximate the repulsive forces of those against v . Further traversal reaches another vertex which has sibling c , representing and approximating the position four vertices in the original graph for repulsive calculation. Further traversal is not possible as the root of the mapping has no parent or sibling vertices.

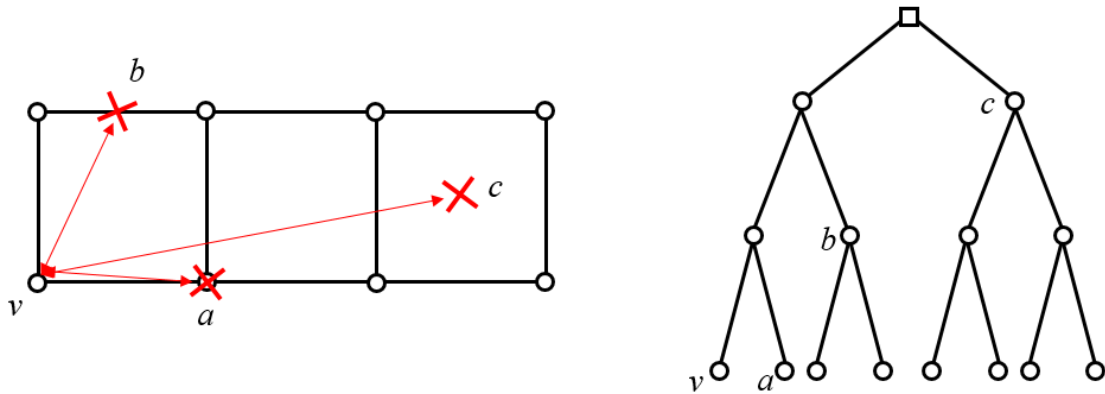


Figure 3.8. An example of Multilevel Global Force approximation in approximating global repulsive forces (left) and the usage mapped to the *MGF* tree (left). Repulsive forces are calculated between a vertex v , and coarse vertices (approximate positions) a , b and c . The result is reduced number of calculations.

The effect on running time is dependent on the structure of the mapped tree, with Multilevel Global Force providing fewer calculations when using a multilevel scheme (Walshaw, 2003) than the Octree (Barnes and Hut, 1964). The example above, Figure 3.8, reduces the number of calculations acting on a vertex from 7 to 3, whereas the Octree (Figure 2.3) reduces the number to 4, totalling in 24 and 32 repulsive force calculations for the graph (per iteration). The structure of the approximation is the basis for the efficiency saving as detailed below.

3.6.2 Structure of Approximation

The term “structure” here refers to the tree-like mapping of matches for approximation (as shown in Figure 2.1 and Figure 2.3). The connectivity of this tree determines the number of calculations made per vertex when approximating inter-vertex interactions.

The difference in structure and dimensions of the tree alters the directions in which repulsive forces are calculated. The Octree provides an equilateral approximation of repulsive forces in four or eight directions around the centre of approximation, resulting in a layout which expands outward in these directions, whereas an *MGF* tree constructed via edge contraction ultimately provides two directions of approximation (between the two coarsest vertices)

causing stretching on the axis between them. Stretching or deformation as a result of this is referred to here as “warping” of the layout (see Figure 3.9).

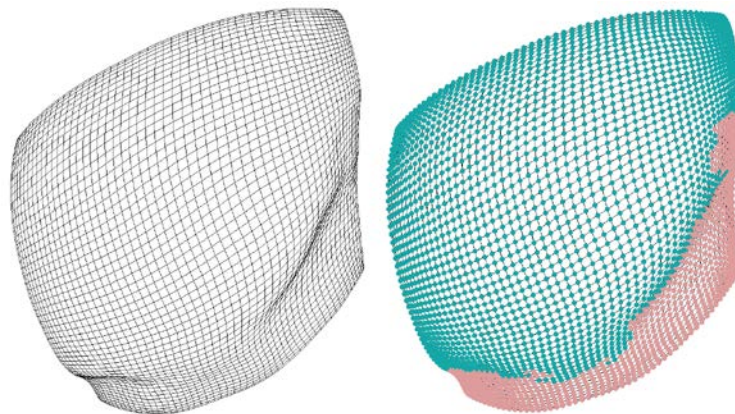


Figure 3.9. Approximation with *MGF* showing weakened repulsive forces in approximation, resulting in two parts of the graph being squashed together to form a crease in the layout. Coloured regions indicate the vertices belonging to the two coarsest vertices in the multilevel scheme.

It should be noted, warping in the Octree is minimised due to the approximation grid being placed with the centre atop the centre-of-mass for the graph, however as vertices move between splits, warping can become more prevalent due to differences in weights. *MGF* circumvents this by not relying on an external structure atop the layout.

Other multiscale schemes (Hachel and Jünger (2005), Gajer et al (2001)) could be used to generate similar approximation trees with differing dimensions. To investigate the impact of structure on layouts and running time, the matching algorithm used here is modified in various ways.

In order to build the most optimal solution, modifications for generating the multilevel scheme are investigated in a hope to reduce the complexity (and running time) of graph drawing algorithms (enabling larger graphs to be drawn, particularly of use for Dynamic Graph Drawing).

Multimatching is a method which contracts multiple edges connected to a common vertex, providing more directions of approximation. The method will generate coarser representations of a graph but is expected to reduce warping over edge contraction, and

generate layouts similar to those by the Octree. A more in-depth description is provided in Appendix 10.20 .

Brute Matching aims to collapse entire neighbourhoods of vertices, coarsening all edges connected to a common vertex. The process is implemented as an extreme Multimatching and is expected to provide an irregular matching tree as opposed to one which aims to achieve a constant matching value. See Appendix 10.8 for more information.

Pattern Processing is a more specific matching technique for identifying and processing common vertex patterns/structures within a graph which can be given a known or expected layout which can be calculated after *FDP*. This interest here is the use of identifying such patterns (for example, leaf vertices) and coarsening them such that they are approximated as a single vertex. The removal of the patterns reduces the complexity of a graph, and any resulting matching/approximation tree.

3.6.3 Summary of Multilevel Global Force

Multilevel Global Force (*MGF*) approximation refers to the usage of the multilevel scheme for approximation of global forces in force directed placement based on the inherent relationships within graphs (in contrast to space decomposition techniques). The method can be exploited from a multilevel scheme, combining approximation of space and structure requiring no additional implementation for its use.

The method can replace existing methods and is near identical to the Octree method currently used in literature (Hu (2006), Hachul and Jünger (2005), Veldhuizen (2007)), and makes approximation more flexible, removing the requirement for detached grids being placed atop the layout. The expectation is a method of approximation which better represents the structure of the graph and does not require the maintenance of space decomposition techniques.

3.7 Dynamic Graph Drawing

A dynamic graph is a collection of vertices and edges which change over time, with the changes not necessarily known beforehand. Dynamic graph drawing is the generation or

adjustment of a layout to reflect the changes to a graph as they occur, typically to optimise the readability of the drawing through the influence of graph attributes.

Many of the force directed placement methods identified in the literature survey are shared with static graph drawing, however the focus is on gradual development of a layout in reaction to changes more than finding an optimal layout as quickly as possible. Few techniques exist which allow for the visualization of large graphs (>11000 vertices as described by Walshaw, 2003), with those available utilising static methods optimised for visualisation and preservation of the graph stability (Frishman and Tal (2008), Papakostas and Tollis (1996)). The methods introduced here look to continue this by suggesting drawing methods derived from static drawing techniques, altered to optimise for visualisation of vertex movements.

The Spring Embedder is of interest in this research due to its usage in static graph drawing and the existing techniques used to overcome local minima and high running times (see Literature Review). The subsections below detail the definitions used here regarding visualisation of dynamic graphs, including descriptions of the amendments supported. In particular there is a new method of dynamic matching for the inclusion of changes to a graph into the multilevel structure. In addition, an adaptation of a spring embedder typically used in static graph drawing (Fruchterman and Reingold, 1991) and adaptations of a multilevel scheme and *MGF* approximation for use in dynamic graph drawing are provided.

3.7.1 Visualisation

Graph layout is visualised as an animation in real time such that modifications to a graph as they are made. The animation is composed of frames, with each frame generated per iteration of force directed placement, resulting in a collection of static graph layouts. Force directed placement is applied continuously, with changes made after each iteration such that they will be displayed in the next frame.

Due to high running times associated with *FDP*, optimisation of force calculation is required to reduce the time to generate frames ensure smooth visualisation. Each frame should result in some movement of vertices into lower energy positions, ensuring the differences in layout between frames are not so significant that all readability or familiarity is lost.

Specifically here, **layout generation** refers to the process of generating and visualising static graph layout from some initial layout, allowing a viewer to observe evolution of a layout and identify structures or patterns as it develops. **Layout adjustment** refers to visualisation of changes to a graph as they occur, having been provided an initial layout. As such, changes to the graph are expected to alter the layout. Combination of the two, although possible, would combine the actions of *FDP* to both generate level and correct layout, making measurement difficult.

3.7.2 The Dynamic Graph

The dynamic graph has two components: the base dataset (a typical static graph) and a collection of amendments to be applied over some time period. In some cases, the base dataset may be an empty graph, or the amendments are unknown beforehand (changes occur in an on-line fashion), however, both structures will exist.

In the experimentation here, the base dataset is treated as a static graph and is provided with a layout (through static graph drawing or layout generation) in preparation for anticipated amendments (layout amendment). As amendments occur the layout is then adapted by a process of continuous force directed placement.

The amendments, referred to as **operations**, are included either as a schedule or occur in real time through some user input. A description of the available operations are provided in Table 3.1 and are standard amongst literature (Misue et al (1995), Böhringer and Paulisch (1990), Shannon and Quigley (2007), Papakostas and Tollis (1996), Veldhuizen (2007)).

3.7.2.1 Graph Modification Operations

Operations refer to the modification of a graph which can be the addition, removal or modification of vertices and edges. Operations are performed over time such that the effect can be seen as part of the drawing process, for example, the gradual separation of two vertices after removal of an edge connecting them.

Operation	Application	Description
Addition	Vertex	Adding a new vertex, often accompanied by an edge to connect it to the graph immediately

	Edge	Adding a new edge to connect two vertices, providing both vertices exist and an edge doesn't already connect them
Removal	Vertex	Removal of a vertex and any edges connecting it to the graph
	Edge	Removal of an edge between two vertices
Modification	Vertex	Modification of a vertex or its attributes such as position, weight and label
	Edge	Modification of an edge such as its attributes, weight and label

Table 3.1. Operations available for dynamic graphs in this research

For many of the operations, the graph is queried to determine if the desired change is possible, for example, identifying whether an edge already exists. The associated complexity of each is dependent on the implementation, the data structures used, the algorithmic design and many other factors. More detailed information regarding the complexity here can be found in 10.22 . Complex operations, such as the introduction or removal of a sub graph or the merging of two graphs can be achieved through combinations of operations.

It should be known that the effect of each operation on a layout is dependent on many factors and is largely unpredictable, for example, removing edges randomly will result in the graph becoming increasingly sparse, although some global layout may remain (Figure 3.10 shows the graph 3025, which has had edges randomly removed), eventually disintegrating into disconnected components. In contrast, adding an edge to connect two distant vertices (Figure 3.11 shows the graph 3025, which has introduced edges between opposite corners of the graph) will drastically alter the global layout, affecting readability in a different way.

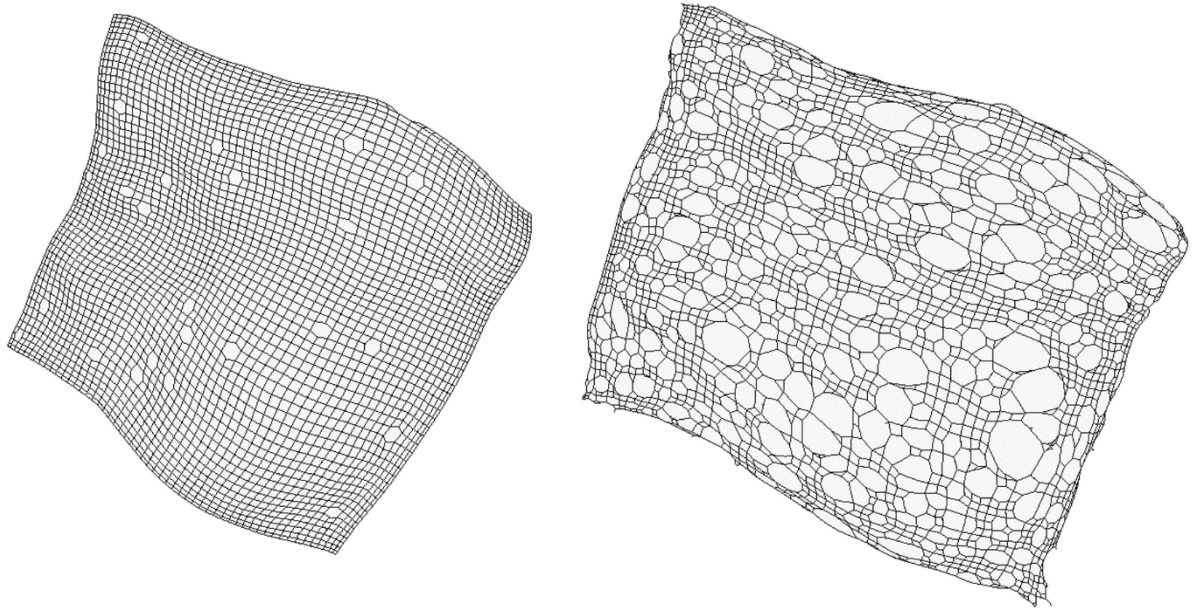


Figure 3.10. The graph 3025 with few (left) and many (right) edges removed, showing preservation of the global layout and degradation of local layouts

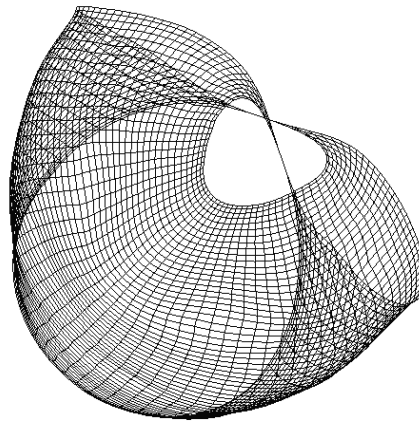


Figure 3.11. The graph 3025 with the addition of two edges connecting opposite corners of the graph a large change in global layout. The graph appears three dimensional due to the folds, however, remains 2D.

In previous literature (Veldhuizen, 2007), amendments to a graph have only been applied to the graph being visualised with no update to the multilevel scheme detailed, and updates to approximation assumed through their usage. As such, the description of such updates are provided in **Dynamic Matching**.

3.7.3 Dynamic Matching

Performing operations on a graph may provide alterations to layout, but only on a local scale. The use of a multilevel scheme can offer layout amendment on a global scale, providing visualisation of the impact those changes have on the dataset as a whole. Dynamic Matching refers to the process of updating the multilevel structure.

Although necessary to retain accuracy for high quality layouts, the update of matchings in the multilevel scheme is an expensive task requiring the analysis of the current scheme to determine whether the changes can be accommodated in the existing structure, or whether alterations are required. In addition, the extent and frequency of operations may alter the multilevel structure so much that identifying an entirely new matching or staggering operations is more efficient.

The act of updating the multilevel scheme is expensive but may also be essential to ensure amendments are incorporated into the global layout generation. Without update the multilevel scheme will retain the approximate structure generated from the original graph, and this could be expected to result in the generation of “ghost movements” (referring to changes in vertex movement as a result of vertices existing in coarser representations, which are no longer in the original graph).

In addition to the update of the multilevel scheme to incorporate changes on a global scale, changes may also alter the multilevel matching in a way which adversely affects layout quality. It can be expected that addition of edges and vertices will result in the ability to make additional branches in the approximation tree, however, removal of vertices and edges can result in fragmentation of the matchings whereby so many vertices are unmatched (matched only with themselves) that the structure of the tree can no longer be considered a good representation of the graph (as shown in Figure 3.12). Although primitive processing provides a means of dealing with this (see Section 3.6.2), methods are required to provide updates to the multilevel scheme which can reduce the effect this may have without other techniques required.

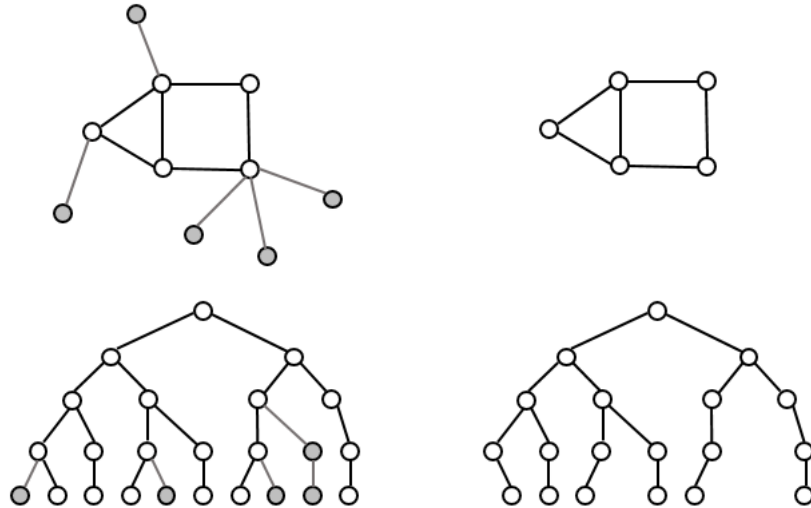


Figure 3.12. Defragmentation of matching as a result of edge and vertex removal

As a result, the following techniques are provided for updating the multilevel structure with amendments made to a dynamic graph, with illustrations of their usage for the addition of two vertices and two edges to an example graph below (Figure 3.13).

- Low-level – changes are only applied to the original graph with only weightings taken into account in the multilevel scheme (Figure 3.14)
- Mid-level – changes apply to half of the levels of the multilevel scheme, with operations being accounted for up to the middle graph, preserving the scheme (Figure 3.15)
- High-level – changes are fully incorporated into the multilevel scheme but the aim is to preserve as much of the original scheme as possible (Figure 3.16)
- Rematch – the entire multilevel scheme is scrapped and generated anew (Figure 3.17)

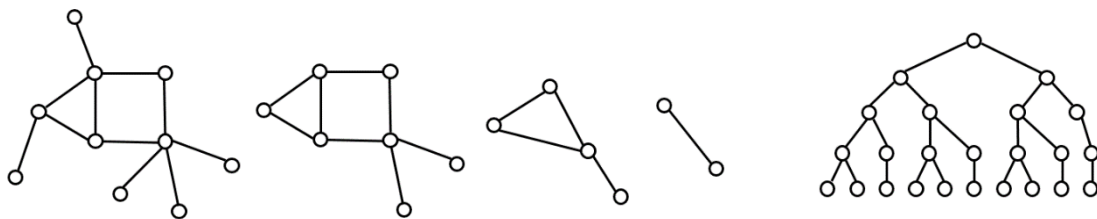


Figure 3.13. Given an example graph (in this case, part of the London Underground), and generation of a multilevel scheme using edge contraction (left), a resulting *MGF* tree is generated automatically (right)

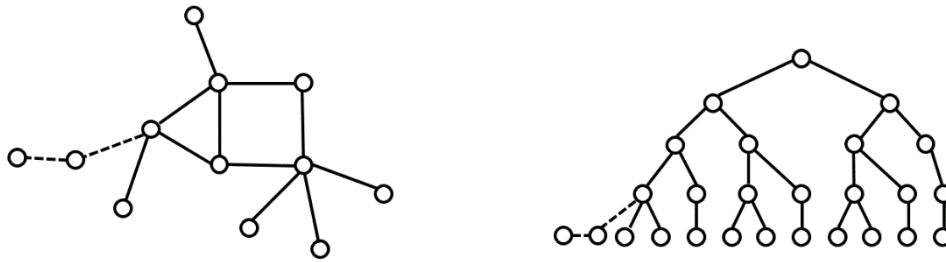


Figure 3.14. No multilevel update for the addition of two vertices and two edges to the graph in Figure 3.13, showing the immediate matching of the two edges and their addition into the multilevel scheme by sharing the parent of the anchoring vertex

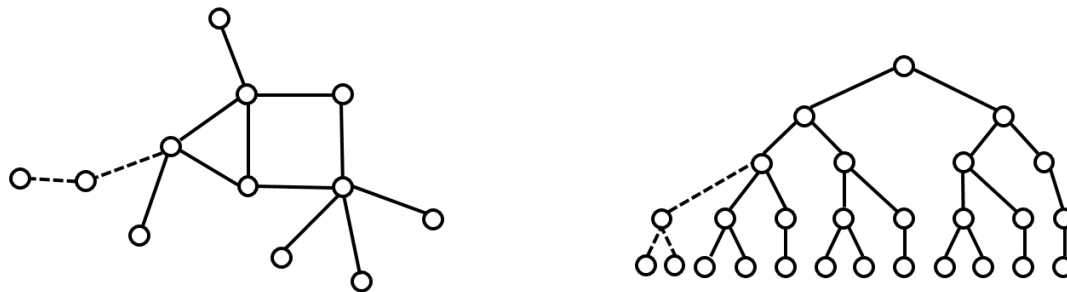


Figure 3.15. Half-update of the multilevel scheme for the addition of two vertices and two edges to the graph in Figure 3.13, showing the incorporation of the vertices as typical matchings up to the 3rd (mid) level, whereby they are joined with the parent of neighbouring matches (identified through traversal of the tree from the anchored vertex)

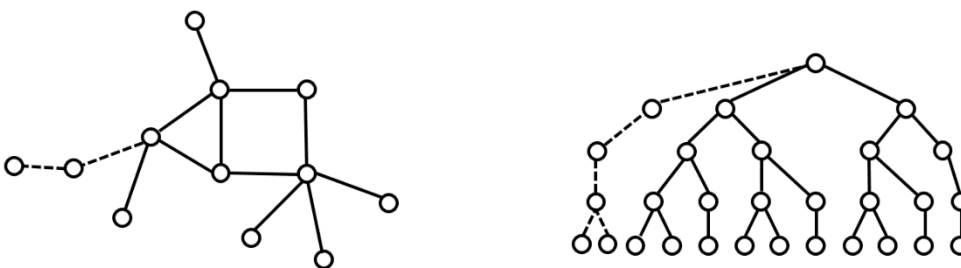


Figure 3.16. Full update of the tree, and coincidentally the same structure as generated by an entirely new matching, of two vertices and two edges added to the graph in Figure 3.13. The tree is updated until an unmatched coarse vertex is identified with which the new matchings can be matched (found at the 4th level). If no matching is available, a new coarsest graph is generated and the matching is matched with itself.

These four update methods are suggested for incorporation of operations into the multilevel scheme, offering differing extent of updates for both multilevel and approximation methods. They offer different controls over the incorporation of changes in global layouts, with a full

update expected to have high impact on global layout, and low-level update expected to preserve global layout.

3.7.4 Dynamic Modified Spring Embedder

As mentioned previously, the Spring Embedder is the approach of interest here, and specifically the modification described by Fruchterman and Reingold (1991), for its ability to overcome minima and provide high quality layouts quickly. Although the heuristic suggested by Eades (1984) can provide the expected high quality layouts and runs continuously, the method is prone to getting trapped in local minima. Additionally, the modified spring embedder has previously been used with multilevel schemes and approximation, making transition of those into dynamic drawing simpler.

The Fruchterman-Reingold method is not viable for use in dynamic graph drawing in its original form due to the method of calculating vertex displacement. The approach is designed to allow for large movements in early iterations, which are achieved through use of strong forces to move vertices large distances relative to the ideal edge length. The forces are so strong however, that as a vertex moves into a new position in one iteration, opposing forces will move the position back in the next. This can result in the oscillation of a vertex between positions, with use of a cooling schedule to limit the movement in each iteration, so that as forces push the vertex one way, when pushed back the movement is reduced. This continues until the movement is less than some tolerance and the algorithm terminates. The erratic movement makes visualisation difficult to follow with the cooling resulting in quick convergence. After the cooling scheme has finished, any subsequent operations on the graph will not result in layout adjustment.

Methods for resetting the cooling schedule (such as quenching, as described by (Davidson and Harel, 1996) allow for the refinement of layout providing the incorporation of dynamic operations, however visualisation is similarly erratic making changes difficult to follow.

Experimentation has shown that replacing the movement limit provided by the cooling schedule (a changing value) with a constant value provides continued movement of vertices, allowing for continued adaptation of layout and the ability to include further changes. The use of a constant value also removes the termination clause allowing for continual layout

development, however oscillation in vertex movement is still exhibited (albeit reduced). The greater the limit on vertex movement, the less pronounced the oscillation and the smoother visualisation, but the more iterations are required for the layout to converge.

Delving deeper into the literature led to an implementation for use on GPUs which visualises graph generation in the intended fashion, however no research documents accompanied the work. Correspondence with the author (Daniel Danko, 2013, Appendix 10.3) suggested the use of individual cooling schedules for the spring and repulsive forces. Through some brief experimentation it was found that removing the cooling schedules altogether in favour of separate constants provided a method to stop the oscillation in vertex movement and retain sufficient force to provide vertex movement and overcome minima. Due to the removal of the cooling schedule, large initial movements used to escape minima are stopped in favour of gradual movement (closer resembling the Eades (1984) implementation).

Displacement of a vertex v , as a result of a vertex u , can be simplified as:

$$\Theta := f_a(v,u) - f_r(v,u)$$

$$v.xyz := v.xyz + (cool \cdot \Theta)$$

The proposed method seeks to alter this

$$\Theta_a := f_a(v,u)$$

$$\Theta_r := f_r(v,u)$$

$$v.xyz := v.xyz + ((cool_a \cdot \Theta_a) - (cool_r \cdot \Theta_r))$$

Where $cool_a > cool_r$ to avoid the graph infinitely expanding due to overpowering repulsive forces.

Unlike traditional cooling which would iteratively decrease the cool value (for example, $cool := cool \cdot \lambda$ where $\lambda := 0.9$), $cool_a$ and $cool_r$ are constants to allow continued vertex movement. The cool parameters are provided with different values in order to find an equilibrium point at which the oscillation stops and layout is generated until some low energy layout is provided.

The Dynamic Modified Spring Embedder can be run indefinitely, adjusting layouts as operations occur. Pseudocode is provided in Appendix 10.22.4 .

3.7.5 Multilevel Layout

As with static drawing algorithms, multilevel schemes are suggested to provide and maintain a global layout. However, due to the requirement to visualise changes to vertex movement gradually, the same methodology cannot be applied (finding a layout of the coarser graphs first and interpolating the completed layout). Instead an adapted approach is required which allows layout to be interpolated after each iteration of force directed placement. In essence, the suggestion presented here is a method which applies *FDP* to all levels of the multilevel simultaneously.

Unlike the static implementation, finer vertices are unable to take the position of coarser vertices or risk overwriting layout provided by local force directed placement (or conflicting movements). Instead, it is suggested that vertex movement is interpolated, rather than vertex position, allowing for finer vertices to be provided layout by moving them in the same direction as their coarser approximation.

The change in vertex becomes:

$$v.xyz := v.xyz + (v.\Theta) + (v.parent.\Theta)$$

Interpolation of movement requires vertices in the finer graph to have positions near those of its coarser counterparts in order for the movement to be applicable and useful for global untangling of the layout. If positions are not similar enough, the movement may result in vertices placed in poor positions resulting in high energy positions. Figure 3.17 gives an example of the interpolation of movement, whereby when the triangular vertex of a coarse graph is moved, the triangular vertices it represents in the finer graph are provided the same movement (different shapes are used to identify parent and child vertices between the graphs). Figure 3.18 shows the same interpolation of movement, however vertices in the finer graph have positions which do not match their coarser parents in the coarse graph, resulting in the interpolated movement depreciating the layout, highlighting the importance of proximity of vertices in fine and coarse graphs.

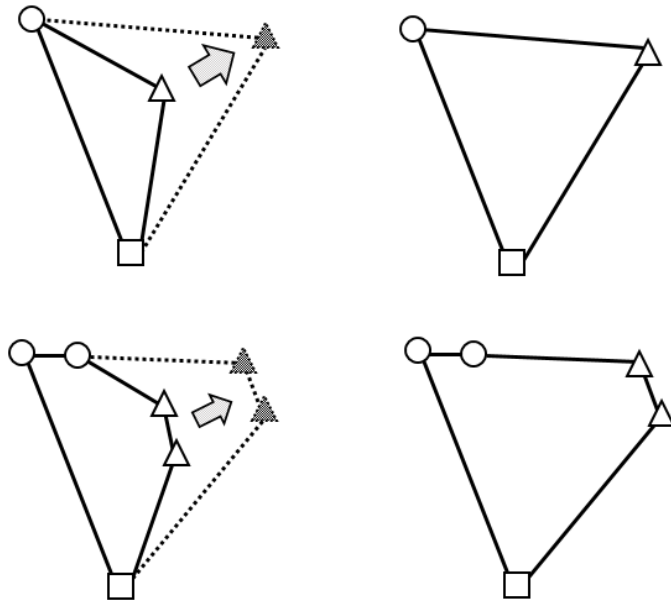


Figure 3.17. Interpolation of vertex movement between two graphs in the multilevel scheme, incorporating the global layout achieved in coarse graphs (top) into finer graphs (bottom). Shapes are used to identify parent and child vertices between the two graphs such that two triangular vertices are represented by one triangular vertex in the coarse graph.

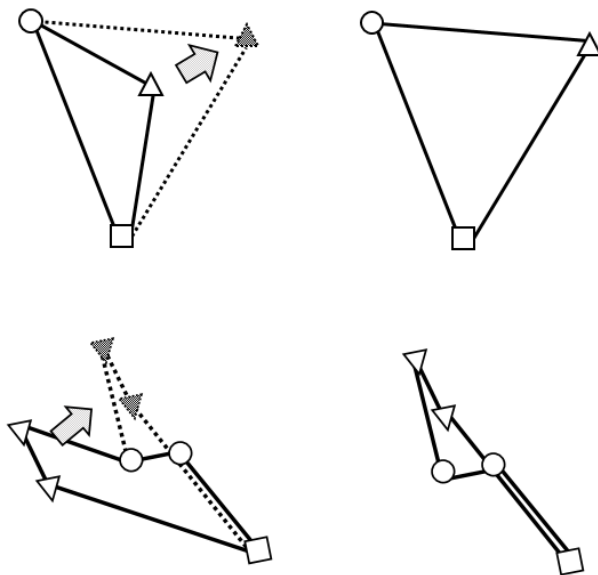


Figure 3.18. Interpolation of movement as provided in Figure 3.17, incorporating the global layout achieved in coarse graphs (top) into finer graphs (bottom), however, due to positions of vertices in finer graphs not matching those in coarser graphs, interpolation of vertex movement results in poor

placement in the coarser graph. Shapes are used to identify parent and child vertices between the two graphs such that two triangular vertices are represented by one triangular vertex in the coarse graph.

Calculation of vertex movement is performed through application of force directed placement, however, due to the requirement that changes to layout are visualised gradually, vertex movement must remain small enough not to risk altering the layout too quickly. To achieve this, force directed placement is applied to each level only once, preventing the smaller coarse layouts converge before finer counterparts which take longer to converge.

The result is a technique which provides a gradual amendment of layout, making generation and adjustment easier to follow.

Although the algorithm is not graceful, the approach provides a heuristic for multilevel drawing which can be considered a “brute force” interpretation of the multilevel drawing process, open to improvements, adaptations and optimisation.

3.7.6 Approximation using *MGF*

Building upon incorporation of the multilevel scheme, Force Directed Placement can be combined with Multilevel Global Force approximation to reduce the running time required per generated frame, allowing for large graphs to be visualised smoothly and providing improved global representation of operations to the graph. Other methods for approximation can be used, for example an Octree (Veldhuizen, 2007), but the nature of Multilevel Global Force provides direct incorporation of operations into multilevel and approximation scheme, impacting the resulting global force calculation.

The expected result is a large reduction in running time with minimal changes to layout, and in a more practical sense, a simpler implementation (vertex positions do not have to be tracked nor do vertices need reassignment to other sections of the approximating grid). However, Multilevel Global Force requires alteration for use in dynamic graph drawing when multilevel refinement is also used.

Given a static graph and multilevel scheme, layout is generated for each successive coarse graph and interpolated, after which the coarser layout is no longer required and can be overwritten by *MGF* for approximation. In dynamic drawing, layout is interpolated after each iteration of Force Directed Placement, requiring the layouts of coarser graphs to remain

intact for future amendments to the layout. As a result, coarser positions should not be overwritten with an approximation of finer vertex positions, or will result in conflicts in positions generated using multilevel *FDP* and *MGF* approximation (an example of the difference in positions is shown in Figure 3.7).

A solution is to use two coordinate systems: one to store multilevel layout and another for approximation (similar to having two multilevel schemes; and one for multilevel layout and another for approximation). A post order traversal of the multilevel scheme (bottom upwards from an initial graph to its coarser abstraction) to update the *MGF* tree will then only update those positions designated for approximation, leaving the other for multilevel layout generation.

3.7.7 Summary of Dynamic Graph Drawing

The aim of the presented research into dynamic drawing methods is to improve the methods currently used in order to allow for larger graphs to be visualised. As such the following have been proposed and discussed:

- Visualisation is achieved through frames which can be observed as an animation
 - Layout generation refers to the generation of layout for a graph with no prior layout known (initial positions are random)
 - Layout adjustment refers to amendment of an existing layout to provide updated visualisation of changes resulting from graph modification operations
- Modifications of a graph with details regarding the extent of the impact on the multilevel structure as a result:
 - No-update – changes occur only to the original graph
 - Mid-update – changes occur in half of the multilevel scheme
 - All-update – changes are accounted in all of the multilevel scheme
 - Full-rematch – the multilevel scheme is re-matched and replaced
- A modification of the Fruchterman and Reingold (1991) spring embedder for use in dynamic graph drawing, splitting the calculation of vertex displacement for individual forces such that equilibrium between the displacements can be altered to provide smooth vertex trajectories

- Making use of efficient forces able to better overcome minima
- Simplified transition of static drawing techniques to dynamic drawing
- Adaptation of multilevel scheme is suggested providing a “brute force” usage which offers global layout generation through interpolation of vertex movement
- Use of the *MGF* approximation is described with adaptation for usage with multilevel schemes in dynamic drawing

3.8 Summary

This chapter covers three main topics of research in which the contributions of the thesis are based: Graph Aesthetics, Multilevel Global Force (approximation) and Dynamic Graph Drawing. The three subjects are combined to form a framework for experimentation. In regard to the problem area identified, the following have been suggested in order to tackle some of the challenges identified in literature:

- An efficient force calculation offering minimal cost (**Multilevel Global Force**)
 - Combining approximation of space (for example, Octree) and structure (for example, multilevel) for calculating global interactions between all pairs of vertices efficiently using structural information within the graph
 - Additional methods for altering the structure of the approximation scheme and improve incorporation of structural information
 - Multi-matching
 - Pattern processing
- Optimization of layout visualisation and adjustment for large graphs such that changes are visualised immediately (**Dynamic Graph Drawing**)
 - Definition of Dynamic Graph Modification Operations (addition, removal and modification of vertices and edges) and Visualisation through use of frames generated per iteration of force directed placement and joined to form an animation
 - Adapted Visual Spring Embedder for improved layout convergence and better transition of multilevel and approximation (static methods) into dynamic drawing methods

- Approximation introduction and adaptation (*MGF*) for either usage on the original graph or in combination with a multilevel scheme
- Global layout and local layout generation simultaneously via multilevel integration (**Dynamic Graph Drawing**)
 - Multilevel adaptation for usage in dynamic graph drawing providing local and global layout simultaneously
 - Protocols for efficiently updating multilevel schemes when operations are applied to a graph
- Definition of aesthetics affecting large and huge graphs (**Measuring Layouts**)
 - Application of graph aesthetics in combination of a multilevel scheme to analyse and compare global and local layouts of large graphs
 - Usage of static graph aesthetics as metrics for measuring quality of and comparing layouts:
 - number of edge crossings in the layout;
 - average and range of edge lengths in a layout (identifying the peripheral effect within layouts)
 - Measuring the metrics for graph stability through monitoring and analysing average and maximum vertex movement and edge crossings between frames in dynamic drawing
 - Although not a metric of quality, running time is also measured for comparison of algorithm performance

4 Multilevel Aesthetics: Experimentation and Results

The experimentation looks to test whether the aesthetic criteria associated with good layouts for smaller graphs can be applied to approximation of layouts for larger graphs to indicate the quality of global layout. Three approaches to measuring multilevel aesthetics are investigated:

Global Layout Analysis	Analysis of the global layout as modelled by the Multilevel scheme, comparing the measured quality (reduction in edge crossings between multilevel and single FDP methods) to identify if a multilevel abstraction (G_N) can be used to measure global layout in the original graph layout (G_0)
Local Partition Analysis	Analysis of the local layout quality in the partitions of vertices which are represented as single vertices in a coarse representation of the graph (G_N), to identify if the multilevel abstraction can be used to better measure local level layout numerically in the original graph layout (G_0)
Multilevel Analysis of Layout Generation	Analysis of developing layouts using multilevel abstraction, to identify if and how the local and global layouts change for differing FDP implementations over time, and if the method can offer further insight into the calculation of emerging layouts and the relationships between the data within

The quality of the generated layouts are compared on two levels, firstly measuring aesthetics for the layout as is (G_0), and then for an approximation (G_N) of the layout as generated by the multilevel scheme ($G_N = G[|G_L|/2]$). The results are then compared. Further examination of the quality of partitions is also provided, intending to record the quality of local layouts.

Dynamic layouts are also analysed to determine if the same approach can be taken for dynamic graphs, particularly in the interest of the metrics for graph stability. Three drawing methods are compared for layout generation:

- a spring embedder heuristic (Eades, 1984)
- adaptation of the modified spring embedder (Section 3.7.4)
- multilevel adaptation of the modified spring embedder (Section 3.7.5)

Each test graph is given an initial random layout, used by each algorithm. Layouts are generated over a period of 300 iterations and the quality measured for the layout as is (G_0), and an approximation of the layout as generated by the multilevel scheme (G_N). Tests are performed on the static test graphs and are repeated 5 times.

4.1 Analysis of Numerical Results

4.1.1 Global Layout Analysis

The measured quality of layouts for the test graphs are summarised in Table 4.1. As expected, the drop in edge crossings as a result of multilevel usage (G_0) is also observed in the approximation of layout (G_N), with a drop of 81% and 77% respectively. Full results can be found in Appendix 10.23.1 .

The methods tested are as follows:

- *FR* - Fruchterman-Reingold Spring Embedder
- *MLFR* - Multilevel Fruchterman-Reingold Spring Embedder

Graphs	Edge Crossings for G_0			Edge Crossings for G_N		
	FR	MLFR	reduction	FR	MLFR	reduction
3025	11983.60	833.40	93.05%	141.40	15.20	89.25%
data	113308.60	39032.40	65.55%	220.80	135.40	38.68%
add32	28507.60	12065.40	57.68%	126.20	34.80	72.42%
4elt	304280.00	21945.40	92.79%	968.40	126.00	86.99%
sierpinski10	691717.20	19069.20	97.24%	1033.60	6.80	99.34%

Table 4.1 Comparison of the number of edge crossings exhibited in the layouts, and approximation of layouts, for several test graphs. Two algorithms are used to generate the layouts; Fruchterman-Reingold Spring Embedder and Multilevel Fruchterman-Reingold Spring Embedder, with the latter utilising a multilevel scheme to improve global layout, reducing the number of edge crossings in the approximate layouts.

The results meet the expectations of the author, the multilevel scheme reduces the edge crossings in global layout, and supports analysis of multilevel aesthetics for comparing algorithm performance for larger graphs. However, the results here only confirm findings which can equally be found by analysing the layout as it is (using a multilevel scheme reduces

edge crossings), it is proposed that multilevel aesthetics analysis is more useful for comparing layouts from algorithms with unknown behaviours (See Dynamic Multilevel Aesthetics Analysis below) in addition to local layout analysis.

4.1.2 Local Layout Analysis of Partitions

Measurements recorded for local layout analysis are provided in Table 4.2. The results indicate that average edge crossings in local layout partitions show a similar drop to that seen in the graphs normal layout in Table 4.1 offering little insight over analysis of the layout as it is.

Graph	Average Partition Size		Average Edge Crossings		reduction
	vertices	edges	FR	MLFR	
3025	74.52	146.33	43.48	3.27	92.48%
data	80.47	425.98	1023.06	618.63	39.53%
add32	77.32	147.49	129.72	114.86	11.46%
4elt	142.15	417.88	321.78	34.63	89.24%
sierpinski10	361.66	723.32	492.75	69.19	85.96%

Table 4.2. Comparison of the quality of local layout, measured as the edge crossings exhibited in partitions of a graph generated by a multilevel scheme. Two algorithms are used to generate layouts for a selection of the test graphs, the Fruchterman-Reingold Spring Embedder and the Multilevel Fruchterman-Reingold Spring Embedder, expecting both to have similar local quality.

The range in edge crossings provide a more meaningful picture, displayed in Table 4.3, showing that both methods are capable of finding good local layout quality with 0 edge crossings, however the multilevel algorithm is capable of reducing the maximum number of edge crossings. Although not explicitly achieved through better global layout, it identifies that local layout is achieved in both algorithms but the multilevel scheme minimises tangles in some areas.

Graph	FR		MFR	
	max EC	min EC	max EC	min EC
3025	168.4	0	33	0
data	2753.2	0	1823.8	0
add32	559.8	0	416.6	0
4elt	1101.2	0	458.6	0
sierpinski10	1483	0	232.2	0

Table 4.3 Comparison of the range of edge crossings exhibited in partitions of a graph generated by a multilevel scheme. Two algorithms are used to generate layouts for a selection of the test graphs, the Fruchterman-Reingold Spring Embedder and the Multilevel Fruchterman-Reingold Spring Embedder, expecting both to have similar local quality.

4.1.3 Multilevel Analysis of Layout Generation

Plotting the change in edge crossings for local and global layout are more interesting, as depicted in Figure 4.1 and Figure 4.2.

Figure 4.1 shows the reduction in edge crossings for the graph regular-medium as layout is generated from an initial random layout (quality is assessed every 20 frames). The chart shows each of the three algorithms effectively reduce the number of edge crossings over 300 frames, but allow the reader to the different rates.

There is little indication as to what's causing the difference in the data alone, it is expected that global layout is generated more quickly for a multilevel algorithm than a standard spring embedder, but from the results this is not clear and assessment is based on subjective analysis alone. Multilevel analysis provides additional results which can support a more informed analysis, for example, Figure 4.2 provides the multilevel analysis for the layouts represented in Figure 4.1, showing that the approximate global layout for the Eades Spring Embedder contain far more edge crossings.

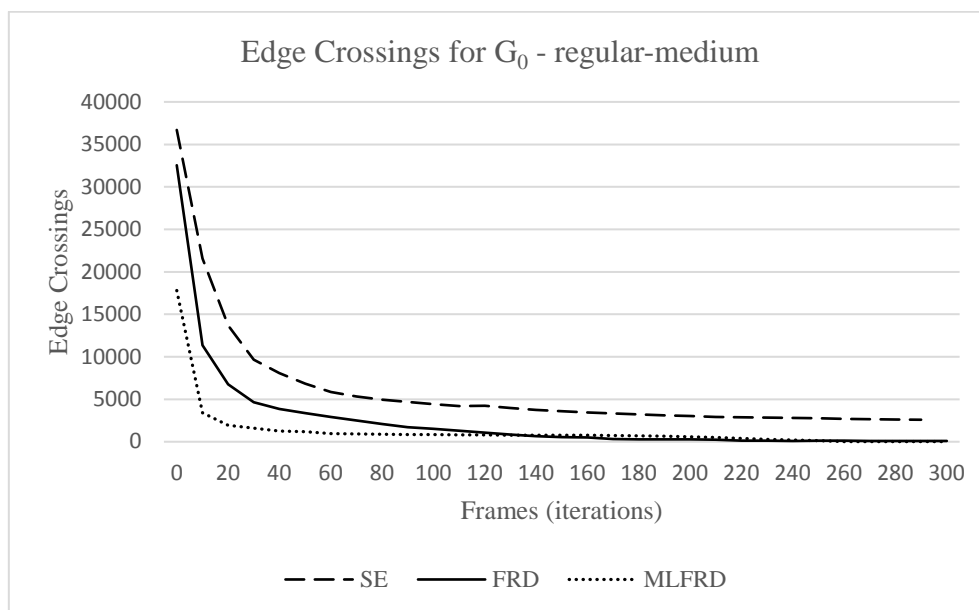


Figure 4.1. Change in edge crossings for the layouts of the graph regular-medium provided by three algorithms, Eades Spring Embedder, Dynamic Fruchterman-Reingold Spring Embedder and a Multilevel Dynamic Fruchterman-Reingold Spring Embedder.

Further inspection of the chart suggests that the Dynamic Spring Embedder is actually more adept at finding a global layout than its multilevel counterpart, finding a global minimum in

fewer frames. This immediately asks questions which may not have arisen from the standard results in Figure 4.1 (such as, why is the performance of the multilevel layout poorer?)¹.

Edge crossings are measured here, however, many aesthetic criteria can be measured in the approximate layout, such as edge lengths and average vertex movement.

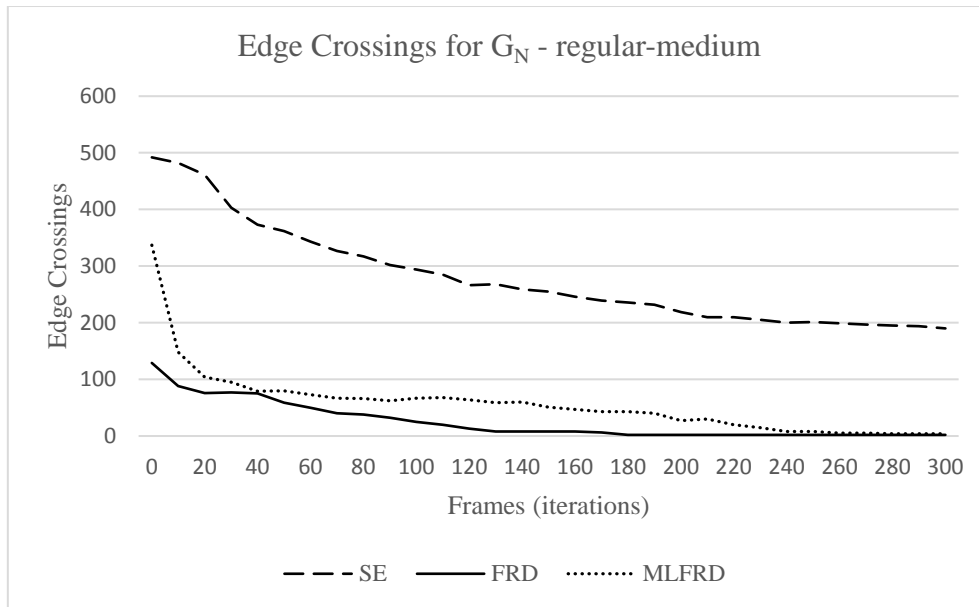


Figure 4.2. Change in edge crossings for the development of approximate layouts of the graph regular-medium provided by three algorithms, Eades Spring Embedder, Dynamic Fruchterman-Reingold Spring Embedder and a Multilevel Dynamic Fruchterman-Reingold Spring Embedder.

4.2 Subjective Analysis

4.2.1 Static Graphs

Although numerical results are the primary interest in order to facilitate comparison of layouts from different algorithms, analysis of the layouts from the authors' viewpoint is also provided to ensure the measured results follow the observable results. As a representation of the methods function, analysis of layouts for 3025 are provided as an example (due to its known 55x55 vertex grid structure).

¹ Further investigation shows that the size of the graph means the standard Dynamic Spring Embedder is able to find a global and local minimum. Larger graphs prove more difficult for the algorithm, where the Multilevel Dynamic Spring Embedder succeeds.

The approximation of layout is shown in Figure 4.3, achieved by generating a multilevel scheme and approximating vertex positions through a postorder traversal of the matching tree. Note that both layouts exhibit zero edge crossings, and how the coarse graph exhibits the same elongated layout. Similarly, if a layout was to include an imperfection such as a fold or a twist, the approximate layout is expected to also exhibit the imperfection, shown in Figure 4.4.

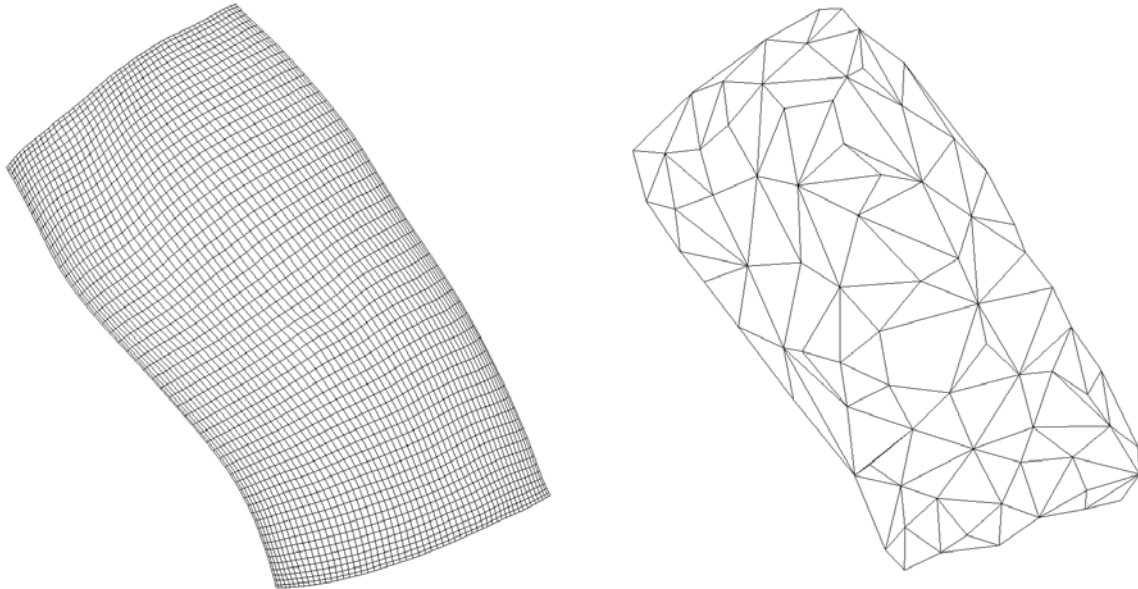


Figure 4.3. A layout for the graph 3025 (left) generated by a Multilevel Fruchterman-Reingold Spring Embedder, and an approximation of the layout generated by a multilevel scheme (right) representing the global structure of the graph.

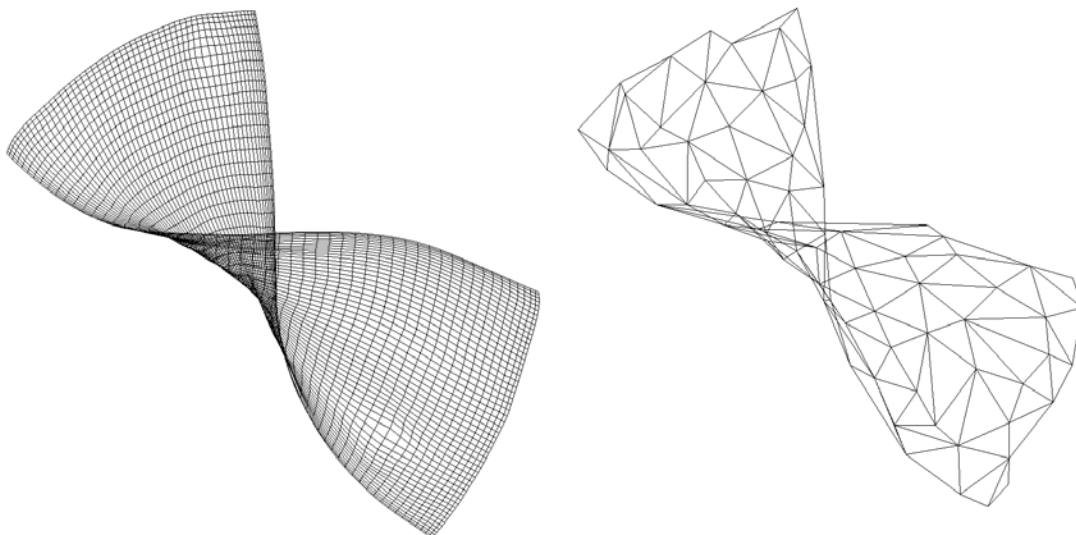


Figure 4.4. A layout for the graph 3025 (left) generated by a Multilevel Fruchterman-Reingold Spring Embedder exhibiting a severe twist in the layout, and an approximation of the layout generated by a multilevel scheme (right) representing the global structure of the graph showing a similar twist.

The same expectation applies to extreme variations in layout, for example, Figure 4.5 shows a layout with little global layout, represented as a tangle of edges in the approximate layout. Although a poor layout in comparison to those above, there is still noticeable areas of the graph where good local layout has been generated, suggesting that the algorithm used to draw the graph has specific trouble with global layout (supporting the numerical analysis above).

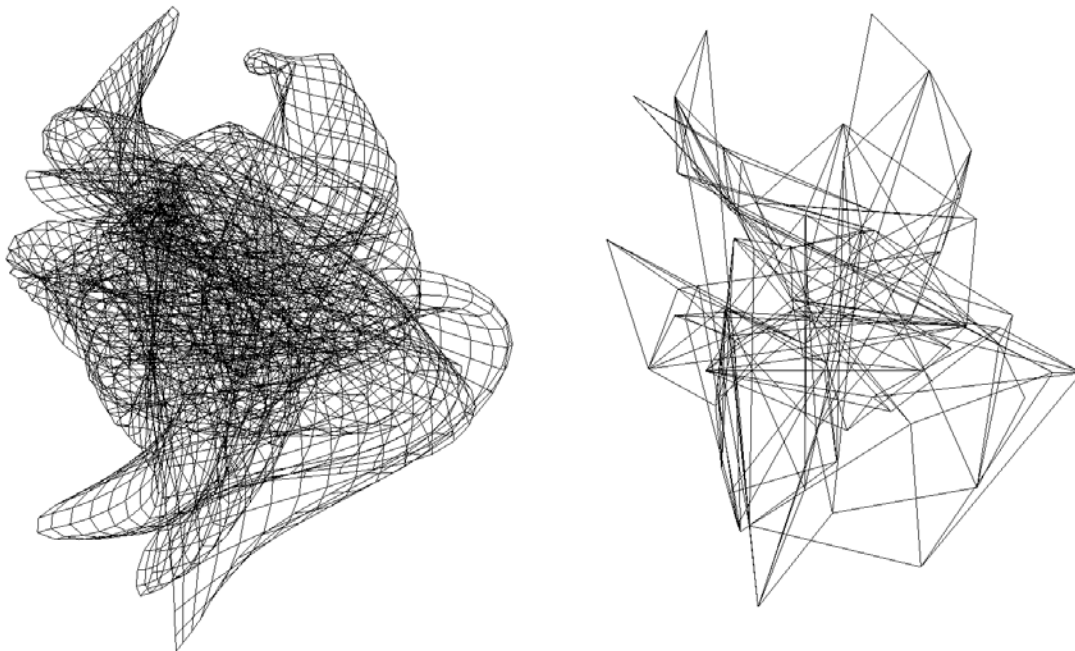


Figure 4.5. A layout for the graph 3025 (left) generated by a Fruchterman-Reingold Spring Embedder exhibiting poor global layout (folds and twists in the layout), and an approximation of the layout generated by a multilevel scheme (right) representing the global structure of the graph showing similar bends and twists of the layout.

Analysing the partitions generated using the multilevel scheme (those groups of vertices represented by one vertex in the approximate layout) through Multilevel Analysis provides a way of representing this numerically such that it can be compared across algorithms. Figure 4.6 illustrates such partitions for the layouts in Figure 4.3 and Figure 4.5.

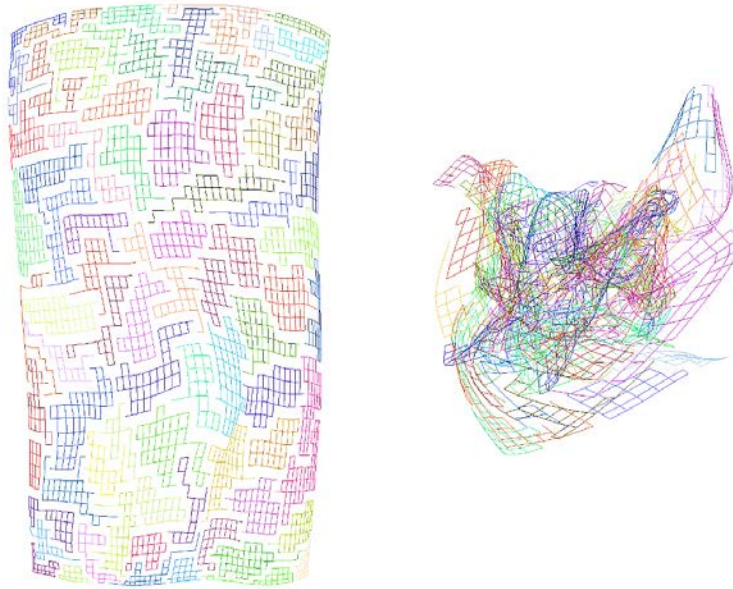


Figure 4.6. Two layouts for the graph 3025, generated using the Fruchterman-Reingold Spring Embedder (left), and by the Multilevel Fruchterman-Reingold Spring Embedder (right), colored to show the partitions used to measure local layout.

4.2.2 Development of Layout

Illustrating multilevel analysis for a dynamic drawing algorithm is attempted through use of frames in Table 4.4 showing the changes to layout for the graph regular-medium as measured in Figure 4.1.

Column A of Table 4.4 shows the initial positions of vertices within the layout at frame 0 (before *FDP* is applied). Note that the vertices in the approximate layout match the positions of their coarser counterparts in the actual layout. The number of edge crossings are provided under each image for context. Columns B through to E show the improvement of layout after 10, 40, 200 and 300 iterations.

As mentioned above, the primary aim of multilevel analysis is numerical foundation for assessing algorithms, however, observation of layout and global layout development can provide insight into what's happening during the drawing process. Visualising the drawing process only highlights local layout being generated quickly with the majority of frames spent overcoming global layout. Comparison of the developing global layout is believed to be useful when comparing algorithms due to identifying which is able to better overcome global minima (such as the Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder which provide unexpected results in Figure 4.2).


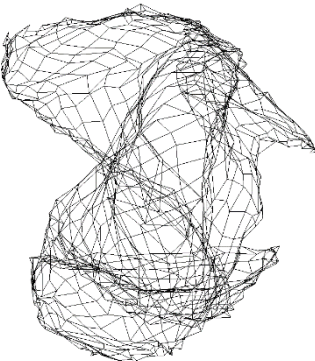
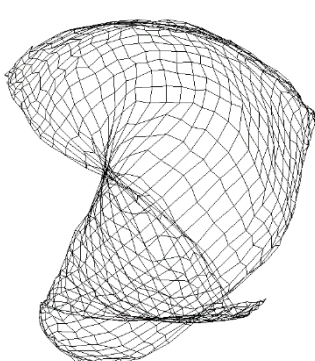
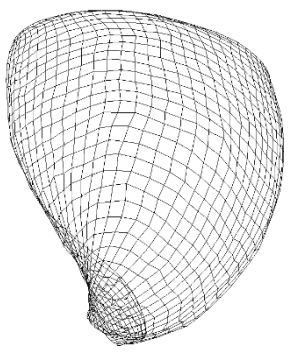
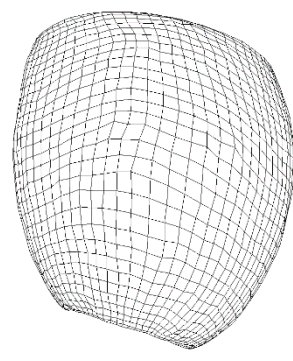
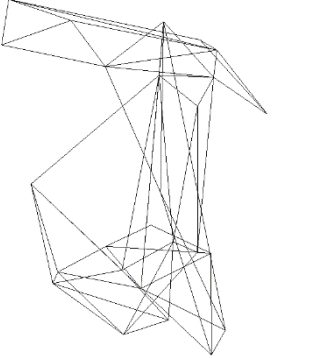
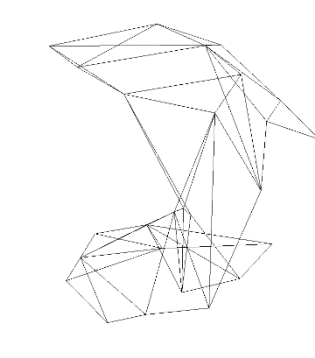
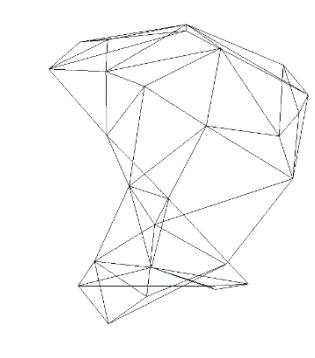
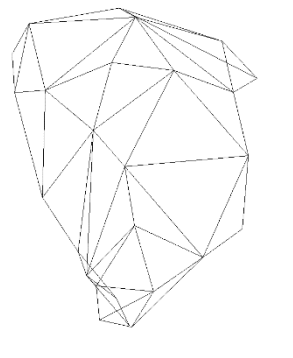
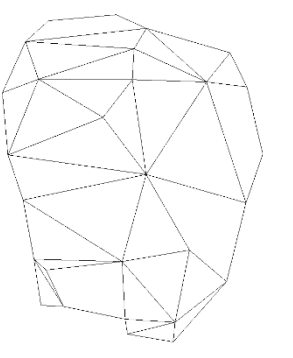
A (Frame 0)	B (Frame 10)	C (Frame 40)	D (Frame 200)	E (Frame 300)
				
17799 Edge Crossings	3402 Edge Crossings	828 Edge Crossings	122 Edge Crossings	25 Edge Crossings
				
117 Edge Crossings	74 Edge Crossings	32 Edge Crossings	13 Edge Crossings	0 Edge Crossings

Table 4.4. Figures showing the improvement of layout for the graph regular-medium, with multilevel interpretation of global layout showing the development of approximate global layout for the Multilevel Dynamic Fruchterman-Reingold Spring Embedder

4.3 Summary

The experimentation supports use of Multilevel Aesthetic Analysis for comparing algorithms as it provides some numerical backing to assessment of algorithm performance and behaviour. Investigation into Static and Dynamic graph drawing provides examples for such usage across both fields.

Although promising, the method is likely to be useful for only thorough analysis and comparison of layouts. In a largely empty area, the method provides some way of extending aesthetics of smaller graphs to larger graphs, however, shows that similar conclusions can be achieved by measuring the criteria in the layout as it is.

5 Static Graph Drawing: Experimentation and Results

This chapter describes the experimentation and results specific for Static Graph Drawing, aiming to investigate the methods for reducing the complexity of Multilevel Force Directed Placement with as little cost to layout quality as possible, as described in Section 3.6 of the Concepts chapter. If any efficiency savings are found, the intention is to extend them to dynamic graph drawing in order to reduce the running time of the algorithms there, such that larger dynamic graphs can be given layout.

The algorithms which are investigated are:

Multilevel Spring Embedder with Multilevel Global Force Approximation	Crawford (2012)
Multilevel Global Force Spring Embedder with Multimatching (<i>MM</i>)	Extends the work of Crawford (2012) Aims to optimise the matching scheme used to generate the <i>MGF</i> through matching groups of vertices instead of edge contraction.
Leaf Placement Scheme (<i>LPS</i>) Combinations: <i>LPS</i> applied after <i>MGF</i> <i>LPS</i> followed by <i>MGF</i> refinement <i>LPS</i> applied after <i>MGF</i> on each level of the multilevel scheme <i>LPS</i> applied after <i>MGF</i> on each level of the multilevel scheme with further <i>MGF</i> refinement	Extends the work of Crawford (2013). Aims at exploiting contraction of loosely connected vertices by giving them an estimated layout to reduce complexity of <i>MGF</i> and/or improve layout quality.
Primitive Matching (<i>PM</i>)	Used with <i>MGF</i> . Aims at matching common patterns which can be given a calculated layout (Stars, Lines, Rings) in an attempt to improve layout quality of coarsest graphs of

	the multilevel scheme (such that it improves final layout quality).
Multilevel Global Force Spring Embedder with Approximation Limit (<i>MGFL</i>)	Investigates limiting the effect of forces in <i>MGF</i> , using plotted distance and theoretical distance (between vertices in the approximation scheme) to reduce complexity.

5.1 Test Data – Static Graphs

The test data used for experimentation is split into two collections: static test graphs and dynamic test graphs, derived from examples in literature.

5.1.1 Static Test Graphs

Static drawing methods are tested using a collection of large graphs found in literature (Walshaw (2003), Hu (2005)), described in Table 5.1. The test graphs are taken from The Graph Partitioning Archive maintained by Dr Chris Walshaw, and have previously been used in Walshaw (2003) and Hu (2005) to demonstrate the drawing capabilities of their drawing algorithms (with the exception of 3025, which is a generated grid of 55x55 vertices). Original images of the graphs and their available sources can be found in Appendix 10.21.5 .

Graph	Vertices	Edges	Average Degree
3025	3025	5940	3.9
add32	4960	9462	3.8
data	2851	15093	10.6
4elt	15606	45878	5.9
sierpinski10	88575	177147	4.0
finan512	74752	261120	7.0
dime20	224843	336024	3.0
mesh100	103081	200976	3.9

Table 5.1. A collection of static graphs used for experimentation of static graph drawing methods. The number of vertices, edges and the source of each graph are provided. Actual layouts for these graphs are provided in Appendix 10.21.5 .

Graphs are available from Walshaw’s research website, The Graph Partitioning Archive (staffweb.cms.gre.ac.uk/~wc06/partition/), with the exception of 3025 which is a regular 55 by 55 vertex grid. Graphs are stored using the Chaco format (Hendrickson and Leland, 1995).

5.1.2 Leafy Test Graphs

Experimentation of some techniques require additional test graphs in order to test for specific behaviour. Leafy graphs are those which exhibit leaf vertices (described in Appendix 10.9), and are used for testing pattern processing. A collection of leafy graphs are described in Table 5.2. The graphs were obtained from Dr Chris Walshaw, and are examples of networks and sparsely connected graphs, which are typically “leafy”.

Graph	Vertices	Edges	Leaves
anna	138	493	25
david	87	406	10
dyads_lc	195	203	83
dyads2_lc	478	922	216
gd96D	180	228	58
websiteCMS	498	1871	71
uk	4824	6837	65
sn6121	6121	24331	1423
add20	2395	7462	123
add32	4960	9462	292

Table 5.2. A collection of graphs known to have one or more "primitive leaf" vertices for experimentation with pattern recognition algorithms. The number of vertices, edges and leaf vertices for each graph are provided.

5.1.3 Layout Quality and Metrics

Layout quality is identified through measurement of graph aesthetics (described in Concepts, Chapter 3) and running time is calculated as the entire time spent generating a layout (static) or an individual frame (dynamic).

Edge crossings are the primary measurement for layout quality, acquired by counting the intersecting edges in a layout. Edge lengths are also recorded as a secondary measure of quality (see Graph Aesthetics, 2.4.3).

Tests are repeated to measure average performance and layout quality, with each experiment indicating the number of repeats (typically 10). Ranges across repeats may also be recorded to compare the regularity of layouts for different algorithms.

Subjective analysis of layouts of layouts is provided for all experimentation, identifying any significant differences between methods or irregularities within layouts, and testing conclusions made from numerical analysis.

5.2 Multilevel Global Force (*MGF*)

Can the multilevel scheme be exploited for use as accurate approximation of both layout and structure, providing a tree for approximating long range global forces in Force Directed Placement?

Multilevel Global Force (*MGF*) is implemented through exploitation of an existing multilevel scheme. The multilevel scheme used following that described by Walshaw (2003), and is used in conjunction with the modified Spring Embedder as described by Fruchterman and Reingold (1991).

Usage of the multilevel scheme and modified spring embedder provides an opportunity to compare layouts against those generated by Walshaw (2003) and Hu (2005), who use the same methods.

5.2.1 Comparisons to state-of-the-art

The comparison to state of the art Octree algorithm by Hu (2006), and the n-body algorithm described by Walshaw (2003) is achieved in two parts, firstly a comparison of the results to those published in the publications, and then a comparison to implementations of the methods using Graph Drawing Framework here (Appendix 33910.22.1). The methods are:

- Multilevel scheme with n-body force calculation (as described by Walshaw, 2003)
- Multilevel scheme with Octree approximation (as described by Barnes and Hut, 1986, and Hu, 2005)
- Multilevel scheme with Multilevel Global Forces (Crawford, 2013)

Details of implementation are provided in Appendix 10.22 . Implementations here do not include additional optimisation methods for reducing running time or generating improved

matching/approximation, they include only Force Directed Placement, Multilevel Scheme and the described Approximation methods.

It should be noted that use of the modified spring embedder (Fruchterman and Reingold, 1991) limits vertex movement through a cooling schedule until some tolerance is met. As the algorithms both use the same tolerance, they finish in the same number of iterations, thus it is not accurate to determine if one method of approximation causes faster convergence than another.

5.2.1.1 Numerical Results

5.2.1.1.1 Comparison of Implemented Multilevel Global Force and Octree

Comparison of running time shows a significant decrease (on average 41% reduction) in running time when using Multilevel Global Force (*MGF*) over use of Octree (*OT*), shown in Table 5.3. Further analysis of running time (see Appendix 10.1) identifies that shared processes (such as coarsening) run in equivalent time, with differences only in calculation of repulsive forces and approximation utilities (such as generation of the approximation structure).

The methods are represented in the results using the following abbreviations:

- *MGF* – Multilevel Spring Embedder (Fruchterman-Reingold) with Multilevel Global Force;
- *OT* – Multilevel Spring Embedder (Fruchterman-Reingold) with Octree approximation.

Graph	Running Time (ms)			<i>MGF</i> Update	Octree Generation
	<i>MGF</i>	<i>OT</i>	Reduction		
4elt	4455.00	7434.64	40.08%	0.0	58.42
3025	628.00	908.82	30.90%	0.0	12.86
add32	1302.80	2410.86	45.96%	6.2	27.86
data	687.80	953.12	27.84%	0.0	10.88
dime20	97381.60	196802.60	50.52%	62.2	1394.36
finan512	36486.00	60708.08	39.90%	58.8	407.72
mesh100	43958.00	77373.42	43.19%	31.2	571.46
sierpinski10	31789.60	63246.06	49.74%	19.2	539.28
			41.01%		

Table 5.3. Comparison of running time for Multilevel Spring Embedder with Multilevel Global Force (*MGF*) or Octree (*OT*) approximation. The results show a significant 41% decrease in running time from usage of *MGF*, in addition to reduced maintenance running time between *MGF* Update and Octree Generation. Running time is shown in milliseconds.

Although a clear improvement in running time, the difference in quality of layouts is less obvious. Table 5.4 compares the average number of edge crossings exhibited in layouts. The results show that in general, there is a noticeable increase (13%) in edge crossings for graphs generated using *MGF*, however the results indicate noticeable fluctuations between graphs, with add32 exhibiting 340% increase in edge crossings for use with *MGF* but a 97% decrease for 3025.

Such differences come from the structure of the multilevel scheme generated (see Section 3.6.2), whereby graph substructures and connectivity can impact the matchings found and causing warping in the layouts. For example, add32 is known to coarsen to a sparse star type graph, resulting in many levels being generated.

The range in edge crossings, also shown in Table 5.4 gives an indication of the regularity of layout quality. Multilevel Global Force provides layouts with little change in edge crossings, suggesting similar layouts being generated for a graph. In contrast, layouts generated using the Octree provides have a larger range, suggested dissimilar layouts (caused by differences in the generated Octree structure for each repeat).

Graph	Edge Crossings			Edge Crossing range	
	<i>MGF</i>	<i>OT</i>	Reduction	<i>MGF</i>	<i>OT</i>
4elt	39035.00	29306.40	-33.20%	290	14161
3025	1.60	410.40	99.61%	32	2052
add32	39511.80	11558.40	-241.84%	78	3617
data	43771.40	35433.40	-23.53%	31	2933
dime20	68879.60	115893.80	41.57%	4625	35211
finan512	5707612.60	5295335.00	-7.79%	1094	67566
mesh100	988967.20	953497.60	-3.72%	4453	15250
sierpinski10	14364.80	37647.20	62.84%	327	3498
			-13.51		

Table 5.4. Comparison of edge crossings and range in edge length for layouts generated by Multilevel Spring Embedder with Multilevel Global Force (*MGF*) and Octree (*OT*) approximation. The results show that layouts generated using Multilevel Global Force (*MGF*) have an increase in edge crossings of 13.51%.

Although showing an increased number of edge crossings on average, the results don't include information regarding the regularity of drawings, which is the regularity of concurrent

drawings having the same graph are given the same layout. This is investigated by randomizing the number of vertices but preserving the connectivity, then comparing the range in edge crossings, further explained in Appendix 10.19 .

5.2.1.1.2 Comparison to Hu (2005)

A comparison to some of the algorithms discussed by Hu (2005) is provided below. The methods compared are named as follows:

- *MLFDP* – Multilevel Spring Embedder described by Walshaw (2003);
- *MSE(r)* – Multilevel Spring-Electrical Model using Hybrid coarsening Scheme, Octree approximation, and cutoff radius for repulsive forces provided as r , described by Hu (2005). The method uses the Fruchterman-Reingold spring embedder;
- *MS(r)* – Multilevel Spring-Electrical Model with General Repulsive Force, as described for *MSE(r)* but with use of the Kamada-Kawai spring embedder method;
- *SE* – Spring-Electrical Model with Octree approximation.

Of the methods, *MSE(∞)* bears closest resemblance to the Octree implementation here, however Hu incorporates other optimisation methods into the algorithm which are not used here. Running time recorded here is converted into seconds to match the times described by Hu (2005).

The running times in Table 5.5 show that for many of the graphs, the results provided by the Octree here are similar to those provided by Hu for *MSE(∞)*, somewhat validating the implementation of the Octree. Multilevel Global Force provides least running time of all methods. Octree implementation provides reduced running time over some Hu methods due to differences in experimental environment.

Graph	Running Time (s)						
	<i>MLFDP</i>	<i>MSE(2)</i>	<i>MSE(∞)</i>	<i>MS(4)</i>	<i>SE</i>	<i>MGF</i>	<i>OT</i>
4elt	24.3	9.4	11.7	102.9	9.2	4.5	7.4
3025	-	-	-	-	-	0.6	0.9
add32	-	3.1	3.3	44.4	2.6	1.3	2.4
data	-	1.1	1.2	17.8	1.3	0.7	1.0
dime20	264.3	195.5	290.6	1984.6	277.7	97.4	196.8
finan512	363.8	56.6	59.8	3714.9	277.7	36.5	60.7
mesh100	-	91.6	109.4	5807.8	89.5	44.0	77.4
sierpinski10	-	44.1	65.1	146.8	75.6	31.8	63.2

Table 5.5. Comparison of running times for the algorithms tested here (Multilevel Global Force (*MGF*) and Octree (OT)) to those published by Hu (2005). Running times are provided in seconds due to the formatting provided by Hu.

5.2.1.2 Subjective Analysis

Numerical results alone only suggest the readability of layouts, subjective analysis is used to determine if the results above match the authors' subjective view of the layouts.

Figure 5.1 provides typical examples for layouts generated for the graph 4elt using Multilevel Force Directed Placement, showing differences between layouts generated with Multilevel Global Force Approximation (left), Octree Approximation (center) and no approximation (right). The most noticeable difference between layouts is Peripheral Effect, a result of differences in the repulsive forces. It is noted that the Octree provides layouts closer resembling No Approximation, however the readability of both is clouded by folds in the layout. In contrast, the layout generated using *MGF* is elongated, and although similar, is distinct from the other layouts with the overlapping branches compressed.

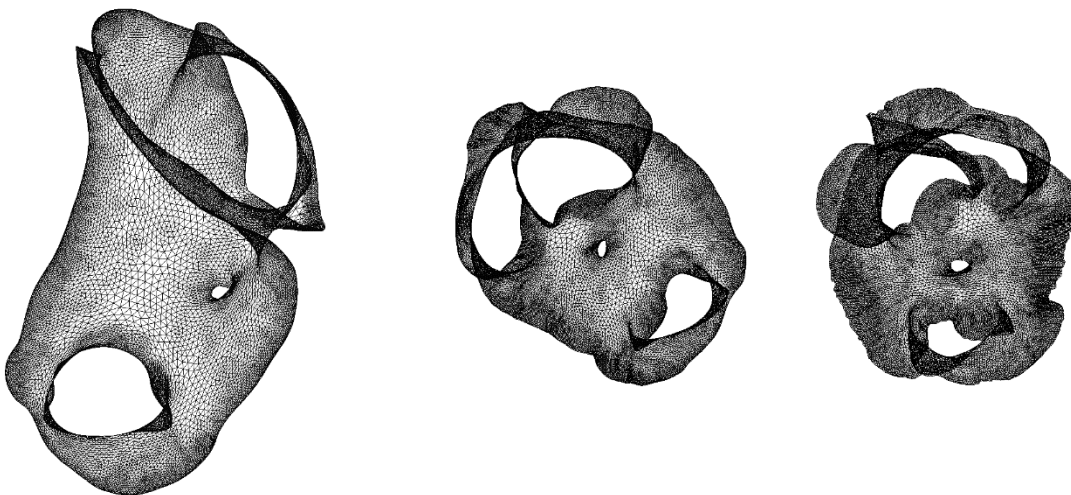


Figure 5.1. Layout generated for 4elt using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left), Octree approximation (centre) and n-body force calculation without approximation (right).

Figure 5.2 shows a similar elongation of the layouts for the graph 3025, showing the extent of the warping in layouts generated using *MGF* (left). In contrast, layouts from the Octree show little to no warping as a result of more equal forces, however, if the approximation structure is not centered, or there is an uneven distribution of vertices in the drawing area, layouts may become warped (right).

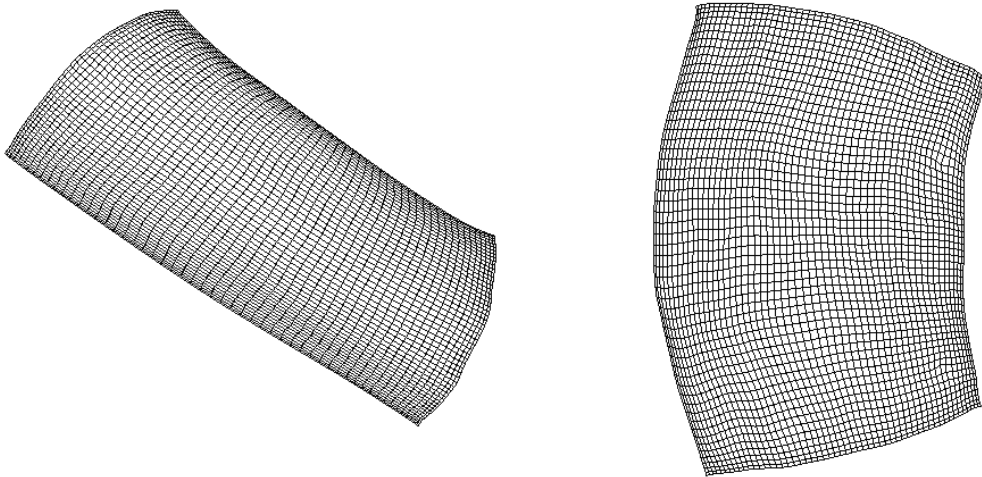


Figure 5.2. Layout generated for 3025 using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right)

The warping in layouts is the result of different repulsive forces, which stretch the layouts shown in Figure 5.1 and Figure 5.2. For larger graphs, the number of levels in the multilevel approximation structure is increased, resulting in higher numbers of repulsive forces. Figure 5.3 compares the layout for Sierpinski10, showing that warping on a global scale is reduced, however, the additional repulsive forces cause the Peripheral Effect to be much more noticeable. In contrast, the Octree layout is much more rigid and flat. Although rough, the Octree layout (right) provides a good representation of subgraph structures which are shown in a triangular form, whereas the *MGF* layout represents them more as circles.

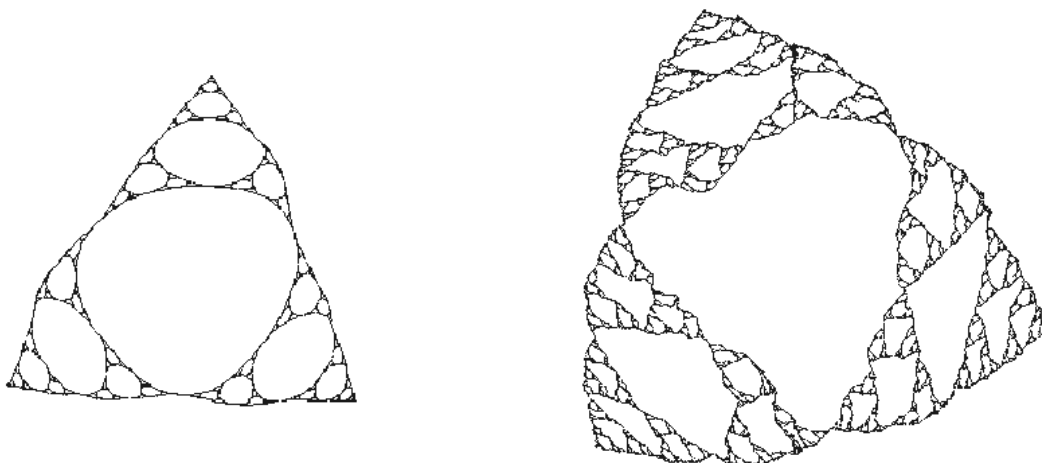


Figure 5.3. Layout generated for sierpinski10 using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right)

Such disregard for local structures may be considered a weakness of the *MGF* approach, which incorporates global layout into their placement, however it can also be regarded as a strength. The graph *dime20* is one of the larger test graphs and is compared in Figure 5.4. The Peripheral Effect is much more noticeable for *MGF* (left), however the compression on the edges of the global layout make it easier to identify a global structure with shape more readable. In contrast, the layout generated by Octree provides a more uniform edge length, providing a layout which looks rougher but allowing a user to see the layout of smaller structures in the graph.

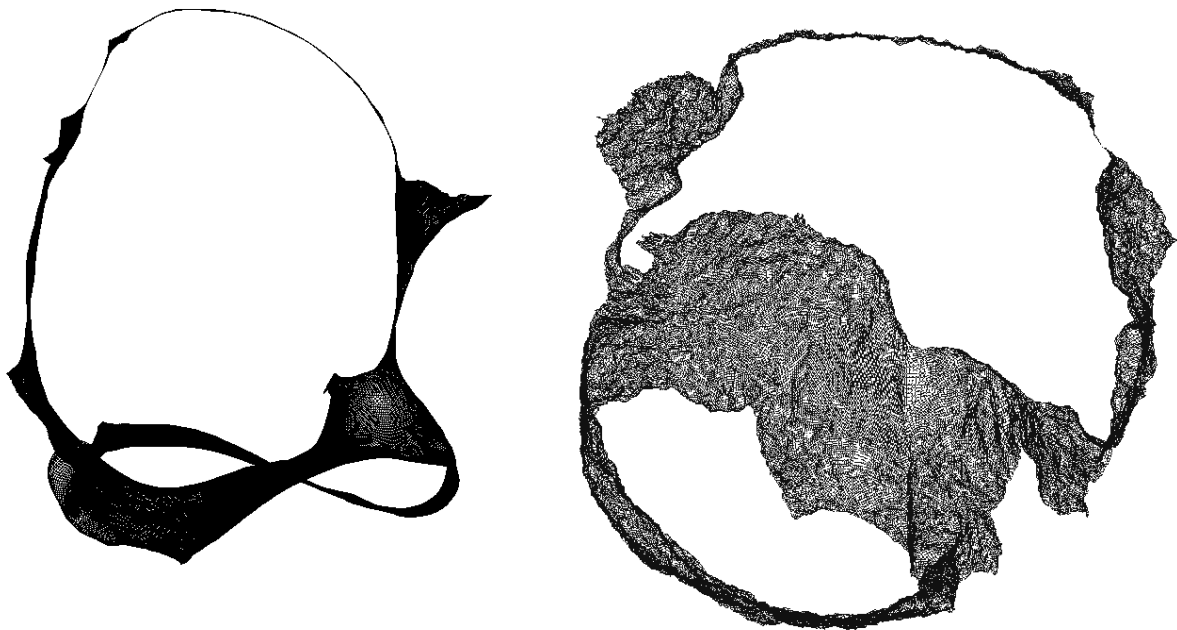


Figure 5.4. Layout generated for dime20 using Fruchterman-Reingold Spring Embedder using *MGF* approximation (left) and Octree approximation (right). Note the difference of the Peripheral Effect, whereby edges for the Octree are more uniform leading to a more expansive and “fluffy” layout, whereas the layout achieved with *MGF* shows noticeably more compression.

5.2.2 Optimising the Multilevel Structure

In an attempt to improve the approximation of graph structure, amendments to the matching process are investigated to identify if any optimisation can be achieved, aiming to improve the representation of the data within the drawings. The primary approaches used depend on modifying the matching mechanism used to generate the multilevel scheme follow.

5.2.2.1 Multimatching

Increasing the number of vertices which can be matched together (from edge contraction of 2) up to 8 vertices is thought to show any correlation with average or modal vertex degree, however the results show little connection. On average, and across all graphs, the quality of layouts generated decreased over time (edge crossings increased by 27-50% for values between 3 and 8). In contrast, there was a minor decrease in running time for larger matching values (an average reduction of 5% for Multimatching values of 4 and above). A full description of the experimentation and results is provided in Appendix 10.20 . Subjective analysis shows that improvement is largely dependent on the graph with no observed correlation to structure or connectivity (see Appendix 10.20.2).

5.2.2.2 Brute Matching

Brute Matching is an extreme version of Multimatching whereby vertices are matched with all adjacent unmatched vertices in order to generate an extreme approximation of structure and reduced multilevel scheme. A full description of the experimentation and results is provided in Appendix 10.8 . The results show negligible change in layout quality on average (the larger test graphs show a reduction of up to 16%) but a huge increase in running time (over 500% for some larger graphs) as a result of the changing multilevel structure (Section 3.6.2).

5.2.2.3 Pattern Processing

Pattern Processing is a more specific technique for identifying vertex patterns within a graph which can be provided with a calculated known layout, removing the need for expensive force directed placement. Two methods are investigated, pattern coarsening within the multilevel scheme, and primitive graph identification in coarser graphs as a means of stopping the coarsening scheme (similar to the coarsening tolerance).

The first method uses a Leaf Placement Scheme which gives leaf vertices layout pointing outward from the centre of mass (albeit skewed by local masses). Application of the LPS is tested in different states:

- Only local level with calculated placement, resulting in a 20% increase and 31% decrease in running time, and 1% decrease and 54% increase in edge crossings, for leafless and leafy graphs respectively;
- Only local level but refined using *FDP*, resulting in 60% and 15% increase in running time, and a 9% and 13% decrease in edge crossings, for leafless and leafy graphs respectively;
- Throughout the Multilevel scheme with calculated placement, resulting in 25% increase and 31% decrease in running time, and a 10% and 86% increase in edge crossings, for leafless and leafy graphs respectively;
- Throughout the multilevel scheme refined using *FDP*, with 74% and 6% increase in running time, and a 16% and 8% decrease in edge crossings for leafless and leafy graphs respectively

The second method identifies primitive graph shapes, where the connectivity within a coarsened graph matches one of 3 states (a ring of vertices, a chain of vertices, or a star of vertices), providing a known layout (expected to translate to accurate global layout) and preventing further coarsening (reducing the levels in the multilevel scheme and resulting *FDP*). The results show an average increase of 0.4% for running time, and a 35.7% increase in edge crossings.

Full description of experimentation and results are provided in Appendix 10.10 .

6 Dynamic Graph Drawing: Experimentation and Results

This chapter describes the experimentation and results specific for Dynamic Graph Drawing, aiming to investigate if the efficiency saving and high quality layouts which have been achieved in Static Graph Drawing, can be applied to Dynamic Graph Drawing (as described in Section N.N of Chapter 3). Four proposed algorithms are evaluated:

Dynamic Modified Spring Embedder (<i>DSE</i>)	Dynamic variant of the <i>MSE</i> . Brings advantages of the <i>MSE</i> (notably strong forces to overcome minima) to dynamic graphs.
Multilevel Dynamic Modified Spring Embedder (<i>ML DSE</i>)	Investigates use of <i>ML</i> with <i>DSE</i> to improve untangling of vertices and overcoming minima in larger graphs.
Dynamic Spring Embedder with Multilevel Global Force (<i>MGF DSE</i>)	Investigates uses of <i>MGF</i> to <i>DSE</i> and <i>ML DSE</i> to reduce their complexity.
Dynamic Spring Embedder with Dynamic Matching (<i>DSE DM</i>)	Introduces a method for dynamic matching to perform operations to dynamic graphs whilst being drawn by <i>DSE</i> , <i>ML DSE</i> and <i>MGF DSE</i> . Four methods for updating are investigated which will update the <i>ML</i> scheme in varying depths; Low/Medium/High level Updates Full Re-matching

The Dynamic Spring Embedder is evaluated in regard to both **layout generation** and **layout generation**. Multilevel Dynamic Spring Embedder and Dynamic Spring Embedder with Multilevel Global Force are evaluated in regard to **layout generation** alone, with their application to **layout adjustment** evaluated in Dynamic Matching in combination with the dynamic update methods.

6.1 Test Data – Dynamic Graphs

There are few mainstream large dynamic graphs existing in the literature, therefore dynamic drawing algorithms are tested on graphs generated by heuristics (described in Appendix 10.21). Three heuristics are used to provide graphs with repeated sub-structures, each with differing degree and in differing sizes (small, medium and large), Table 6.1.

Graph	Vertices	Edges	Average Degree	Description
sparse-small	255	508	2.0	Binary trees with differing depth, generated automatically by computer.
sparse-medium	1023	2044	2.0	
sparse-large	8191	16380	2.0	
regular-small	256	960	3.8	Regular $N \times N$ vertex grids, generated automatically by computer.
regular-medium	1024	3968	3.9	
regular-large	8100	32040	4.0	
<i>dense-small</i>	364	13998	38.5	Densely connected graphs with high number of edges between vertices, generated automatically by computer.
<i>dense-medium</i>	1093	129620	110.4	

Table 6.1. A collection of static graphs to be used as the base dataset for dynamic graphs for experimenting with dynamic drawing algorithms. The number of vertices and edges, the average degree, and a description regarding the structure of the graphs are provided.

In addition to these initial static graphs, a list of amendments are provided for each. Details regarding the types of amendments are provided below with details regarding storage and application provided in Implementation (10.22).

Two types of gradual layout development is tested here:

Layout Generation – the generation of layout from initial random positions, with changes measured over time for some predefined time period. This does not involve graph modifications.

Layout Adjustment – the application of graph operations to an already defined layout, applied periodically over some predefined time period.

6.1.1 Types of Operations

Graph modification operations can include any imaginable change to a graph structure, in any combination and in any frequency. In literature, modifications typically expand or reduce the

size of a graph, adding or removing vertices and edges over some time period. The operations defined here follow these two main types and include a mix of the two:

- *growth* – increasing the number of vertices and edges
- *shrink* – decreasing the number of vertices and edges
- *maintain* - Little difference in the number of vertices and edges

Information regarding the generation of operations and the rules in place for emulating “realistic application of operations” are described in Implementation (Appendix 10.22).

More advanced operations such as the removal or addition of subgraphs, or merging graphs, can be applied using collections of these base modifications, but are not investigated here due to the range of possibilities.

6.1.2 Frequency of Operations

The frequency of operations describes the rate at which amendments to a graph are made, more specifically, the number of operations applied per number of frames. The number of operations is generated as some percentage of the initial graph size (vertices and edges), and are applied over a period of 200 frames, thus frequency is dependent on the size of the graph.

Although not explicitly tested here, the frequency of operations impacts layouts, and so the test data is chosen to avoid such impacts. Figure 6.1 gives an example of a negative impact on layout as a result of edges being added to a graph faster than Force Directed Placement can give them layout.

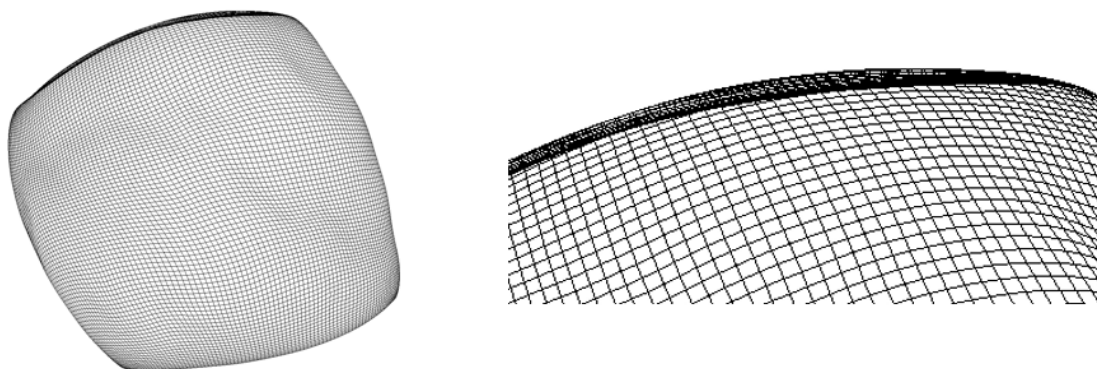


Figure 6.1. Layout provided to the graph regular-large using *FRD MGF* with single level multilevel update. The layout shows that added vertices and edges are not provided with low energy positions before the next set of operations occur, causing a build-up represented as a dense area on the graph (right).

6.2 Dynamic Spring Embedder

As performed in **Static Graph Drawing**, some experimentation is undertaken to determine the parameter set for the algorithm. This identifies the general parameters which offer best performance between tests.

6.2.1 Configuration

Details of the configuration for the Dynamic Spring Embedder are provided in Appendix 10.11

.

In summary, experimentation uses default values of 0.5 and 0.9 for the repulsive and spring force cooling schedules, which provide highest quality results in general. See Appendix 10.11 for the data.

6.2.2 Comparison of Spring Embedder Algorithms

To determine if the proposed spring embedder adaption is fit for purpose, layouts are generated for a collection of small graphs and compared to implementations of other Spring Embedder models. The Spring Embedder models used for comparison are:

- **SE** - Spring Embedder Heuristic described by Eades (1984)
- **FRI** - Modified Spring Embedder described by Fruchterman and Reingold (1991)
- **FR2** – Modified Spring Embedder described by Fruchterman and Reingold (1991), with reduced cooling factor to allow for 300 iterations
- **FRD** - Dynamic Spring Embedder using default parameters
- *(Additional adaptations of the Dynamic Spring Embedder using alternate cooling schedules are included in the extended results in Appendix 10.25 but do not show any benefit over the algorithms above).*

Initial layouts are provided through random placement of vertices within the drawing area. Experiments are run for 300 frames (300 iterations of *FDP*) and quality is assessed every 10 frames. Tests are repeated 10 times for each graph.

6.2.2.1 Analysis of Numerical Results

Analysis of the final layouts shows that the Dynamic Spring Embedder method generates drawings with fewer edge crossings on average, applicable to most test graphs except the dense graphs which exhibit less edge crossings when drawn with the Eades Spring Embedder. The

Fruchterman and Reingold Spring Embedder performs worst in regards to edge crossings, exhibiting highest figures for all graphs.

The cause for the poor behaviour is the cooling schedule, which forces the Force Directed Placement to end early (before 50 iterations), while the other two continue to gradually improve layout.

	edge crossings		
	dense	regular	sparse
<i>FR</i>	4816331	15929.4	3388.2
<i>SE</i>	3892754	244.4	49.1
<i>FRD</i>	4613519	45.7	3.4

Table 6.2 Number of edge crossings exhibited in layout after 300 iterations of *FDP*, comparing the Modified Spring Embedder of Fruchterman and Reingold (*FR*), the Eades Spring Embedder (*SE*), and the proposed Dynamic Spring Embedder (*FRD*)

By reducing the cooling value used in the Fruchterman and Reingold algorithm, it can be run for an extended number of frames before movement stops, shown in Figure 6.2 comparing the method to the algorithms above.

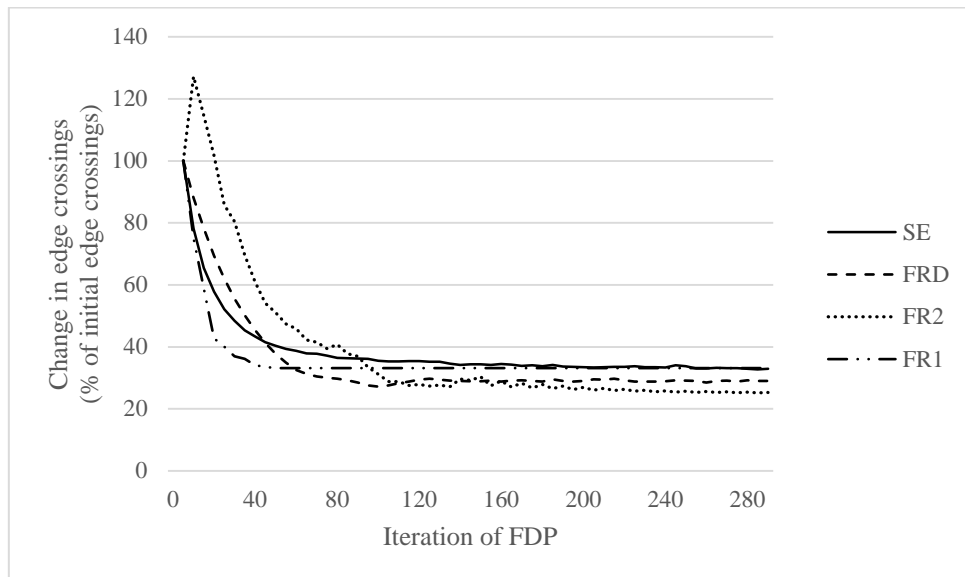


Figure 6.2. Change in edge crossings exhibited in layouts provided to the collection of test graphs by Eades Spring Embedder (*SE*), Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder Cooling Schedule (*FRDC*) over 300 iterations of *FDP* application

The change in edge crossings is similar for each of the algorithms, with many reaching a minimum between 25% and 35% of the initial edge crossings of the randomly positioned vertices.

FR1 shows the steepest drop in edge crossings initially due to the cooling schedule allowing for high energy movements which are quickly cooled to rest vertices in place. *FR2* shows the weakness when changing the cooling schedule, which allows for continuous high energy movement (resulting in a slower drop in edge crossings).

The highest quality final layouts come from *FR2* with the modified cooling time, followed by *FRD*. Although providing best layouts on average, the *FR1* still retains an inflexible cooling schedule which forces reduced movement over time.

Movement is additionally recorded, however due to differences between algorithms, the precise values are not truly comparable. The chart in Figure 6.3 compares *SE*, *FRD* and *FR1*, showing the relative change in movement of the graphs.

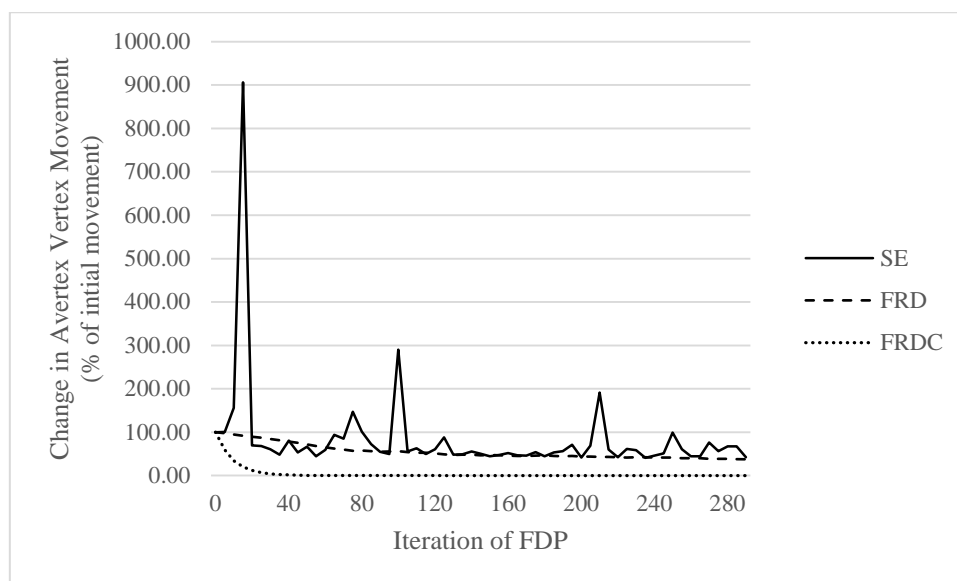


Figure 6.3. Change in average vertex movement between frames, exhibited in layouts provided to the collection of test graphs by Eades Spring Embedder (*SE*), Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder Cooling Schedule (*FRDC*) over 300 iterations of FDP application

The *SE* algorithm has noticeable peaks in its movement, the result of spikes in movement as local minima is overcome, and are more noticeable due to the relatively small movement occurring between spikes. Movement in *FR1* is higher and allows for larger movements initially, quickly decreasing to 0. In contrast, the movement in *FRD* is more continuous, neither

cooling nor spiking. The actual values show that the *FRD* and *FRI* algorithms have much higher amounts of movement in contrast to *SE*, but cannot be accurately compared due to the different parameter sets used by the algorithms.

6.2.2.2 Subjective Analysis

The analysis above shows layouts drawn using the Eades Spring Embedder (*SE*) are typically drawn with higher edge crossings than the Dynamic Spring Embedder (*FRD*). Figure 6.4 shows that for regular-small, the layout generated using *SE* exhibits tangles commonly seen when unable to overcome minima. In contrast, *FRD* provides layouts with minimal edge crossings, but a more prominent peripheral effect (which can be reduced through modifying the algorithms configuration, shown in Figure 5.3). Similar conclusions are made for layouts of sparse-small in Figure 6.6.

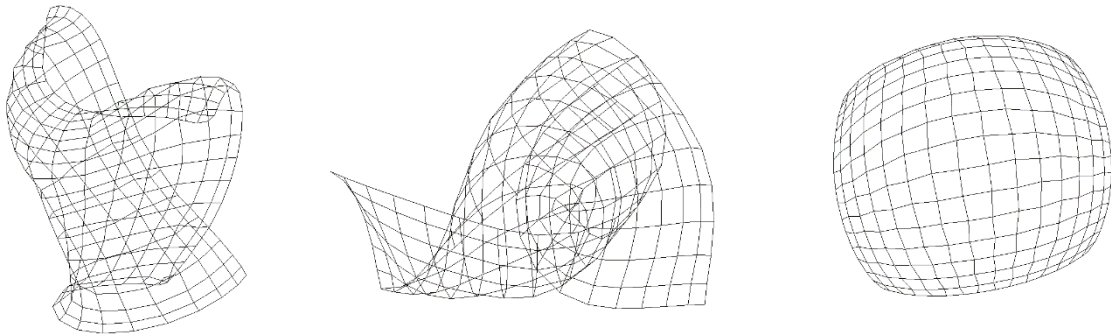


Figure 6.4. Layouts provided to the graph regular-small provided by the Eades Spring Embedder (left), Fruchterman-Reingold Spring Embedder (middle) and Dynamic Spring Embedder (right), after 300 iterations of FDP application

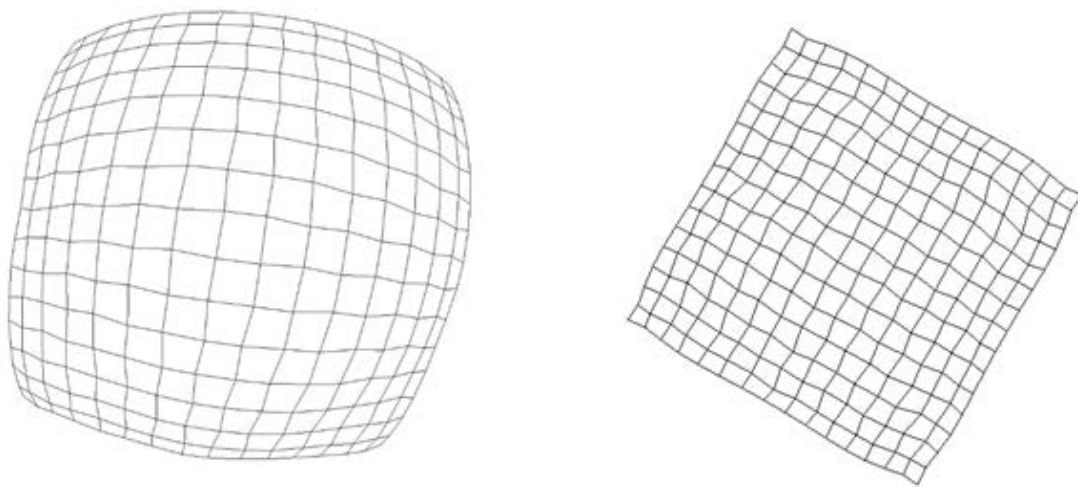


Figure 6.5 Reducing the Peripheral Effect of a layout by modifying the configuration of the algorithm

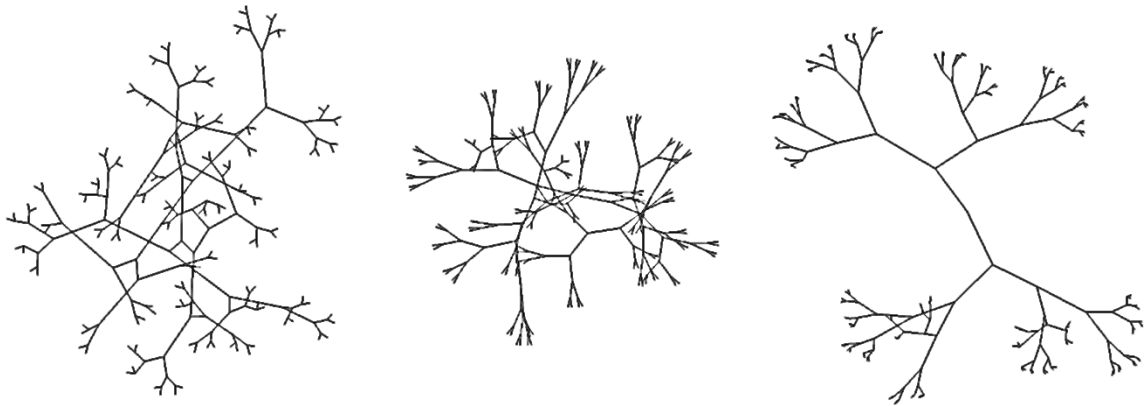


Figure 6.6. Layouts provided to the graph *sparse-small* provided by the Eades Spring Embedder (left), Fruchterman-Reingold Spring Embedder (middle) and Dynamic Spring Embedder (right), after 300 iterations of *FDP* application

The layout provided to leaf vertices in Figure 6.6 shows that for the Fruchterman and Reingold Spring Embedder (*FR*) and *FRD*, vertices are orientated pointing outwards from the centre of mass, whereas for *SE* the leaf vertices are orientated pointed outward from their anchoring vertex, suggesting stronger local layout.

Layouts for the graph *dense-small* are shown in Figure 6.7 with the layout generated by the Eades Spring Embedder showing the spikes in the vertex movement measured above. It is noted by Eades that some graphs are not drawn well, it is expected that there are cases when vertices are so close to one another, they cause an exponential rise in vertex movement causing such large spikes.

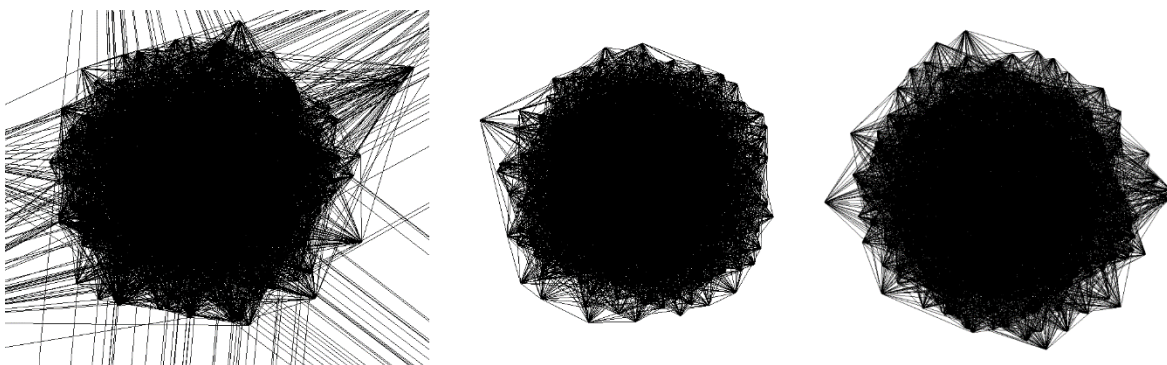


Figure 6.7. Layouts provided to the graph *dense-small* provided by the Eades Spring Embedder (left), Fruchterman-Reingold Spring Embedder (middle) and Dynamic Spring Embedder (right), after 300 iterations of *FDP* application

Layouts generated by *FRD* and *FR* are similar, however during visualisation, the layout of *FRD* appears to move continually as vertices attempt to find positions which reduce the graph energy.

Development of layout for regular-small is provided as an example in Figure 6.8 (layouts for other test graphs provided in Appendix 10.24). The layout is provided as observed during frames from the 10th, 50th, 100th, 200th and 300th iterations of *FDP*, showing a sharp decrease in edge crossings initially, slowing to a more gradual change in edge crossings and movement (as represented in Figure 6.8).

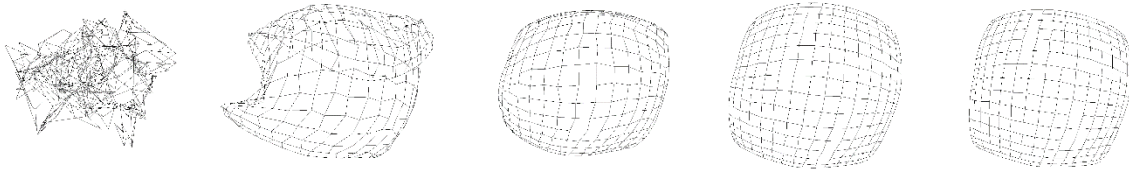


Figure 6.8. Emergence of layout for the graph regular-small using the proposed Dynamic Spring Embedder. From left to right, layout after (A) 10 iterations, (B) 50 iterations, (C) 100 iterations, (D) 200 iterations, and (E) 300 iterations.

During visualisation the initial change in edge crossings is difficult to follow, however after the 50th frame changes are much steadier. Due to the drastic change in layout, it is expected layout generation can be used to emulate extreme layout adjustment.

6.2.3 Layout Adjustment

Is the proposed adaptation to the spring embedder suitable for adjusting and retaining readability of layouts for dynamic graphs?

For the experimentation, each of the test graphs is provided a layout exhibiting minimum edge crossings generated prior to the tests. Experiments are run for 300 iterations of *FDP* following the procedure described in Section 6.2 .2. Analysis of the metrics for graph stability is used to determine the extent of changes to layout, confirmed by subjective analysis.

Analysis of Dynamic Spring Embedder is compared to the Eades Spring Embedder as both continually develop layout (unlike *FR*). Collected results are generalised under the three classification of amendments: *shrink*, *growth* and *maintain* (described in Section 6.1). Analysis of running time is not included as there is no optimization of either algorithms.

6.2.3.1 Analysis of Numerical Results

For *shrink* operations, there is a clear difference between the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*), shown in Figure 6.9 the number of edge crossings between frames frequently spikes when using *FRD* whereas the *SE* method shows a steadier decrease. The spikes suggest a large reaction to changes, as the layout quickly tries to find a new minimum energy. It is expected that the reaction is due to the relatively high repulsive forces when compared to the Spring Embedder, quickly separating connected parts of the graph.

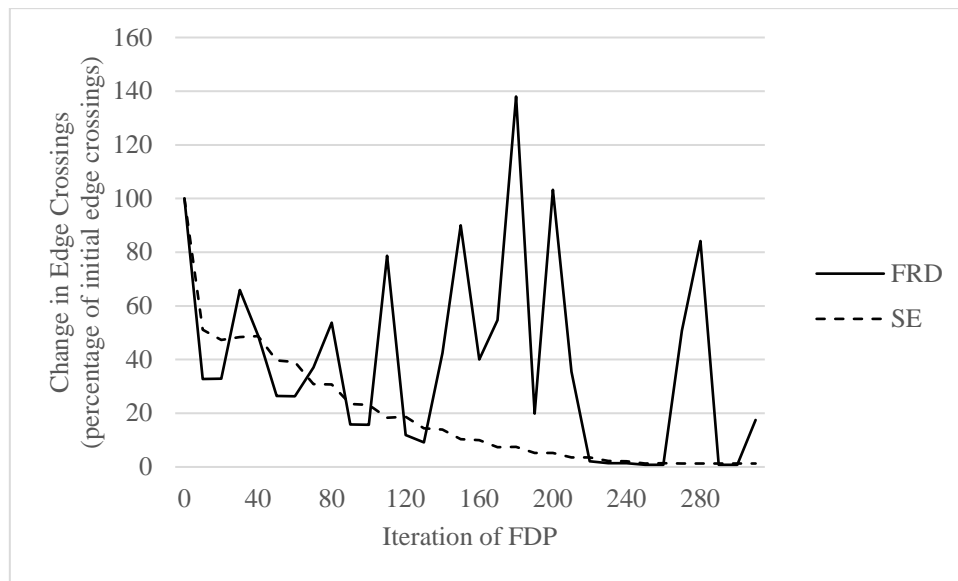


Figure 6.9. Change in edge crossings exhibited during layout generation for all test graphs during application of *shrink* operations, comparing the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*)

The chart suggests readability is hard to follow due to large increases in edge crossings, however the movement shown in the layout is not as active exhibiting little change to the relative movement of vertices within the graph (see Figure 6.10). In contrast, the relative movement of the Spring Embedder is high, due to the large movements in reaction to the shrink operations.

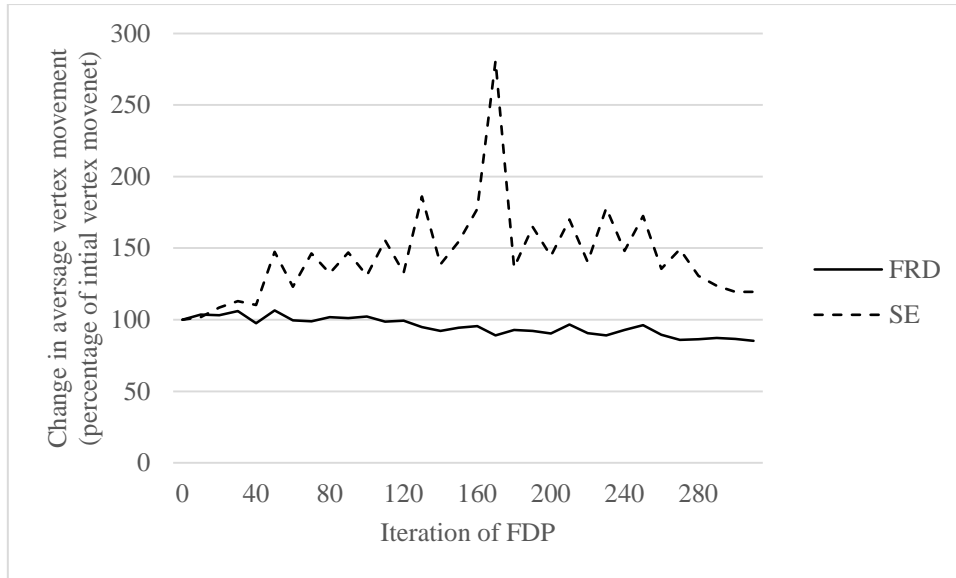


Figure 6.10. Change in average vertex movement exhibited during layout generation for all test graphs during application of *shrink* operations, comparing the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*)

For the *growth* amendments, both algorithms show increase in edge crossings as new vertices and new edges are introduced and overlap the existing layout. Such edge crossings appear as spikes in Figure 6.11 which remain high for the *SE* initially. During the cool down period, both algorithms are able to reach a similar minimum. *FRD* layouts reliably reach minimal edge crossings after changes have been applied, suggesting it is better able to overcome minima, but results in greater movement in the layout.

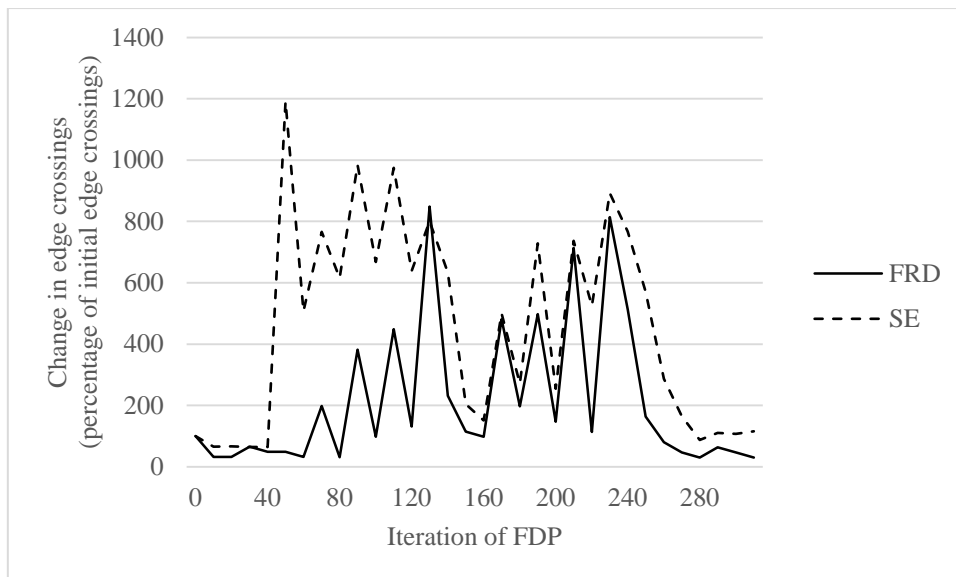


Figure 6.11. Change in edge crossings exhibited during layout generation for all test graphs during application of *growth* operations, comparing the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*)

Similar to the analysis of the shrink operations above, the chart in Figure 6.12 shows a continuous movement of vertices for *FRD* and large amounts of relative movement for *SE*. The spikes in movement represent the introduction of operations and the correction of layout as described above.

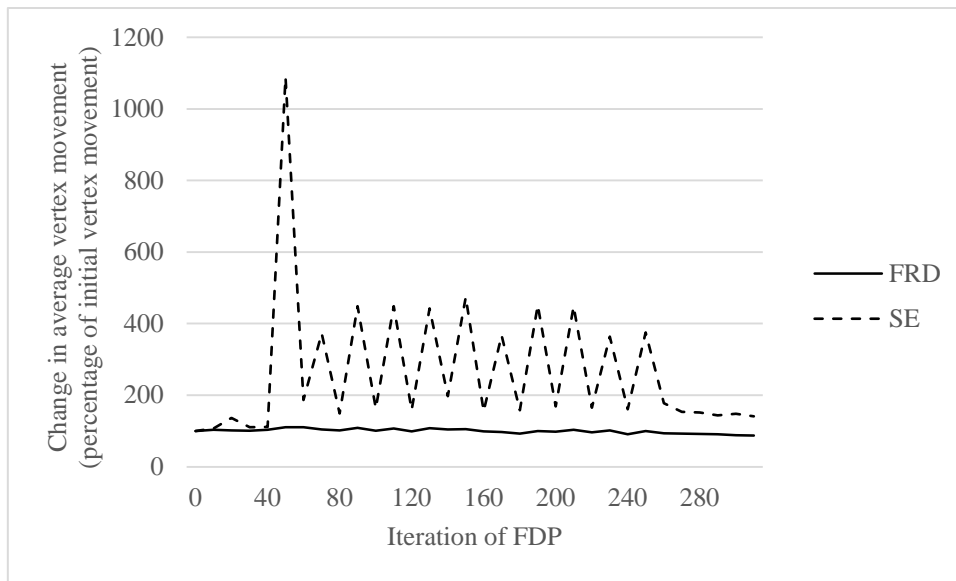


Figure 6.12. Change in average vertex movement exhibited during layout generation for all test graphs during application of *growth* operations, comparing the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*)

The *maintain* operations (which combine the *growth* and *shrink* operation sets) follows the behaviors of the results above. During the final phase however, leaf vertices are added randomly and the *SE* method shows a significant spike in edge crossings. The cause is similar to the effect shown in Figure 6.19 such that repulsive separation forces vertices apart by large distances, causing overlap with the rest of the layout.

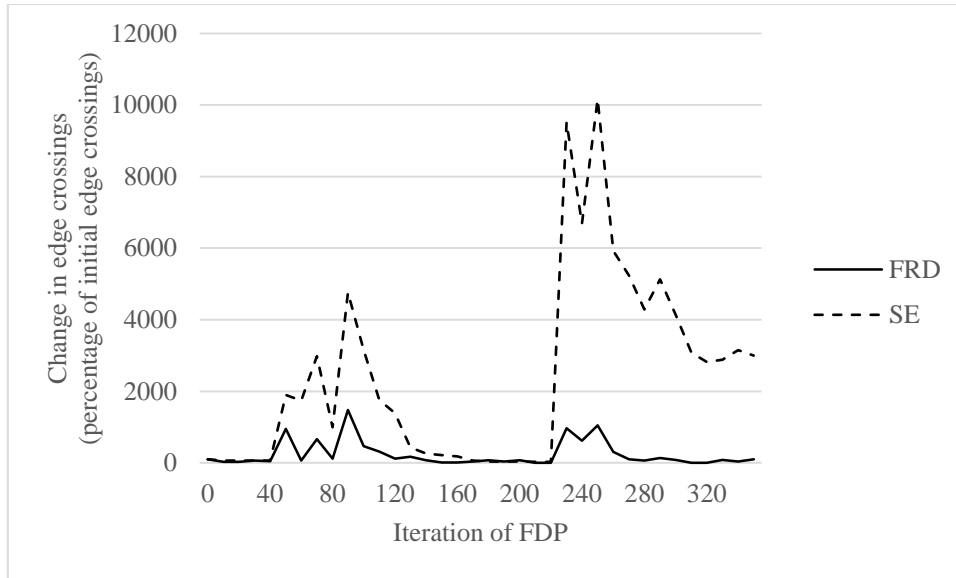


Figure 6.13. Change in edge crossings exhibited during layout generation for all test graphs during application of *maintain* operations set, comparing the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*)

Similar observations are shown for the average vertex movement for both algorithms, whereby large spikes occur as leaf vertices are introduced and repelled away from the body of the graph (see Figure 6.19 in Subjective Analysis below).

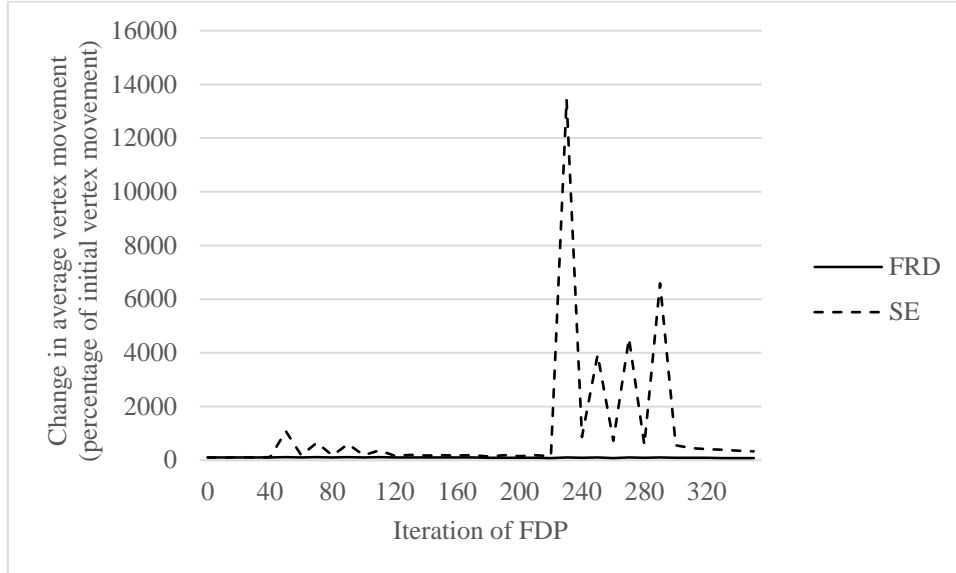


Figure 6.14. Change in average vertex movement exhibited during layout generation for all test graphs during application of the *maintain* operations set, comparing the Dynamic Spring Embedder (*FRD*) and the Eades Spring Embedder (*SE*)

6.2.3.2 Subjective Analysis

Subjective analysis is provided for the graph regular-small due to its recognisable and predictable $N \cdot N$ vertex grid layout. Further analysis for other graphs is provided in Appendix 10.24.2 .

Application of *shrink* operations is depicted in Figure 6.15, showing the change to a layout when using the Dynamic Spring Embedder (*FRD*). The removal of vertices and edges shrinks the graph from an initial layout (left) to form a smaller amended layout (right). The visualisation of changes is smooth and fluidly, however, singular changes occur quickly and are difficult to follow.

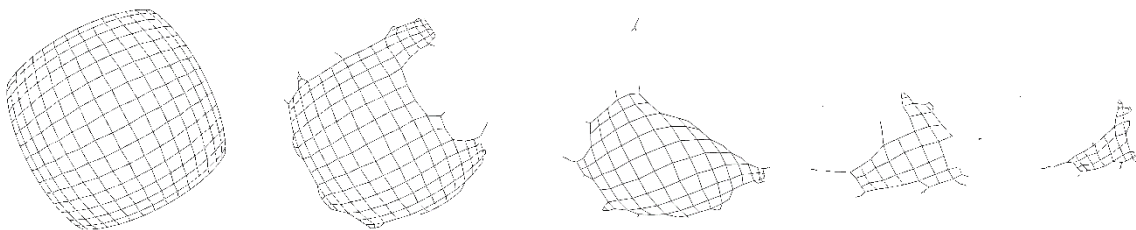


Figure 6.15. Application of *shrink* operations – removal of vertices and edges – to the graph regular-small. From left to right, an initial layout is provided to the base graph (A), after which vertices are removed in staggered phases (B, C, D) and a final layout generated after all operations have finished in (E). Note that the layouts in (D) and (E) are the same, however, shrinking and compression occurs as a result of reduced repulsive forces.

Figure 6.16 provides a comparison of the final layouts generated using Dynamic Spring Embedder (*FRD*, left) and Eades Spring Embedder (*SE*, right), following application of amendments. The layouts are similar with minor differences: the layout provided by *SE* looks more uniform and taut than the *FRD* layout.

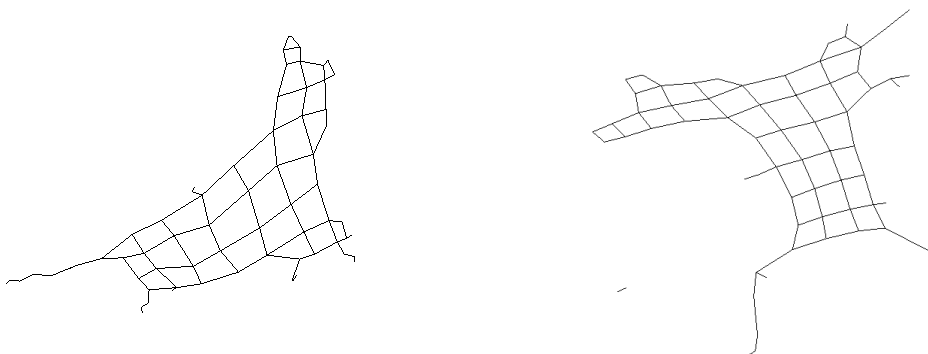


Figure 6.16. Layouts for regular-small after application of *shrink* operations, as drawn using Dynamic Spring Embedder (left) and Eades Spring Embedder (right)

Figure 6.17 illustrates the expansion of regular-small using the *growth* operations, showing preservation of the initial layout as vertices and edges are introduced. The grid structure is extended in one direction to form a rectangular structure, with observation of the process visualising a smooth transition as new vertices and edges are added, and little change to the positions of unaffected vertices.

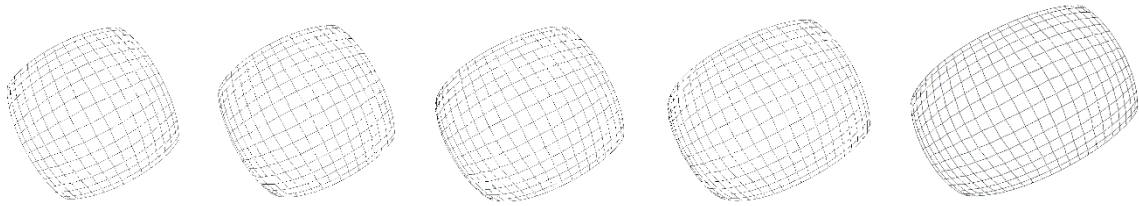


Figure 6.17. Application of *growth* operations – addition of vertices and edges – to the graph regular-small. From left to right, an initial layout is provided to the base graph (A), after which vertices are added in staggered phases (B, C, D, and E). Layout is achieved almost instantly as other vertices are being added, leading to a smooth visualization of the extension.

The final layouts for regular-small after *growth* operations are illustrated in Figure 6.18, showing that the layout provided by the Eades Spring Embedder (right) has a more uniform edge length, but appears distorted (“wavy”), whereas the Dynamic Spring Embedder generated layout (left) appears more symmetrical and rounded (due to the Peripheral Effect).

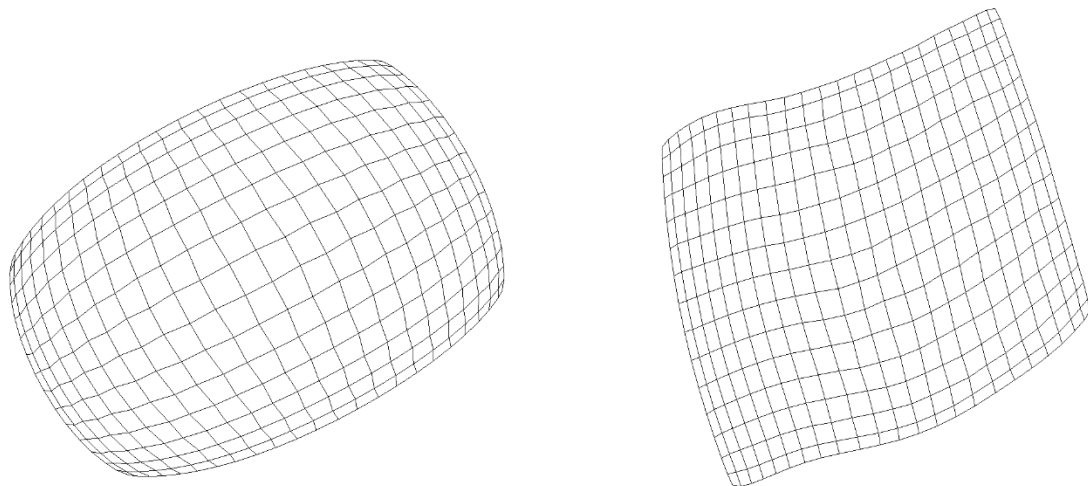


Figure 6.18. Layouts for regular-small after application of *growth* operations, as drawn using Dynamic Spring Embedder (left) and Eades Spring Embedder (right)

As noted in the quantitative analysis, the *maintain* operations show that the addition of leaf vertices can lead to large repulsive forces separating vertices when using the Eades Spring Embedder method. Figure 6.19 depicts such an event, whereby vertices are moved and edges are stretched, causing increased edge crossings and deformation of the layout as a result. The initial placement of these vertices is also expected to exacerbate the effect, as vertices are intentionally drawn close to their anchor.

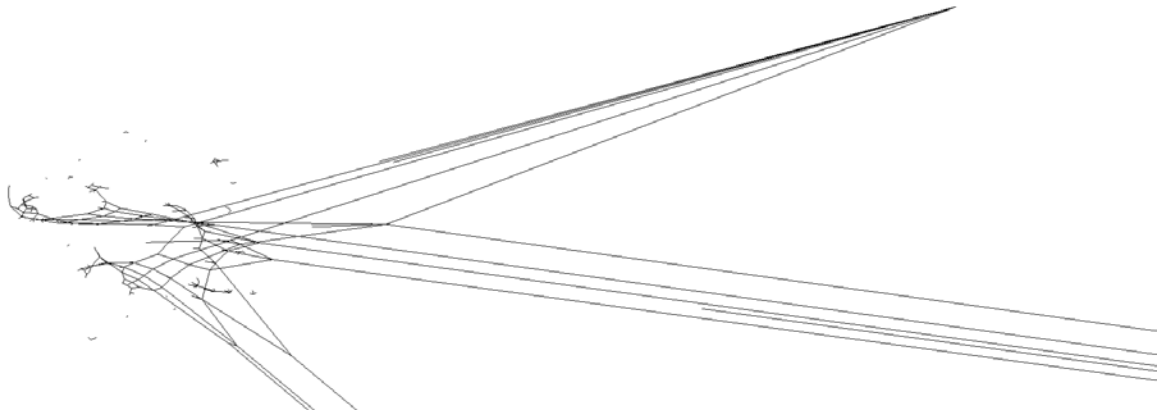


Figure 6.19. Layout generated for regular-small after application of the *maintain* amendments, whereby vertices become repelled so much they move large distances, resulting in the extended edge lengths and poor vertex placement seen in the image

6.3 Multilevel Dynamic Modified Spring Embedder

The expectation of the multilevel scheme is to reduce global untangling, resulting in fewer edge crossings (as exhibited in static graph drawing, Walshaw, 2003). Comparison is made between the Dynamic Spring Embedder (Section 3.7.4) and Multilevel Dynamic Spring Embedder (Section 3.7.5), sharing the same underlying Force Directed Placement.

Tests are performed using layout generation from an initial randomised layout. The algorithms are then run for 300 iterations and repeated 10 times for each test graph. The number of edge crossings and edge lengths are measured to compare the quality of layouts generated.

Layout adjustment is tested as part of the Dynamic Matching subsection (Section 6.5).

6.3.1 Analysis of Numerical Results

6.3.1.1 Final Layouts

Comparison of final layout quality is provided in Table 6.3.

Graph	Edge Crossings in Final Layout		Range in Edge Length	
	<i>FRD</i>	<i>FRDML</i>	<i>FRD</i>	<i>FRDML</i>
sparse-small	1.6	16.0	12.7860	12.7708
sparse-medium	22.4	79.8	25.1969	33.0241
sparse-large	1948.6	1443.4	48.7551	82.2162
regular-small	0.0	4.4	9.6504	11.5799
regular-medium	663.2	306.2	12.9208	23.6915
regular-large	31830.6	12710.4	15.7241	29.5661
<i>dense-small</i>	4134494.0	3742154.0	3.6427	9.9145
<i>dense-medium</i>	385933360.4	335259399.8	3.4626	9.1963

Table 6.3. Comparison of edge crossings and range in edge lengths for layouts to a selection of graphs, generated by *FRDML* and compared to *FRD* to identify if layout quality improves for larger graphs (as a result of global layout being better achieved).

The results show an average 42% decrease in edge crossings for the larger graphs, but a 410% increase in edge crossings for smaller graphs, suggesting the multilevel method is best applied for larger graphs only. Further to this, it suggests the multilevel layout conflicts with the local layout in these smaller layouts.

In addition to the change in edge crossings, the range in edge lengths increases as a result of using multilevel layout generation, suggesting the movement in coarser layouts causes the layout to expand more than normal.

6.3.1.2 Metrics for Graph Stability

The numerical analysis above suggests multilevel is best suited for larger graphs, but on average is not beneficial to graph drawing. The same applies when comparing the change in edge crossings, with Figure 6.20 showing edge crossing for both algorithms quickly drop in the first 50 frames to reach a similar minimum.

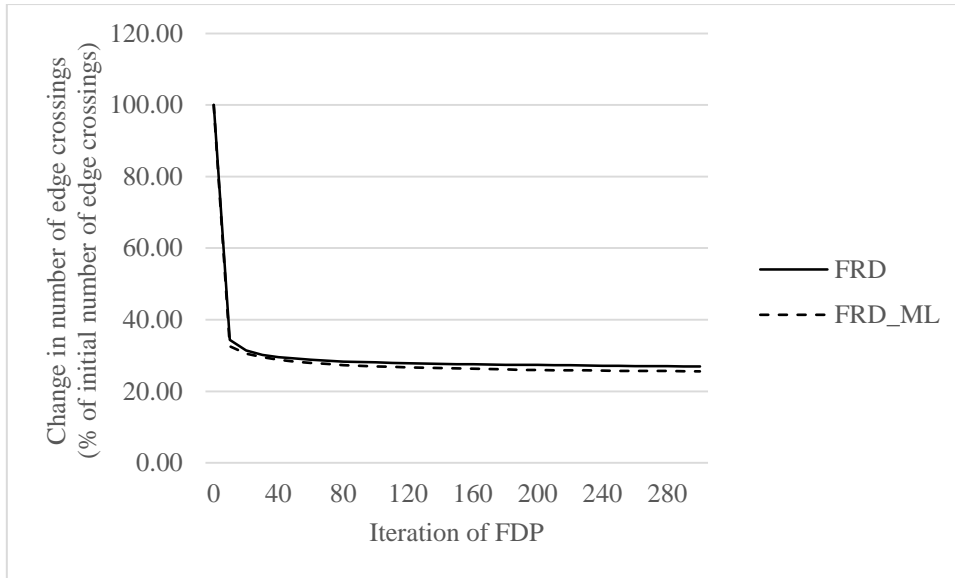


Figure 6.20. Change in edge crossings as layout is generated for general graphs of different sizes, exhibiting little difference between using Dynamic Spring Embedder (*FRD*) and Multilevel Dynamic Spring Embedder (*FRD ML*)

Comparison in graph size shows that there is a relatively larger drop in edge crossings for larger graphs (Figure 6.21). It is noted that this is the same for both *FRD* and *MLFRD* with only 1.88% difference between them (see Appendix 0 for more details).

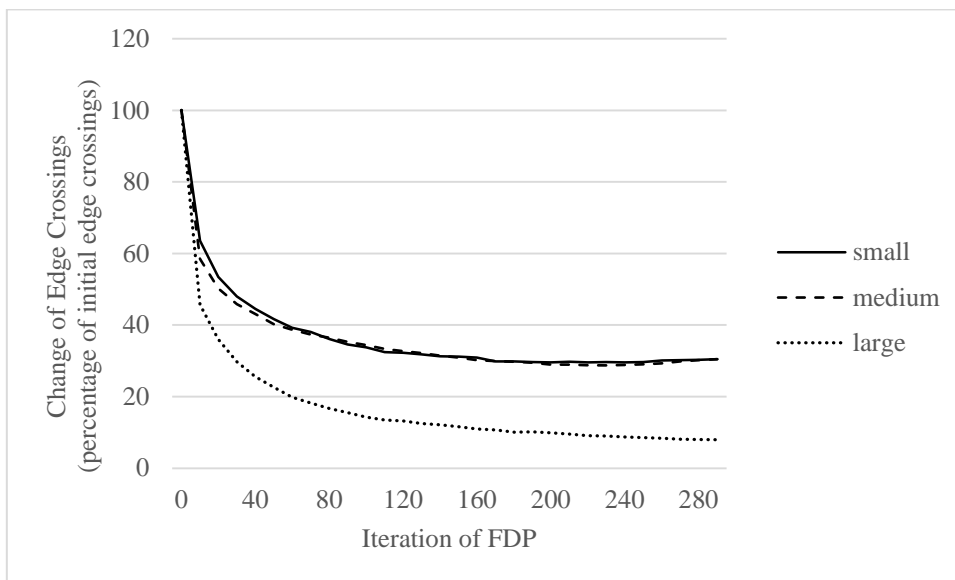


Figure 6.21. Comparison of the change in edge crossings exhibited in layouts for different sizes of graph when drawn using Multilevel Dynamic Spring Embedder

Analysis of average vertex movement offers similar suggestions, with the difference between algorithms reach a maximum of 5% difference, illustrated in Figure 6.22. Both algorithms

reach a similar point of minimum movement, yet given the curve on the lines, it can be expected that *MLFRD* will settle to higher vertex movement than the *FRD* counterpart.

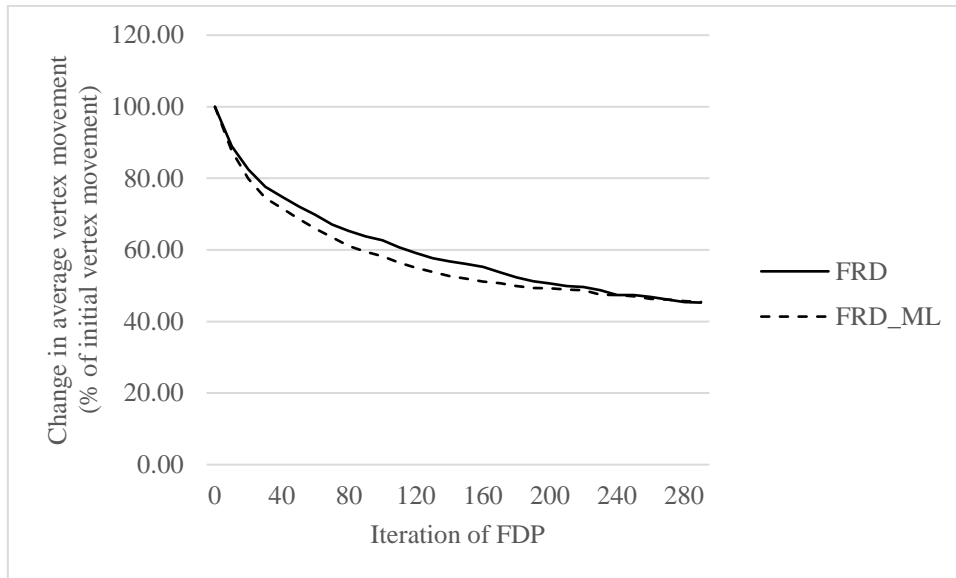


Figure 6.22. Change average vertex movement as layout is generated for general graphs of different sizes, exhibiting little difference between using Dynamic Spring Embedder (*FRD*) and Multilevel Dynamic Spring Embedder (*FRD ML*)

6.3.2 Subjective Analysis

Although the measured results above suggest little benefit of multilevel refinement in regard to edge crossings, the observed results are more promising.

Figure 6.23 and Figure 6.24 show examples of layouts for the two largest test graphs, regular-large and sparse-large, comparing layouts generated using the Dynamic Spring Embedder (left) and Multilevel Dynamic Spring Embedder (right). Both layouts exhibit noticeable improvement in global untangling, achieving fewer folds and overlaps within the layout.

In contrast Figure 6.25 compares the layouts for a smaller graph (regular-small), showing a warped layout as a result of the multilevel refinement, as suggested in the numerical analysis above.

Additional Layouts are included in Appendix 10.27 .

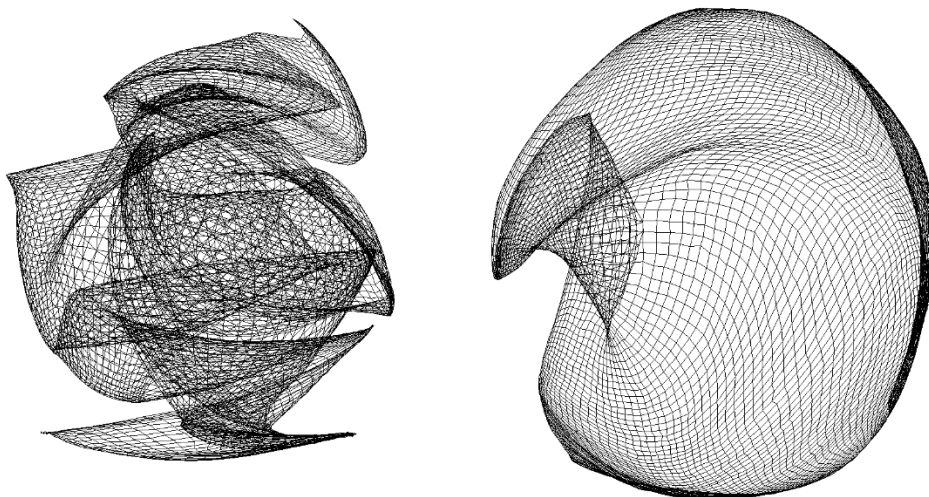


Figure 6.23. Comparison of layouts for the graph regular-large, generated using Dynamic Spring Embedder (left) and the Dynamic Spring Embedder with Multilevel scheme (right).

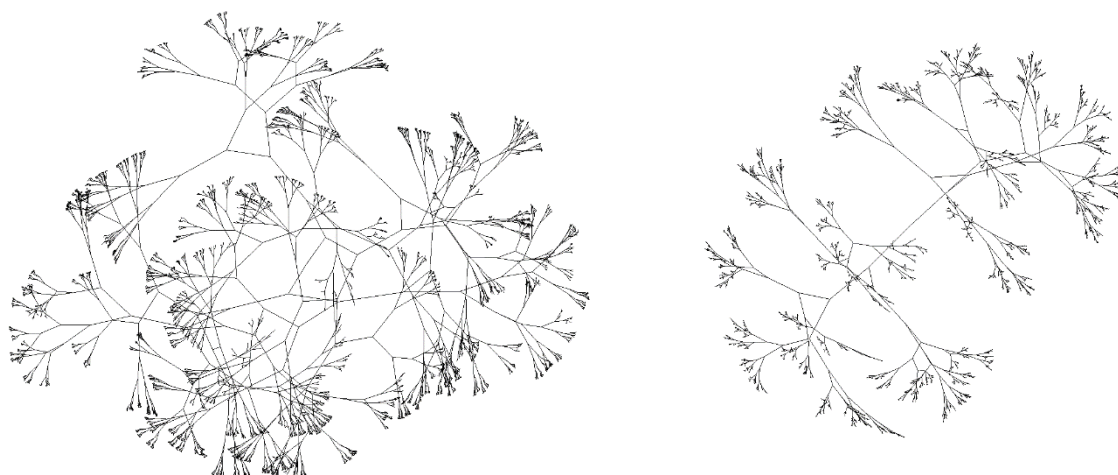


Figure 6.24. Comparison of layouts for the graph sparse-large, generated using Dynamic Spring Embedder (left) and the Dynamic Spring Embedder with Multilevel scheme (right).

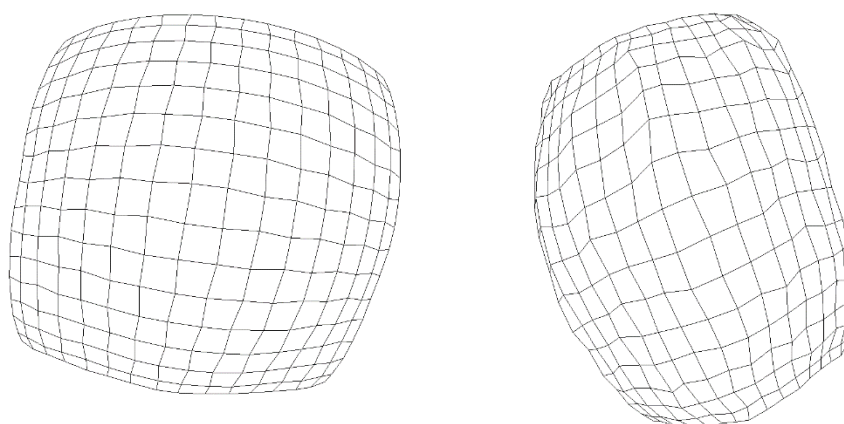


Figure 6.25. Comparison of layouts for the graph regular-small, generated using Dynamic Spring Embedder (left) and the Dynamic Spring Embedder with Multilevel scheme (right).

Storyboards depicting the improvement of layout for regular-large with and without use of multilevel refinement is provided in Table 6.4. Storyboards for other test graphs drawn using *FRD* are included in Appendix 10.24.1 .

The most obvious change in development is the movement of related vertices as controlled by the coarser multilevel graphs in *MLFRD*. In contrast, the *FRD* method places vertices individually requiring more development.

Although improving readability for the author, the changes of layout for larger graphs is unhelpful due to the high running times associated with the Spring Embedder.

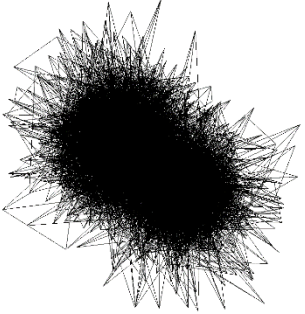
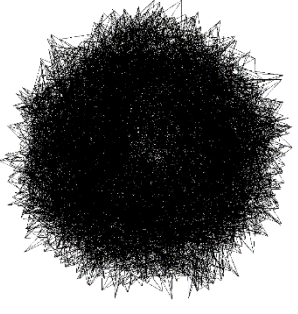
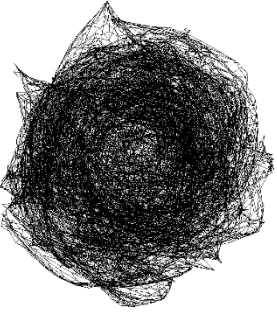
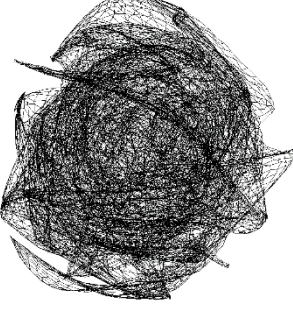
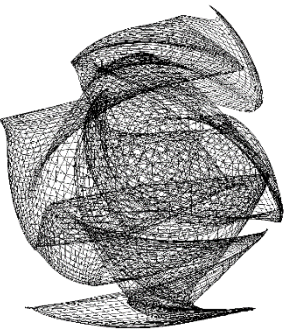
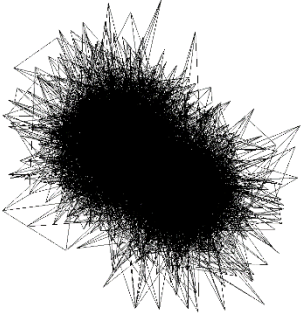

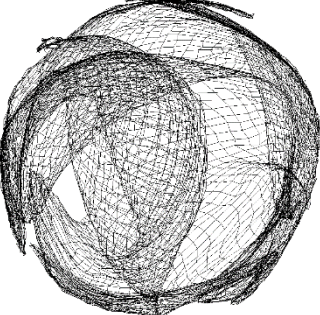
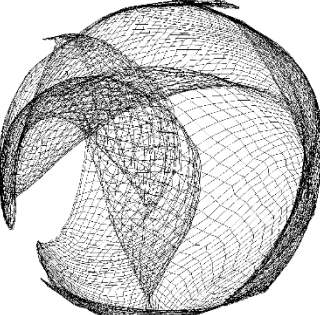
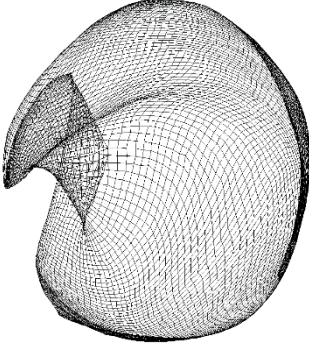
Frame 0	Frame 10	Frame 50	Frame 100	Frame 300
Regular-large using Dynamic Spring Embedder				
				
Regular-large using Multilevel Dynamic Spring Embedder				
				

Table 6.4. Comparison of the improvement of layout for the graph regular-large over 300 iterations using the Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder , showing the improvement on global layout as a result of multilevel layout refinement

6.4 Dynamic Modified Spring Embedder with Multilevel Global Force Approximation

As with the multilevel experimentation above, layout generation is applied to graphs from an initial randomised placement. Experiments are run for 300 iterations and are repeated 10 times for each graph, with running time being the priority measurement, and layout quality also being measured for comparison.

For Layout Adjustment, see Dynamic Matching (Section 6.5).

6.4.1 Configuration

Details of the configuration for the Dynamic Spring Embedder with Multilevel Global Forces are provided in Appendix 10.5 .

In summary, experimentation uses default values of 0.5 and 0.9 (0.7 for graphs with less than 1000 vertices) for the repulsive and spring force cooling schedules, which provide highest quality results in general. See Appendix 10.5 for the data.

6.4.2 Analysis of Numerical Results

Results for *MGF* approximation show a massive reduction in running time, between 95% for smaller graphs and 99.8% for the large test graphs. The reduction means that the dynamic spring embedder can be applied on larger graphs without long delays between frames, allowing for visualization through animation instead of still frames.

Graph	Running Time per Iteration (ms)	
	<i>FRD</i>	<i>FRD MGF</i>
sparse-small	15.47	0.88
sparse-medium	239.21	2.94
sparse-large	21384.07	38.28
regular-small	18.12	1.01
regular-medium	341.20	2.69
regular-large	25656.01	38.17
<i>dense-small</i>	337.95	1.91
<i>dense-medium</i>	8329.75	12.97

Table 6.5. Comparison of running time required to complete one iteration of Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) for various test graphs

Due to the extreme change in running time, smaller graphs change too quickly for any meaningful data to be extracted. Discussion regarding minimum and maximum frame rate for visualization is described in Appendix 10.17 .

Although reducing running time, use of *MGF* causes the number of edge crossing to rise by an average of 200% (Table 6.6). Smaller graphs show a larger increase in edge crossings of 462%, whereas medium have less at 133% increase and large graphs have least with 29% increase. This behavior is similar to the results for multilevel refinement, and suggests *MGF* is most useful for larger graphs.

Graph	Edge Crossings in Final Layout		Range in Edge Length	
	<i>FRD</i>	<i>FRD MGF</i>	<i>FRD</i>	<i>FRD MGF</i>
sparse-small	1.6	16.4	12.7860	10.1151
sparse-medium	22.4	89.4	25.1969	20.5631
sparse-large	1948.6	2601.4	48.7551	44.7275
regular-small	0.0	156.6	9.6504	7.9099
regular-medium	663.2	1371.0	12.9208	11.5247
regular-large	31830.6	39690.8	15.7241	15.0405
<i>dense-small</i>	4134494.0	4098098.6	3.6427	3.5225
<i>dense-medium</i>	385933360.4	362664547.0	3.4626	3.3121

Table 6.6. Comparison of edge crossings and range in edge lengths exhibited by layouts generated by Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) algorithms for various test graphs

It is important to note the difference in running time. Although generating layouts with poorer layout in 300 iterations, the time take to apply 1 iteration of Dynamic Spring Embedder on the graph regular-large, is the same time taken to perform 558.6 iterations with Multilevel Global Force approximation.

The results also show a minimal difference between range in edge lengths with or without use of approximation, suggesting little difference to the Peripheral Effect as a result of *MGF* usage.

6.4.2.1 Metrics for Graph Stability

The difference in the change in edge crossings over time between using *FRD* with and without *MGF* is negligible. This is especially pleasing as the interest is in reducing running time while minimizing the effect on layout quality and convergence.

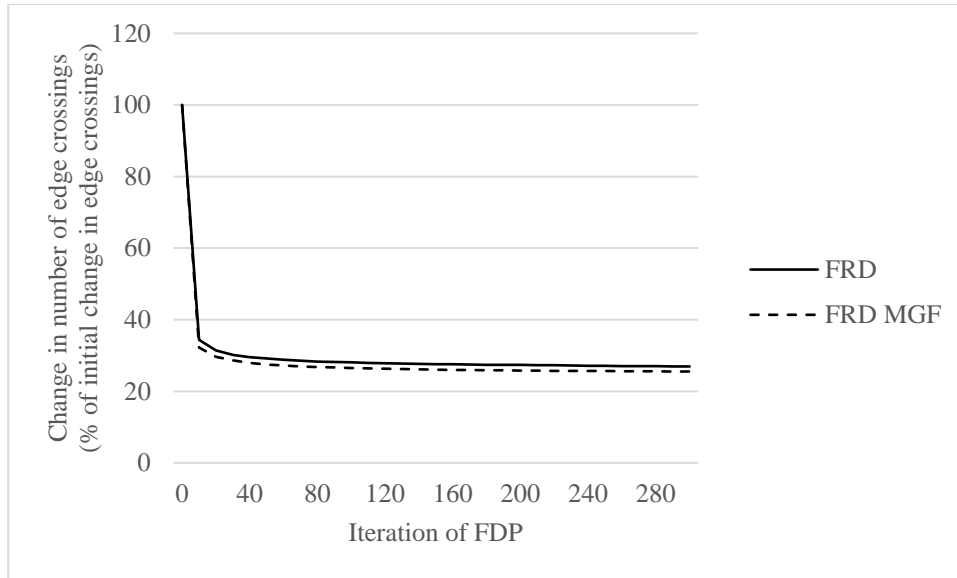


Figure 6.26. Change in edge crossings as layout is generated for general graphs of different sizes, exhibiting little difference between using the Dynamic Spring Embedder (*FRD*) and Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*)

Although minimal difference for the change in edge crossings, Figure 6.27 shows a 20% reduction in vertex movement when using *FRD MGF*. The drop in movement, and the similar rate at which movement is reduced (both algorithms reduce movement at similar rates between the 60th and 300th frame), suggests weakened forces being calculated when using *MGF*.

This may be the cause of the increased edge crossings, as reducing movement will slow the improvement of layout. Reconfiguring the *MGF* forces (described in Section 6.4) may reduce the difference.

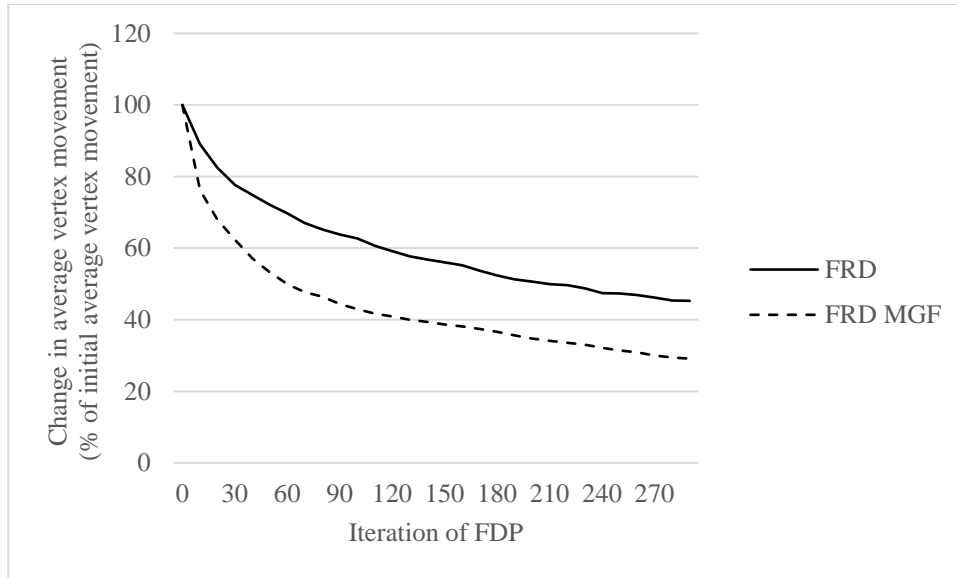


Figure 6.27. Change in average vertex movement as layout is generated for general graphs of different sizes, exhibiting a 20% decrease from using Dynamic Spring Embedder with Multilevel Global Force approximation (*FRD MGF*) in comparison to Dynamic Spring Embedder (*FRD*)

6.4.3 Subjective Analysis

As exhibited in the layouts generated using a multilevel scheme, the layouts generated for smaller graphs exhibit adverse effects on the readability. Figure 6.28 illustrates this by comparing layout for regular-small using the Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right), showing a fold in the layout caused by reduced displacement (Section 3.6.4.3).

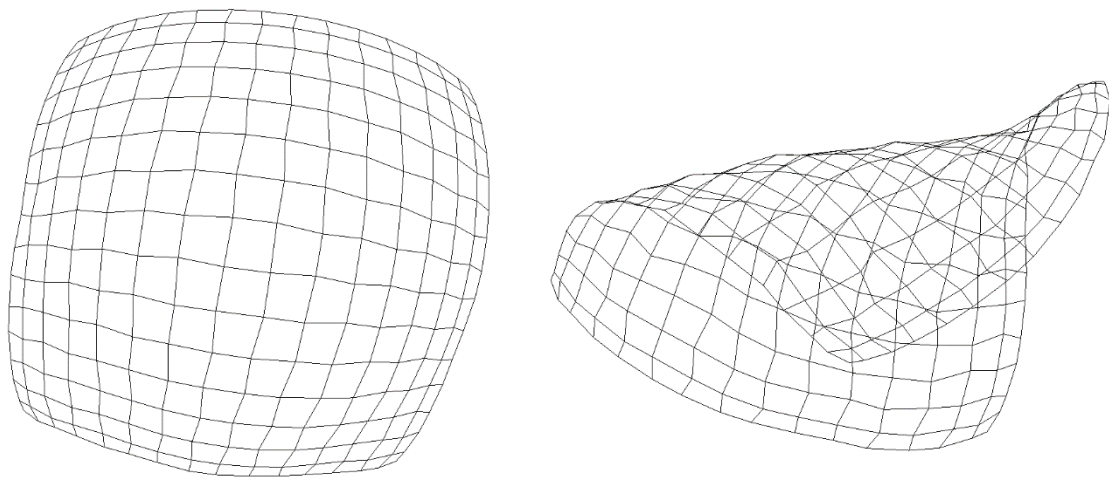


Figure 6.28. Comparison of layouts for the graph regular-small, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right), showing a fold in the layout as a result of the approximation

The effect on larger graphs is less noticeable with layouts having minor observable differences, however comparison is difficult due to drawings having poor global layout, exhibiting folds and overlapping layout. Although exhibiting poor global layout, the drawings for regular-large (Figure 6.29), sparse-large (Figure 6.30) and *dense-medium* (Figure 6.31) are similar to those generated using standard Dynamic Spring Embedder, making *MGF* useful for large graphs by reducing running time without depreciating layout quality.

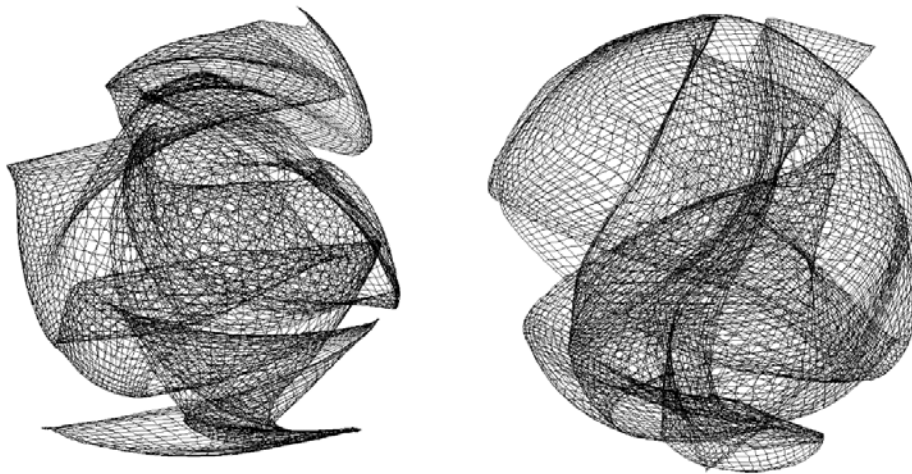


Figure 6.29. Comparison of layouts for the graph regular-large, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right)

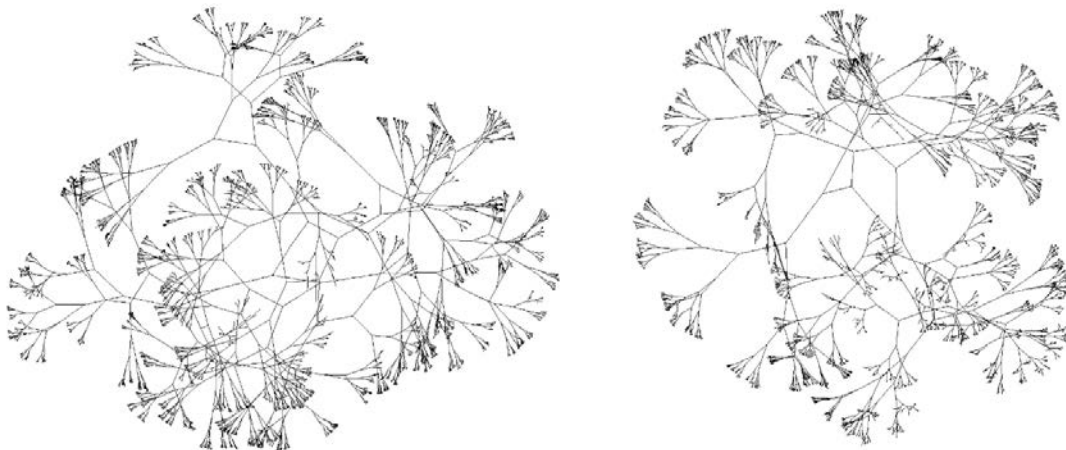


Figure 6.30. Comparison of layouts for the graph sparse-large, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right)

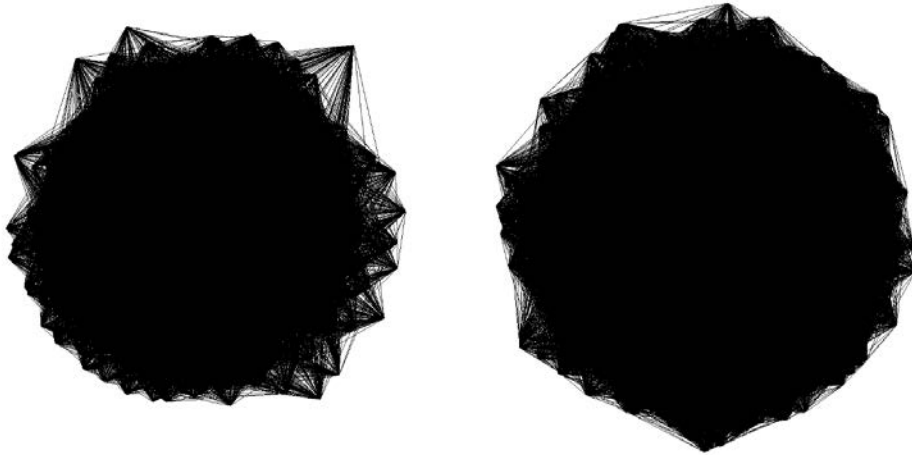


Figure 6.31. Comparison of layouts for the graph *dense-medium*, drawn using Dynamic Spring Embedder (left) and Dynamic Spring Embedder with Multilevel Global Force approximation (right)

6.5 Dynamic Matching

There are four update methods used to incorporate graph amendments into the multilevel schemes used by global layout refinement and global force approximation, each of which are expected to have varying impact on layout quality as a result. Experimentation aims to answer the following questions:

Is there measurable difference in layout quality and running time between usages of the four update methods for incorporating operations in to the multilevel scheme?

What does the difference indicate about the methods and operation types?

The methods are tested on two algorithms: the Multilevel Dynamic Spring Embedder and the Dynamic Spring Embedder using Multilevel Global Force, determining the impact on layout quality as a result of the amendments impacting coarser representations of a graph and the resulting MGF approximation.

For each of the test graphs, a layout exhibiting minimum edge crossings is generated prior to the experiment. Tests are run for 300 iterations of *FDP* application following the described procedure in Section 6.2 . Analysis of the metrics for graph stability is used to determine the effect the amendments have on the layout and the extent of changes to layout, verified by subjective analysis.

Information regarding the complexity of operations is provided in Appendix 10.22.4 . Additional investigation is given to the effect of updates on differing graph structures in

Appendix 10.18 , identifying some of the patterns which can be seen in the results below as a result of the test data.

6.5.1 Multilevel Matching Update

The primary focus of the experimentation here is to identify if incorporating graph operations into the coarser representations of a multilevel scheme increases the impact operations have on layouts. Analysis is performed on the results of 3 operations sets (outlined in Section 6.1) with differing update methods, from minimal (single level) updates to full updates of the multilevel scheme.

6.5.1.1 Quantitative Analysis

In general, the results show that the impact of the multilevel update is dependent on the type of operation. For *shrink* operations, there is little difference between update methods (Figure 6.32), with all update types increasing and decreasing edge crossings rapidly with no clear method providing consistently low edge crossings. On average, the *rematch* and *high update* methods result in higher edge crossings, with *single* and *middle update* method averaging least.

The impact on movement is summarised for the *maintain* operations below.

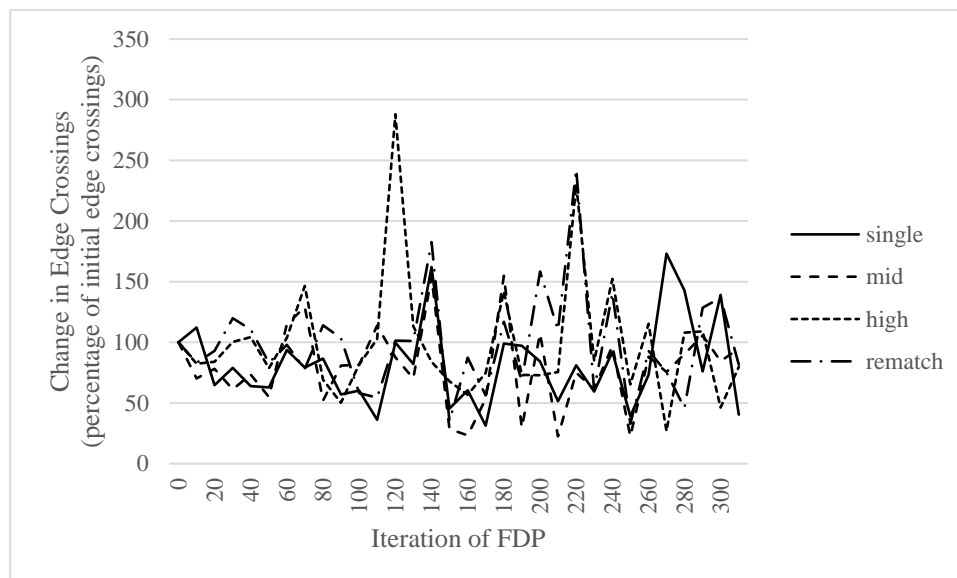


Figure 6.32. Comparison of the change in edge crossings for incorporation of *shrink* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

The results may indicate that, from a quantitative and measured standing, *shrink* operations are independent of update methods and cause large changes to layout regardless. In contrast, the *growth* operations show a direct correlation with the update methods. Figure 6.33 shows that the greater the propagation of operations through the multilevel scheme, the smaller the increase in edge crossings.

The *rematch* method provides results on par with the *single* update method, suggesting that preserving the matchings is better for reducing the edge crossings during growth operations, than having a multilevel scheme which better represents the structure (see Fragmentation, 7.3.2).

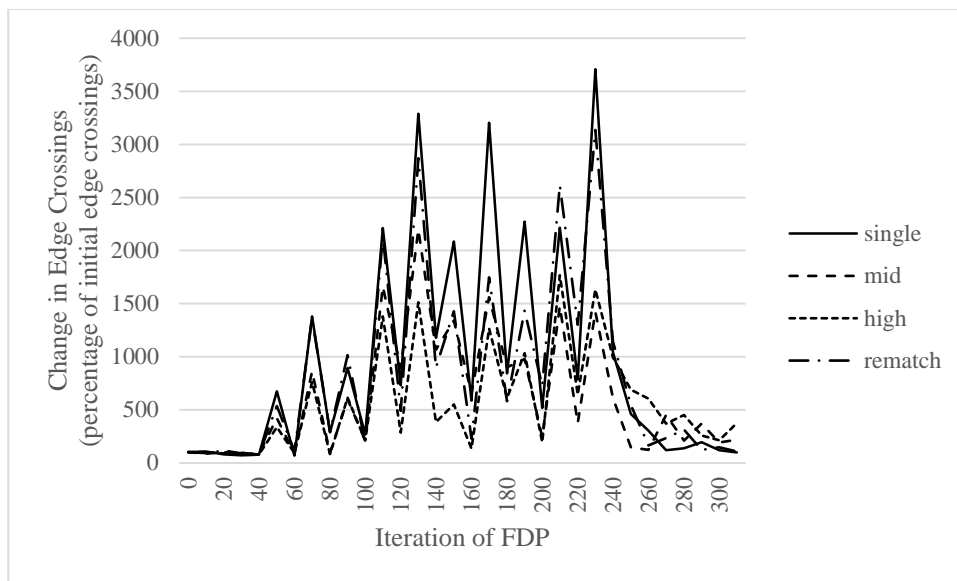


Figure 6.33. Comparison of the change in edge crossings for incorporation of *growth* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

To no surprise, the *maintain* operations follow the same behaviours as the methods above, with the *mid* and *high* update methods having least impact on edge crossings. During the final stage however, the change in edge crossings indicate that the single level update may be more beneficial, tying in with the suggestion that targeting specific structures can lead to improved performance (as found in Pattern Processing, Section 0).

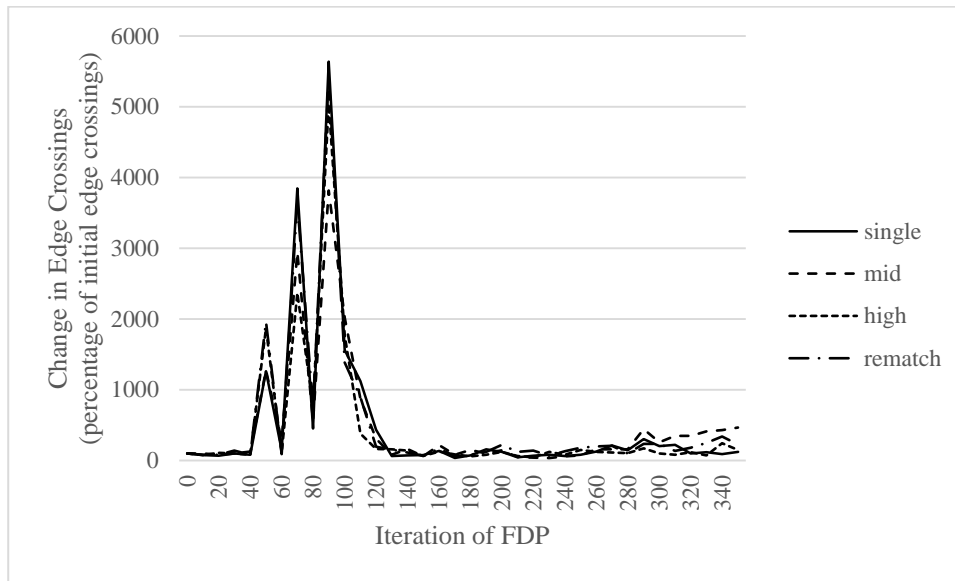


Figure 6.34. Comparison of the change in edge crossings for incorporation of *maintain* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

Movement of vertices between frames shows little difference between *shrink* and *growth* operations for all updates. The movement as a result of *maintain* operations is shown in Figure 6.35, showing the similarity in movement across update methods.

Throughout the operations, the high level update results in highest movement, with rematch generating least for growth and single for shrink operations. Although suggesting the methods are useful for different operation types, there is up to nearly 1000% difference between methods for the growth period, and a difference of 40% for shrink periods. Given that movement increases by 15% as a result of operations, the relative movement is minimal.

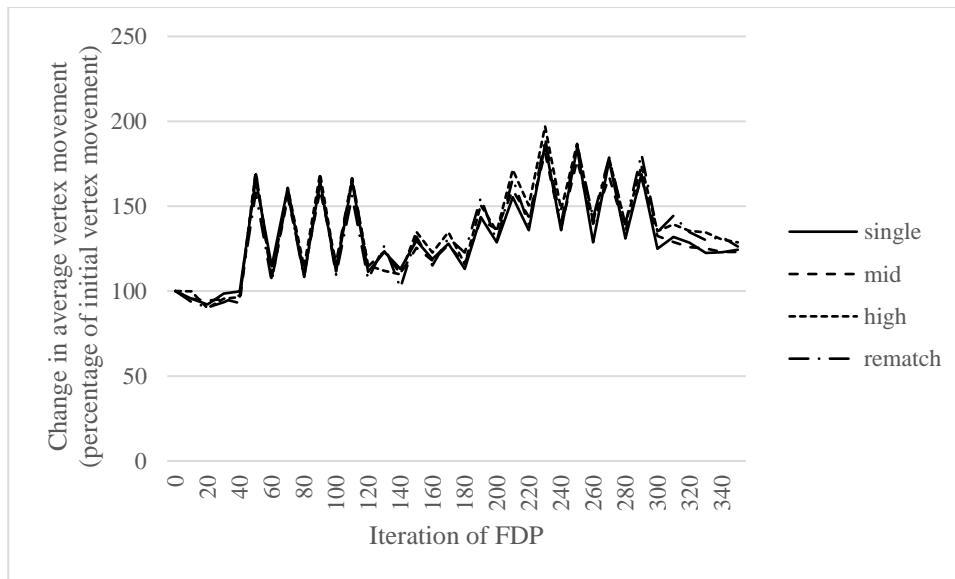


Figure 6.35. Comparison of the change in average vertex movement for incorporation of *maintain* operations into multilevel layout of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

6.5.1.2 Subjective Analysis

Due to the number of results, subjective is only provided for the final layouts, and where notable behaviours are exhibited, comments are made regarding changes over time which cannot be expressed as still images.

For *growth* operations on the *regular* and *sparse* type graphs, the frequent changes show a stacking effect, whereby vertices are added faster than the layout can be generated to place them, manifesting as compression at their anchoring points (described in Section 7.3.4).

As a result of the stacking effect, warping is observed during mid and high update methods (Figure 6.36 (right) and Figure 6.37 (left)), suggesting fragmentation within the multilevel scheme causes coarser graphs to “pull” on the existing layout. In contrast, operations using the single and rematch methods extend fluidly and symmetrically from the anchor points.

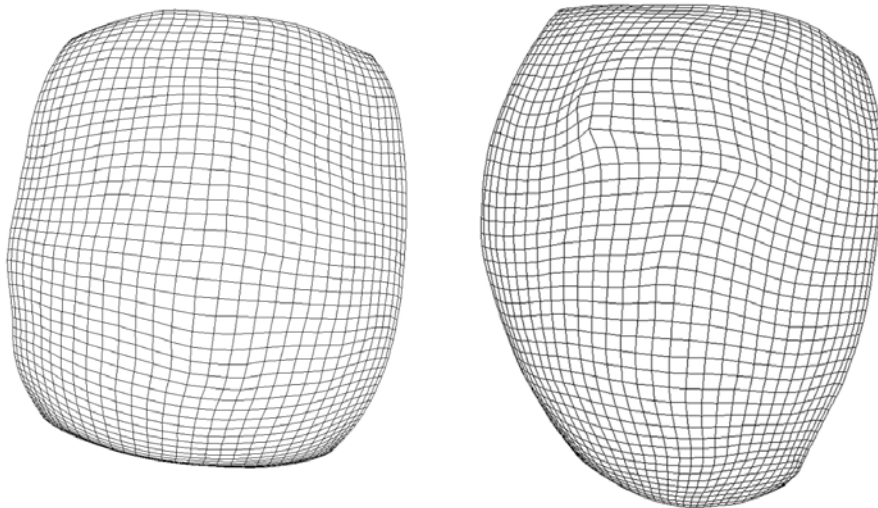


Figure 6.36. Comparison of layout generated for the graph regular-medium using Multilevel Dynamic Spring Embedder, using single-level (left) and mid-level (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout

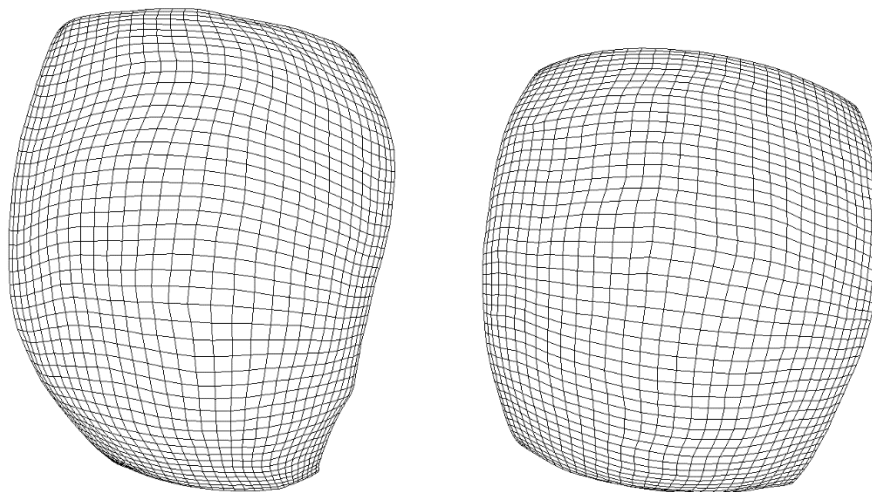


Figure 6.37. Comparison of layout generated for the graph regular-medium using Multilevel Dynamic Spring Embedder, using high-level (left) and rematch (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout

For dense graphs, the multilevel update plays a more noticeable role in the placement of addition of vertices within the layout. For single level updates (left), additional vertices and edges have little effect on the layout and do not escape the denseness of the layout (note there is only one noticeable protrusion on the bottom right of the drawing). In contrast, as the extent of the update is increased, the protrusions become stretched and clearer to the

observer, stretching out from the centre of the graph and becoming more visual as a result of the new vertices becoming more prominent in coarser layouts.

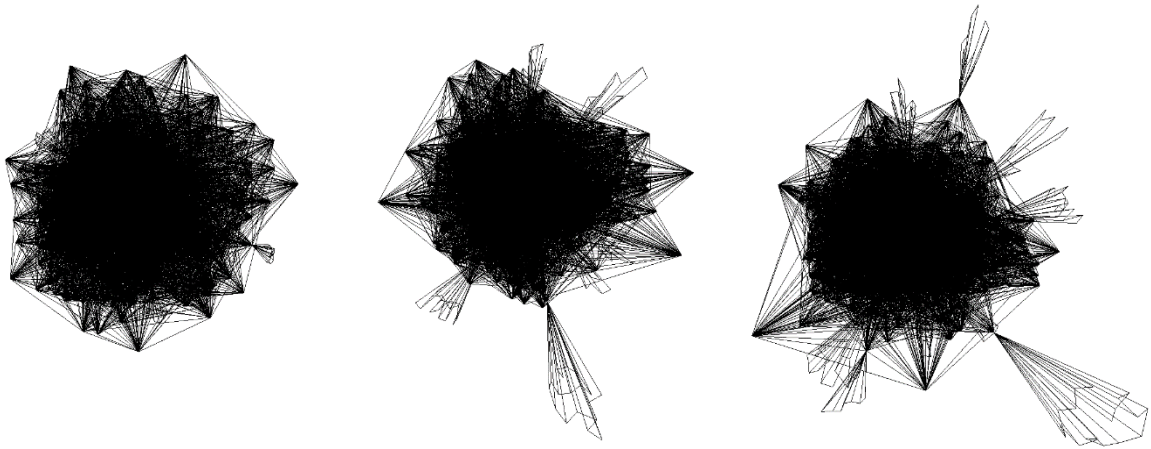


Figure 6.38. Comparison of layout generated for the graph *dense-small* using Multilevel Dynamic Spring Embedder, using single-level (left), mid-level (center) and high-level (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout

In contrast to the *growth* operations, the *shrink* operations show little effect as a result of the update methods. Figure 6.39 provides an example of sparse-medium, showing similarity of layout between a single level update and a full rematch. Similar behaviour is noted across all graphs using *shrink* operations, whereby removal of vertices causes high movement but minimal edge crossings.

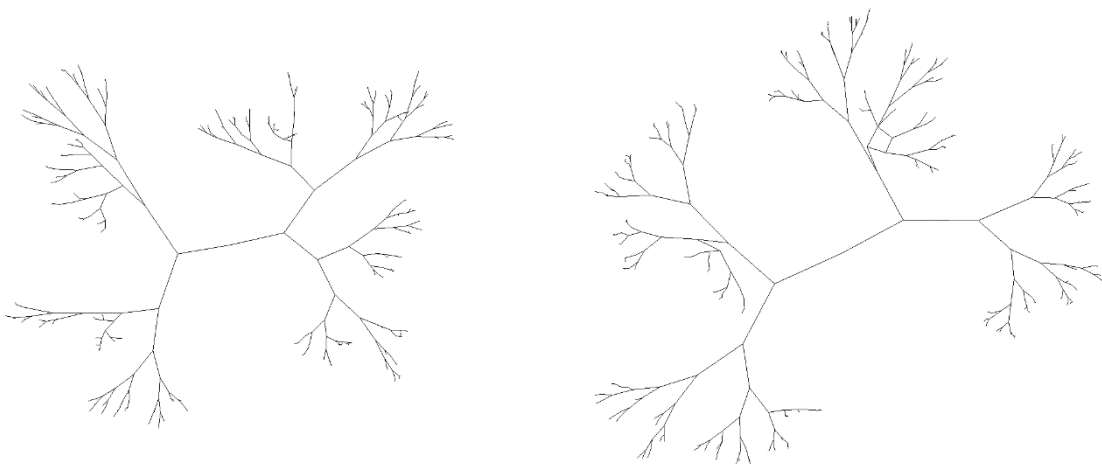


Figure 6.39. Comparison of layout generated for the graph *sparse-medium* using Multilevel Dynamic Spring Embedder, using single-level (left) and rematch (right) update methods for incorporating amendments into the multilevel scheme and associated multilevel layout

6.5.2 Multilevel Global Force Updates

As with the multilevel refinement method above, the same testing is applied to the Multilevel Global Force approximation scheme, the intention of which is to identify if there is any impact on layout quality from changing approximation.

6.5.2.1 Quantitative Analysis

The results highlight a more noticeable impact on the metrics of graph stability than seen when testing the multilevel refinement.

The chart in Figure 6.40 shows that *shrink* operations have higher impact on edge crossings with single level update (increasing edge crossings by 800%), whereas the rematch method includes operations more fluidly, increasing and decreasing operations as seen in Figure 6.40 for the Spring Embedder alone.

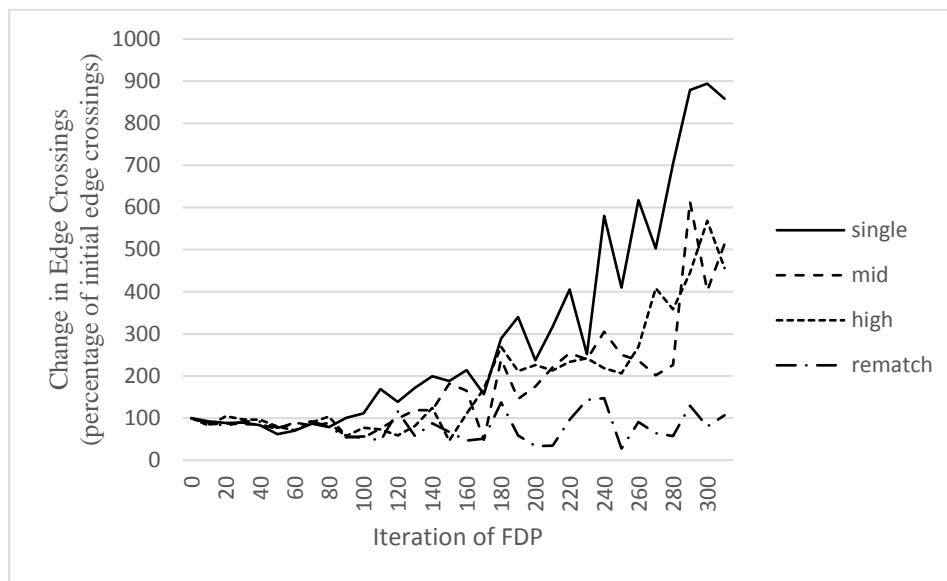


Figure 6.40. Comparison of the change in edge crossings for incorporation of *shrink* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

Figure 6.41 illustrates that single level updates cause minimal movement, suggesting the vertices cannot move past local minima, causing increased edge crossings. In contrast, the rematch method is able to better apply forces to vertices and quickly provide low energy positions, increasing movement as the graph increases in size but equally reducing movement as minima is found.

The mid and high level updates generate larger amounts of movement associated with fragmentation in the multilevel scheme, whereby the changes cause ‘loose’ branches of vertices with little weight, which are greater affected by the denser branches of the matching tree (see Fragmentation, 7.3.2).

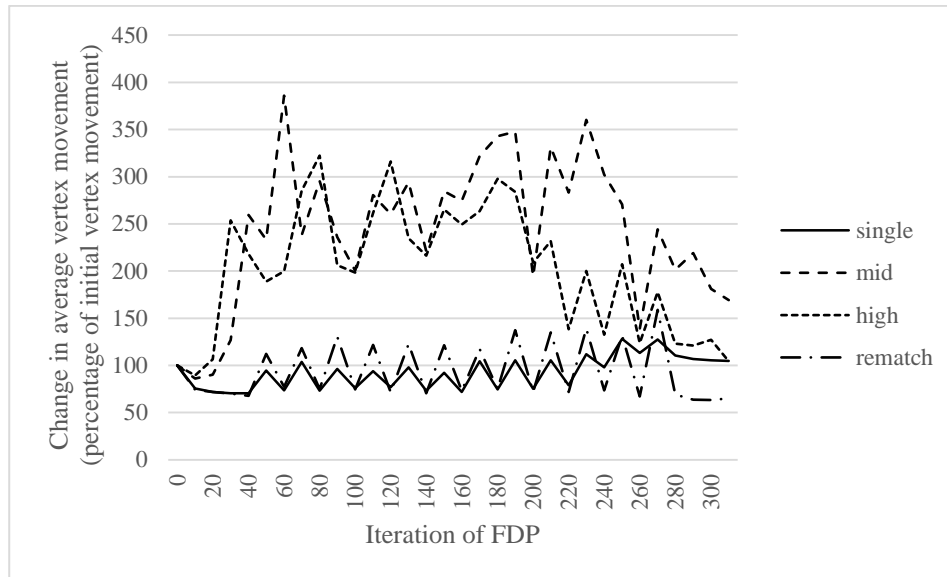


Figure 6.41. Comparison of the change in average vertex movement for incorporation of *shrink* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

Growth operations operate in a similar way, with large differences between the update methods. The mid-level updates provide least changes in edge crossings between iterations, whereas rematch causes the greatest increase, likely due to the increase in possible matchings at the lowest levels. The performance of the mid update is thought to be a result of both preserving layout and also incorporating changes (enforcing both local and global layouts).

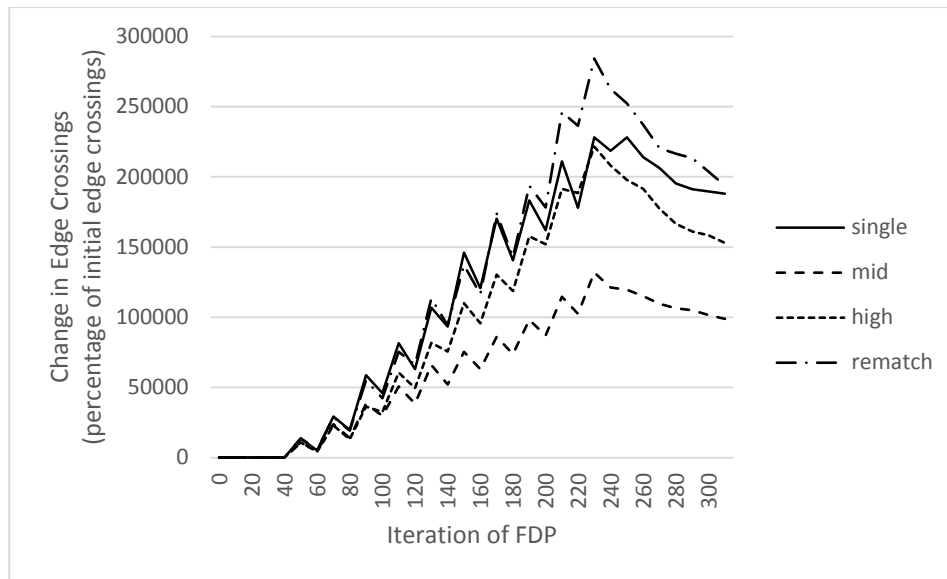


Figure 6.42. Comparison of the change in edge crossings for incorporation of *growth* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

Movement during *growth* operations is similar to that of the *shrink* operations, with the exception of mid updates which no longer cause high movement and instead cause least. The change may be attributed to many things, but is expected to be primarily from good initial placement of vertices as defined by the growth operation set.

The high movement from the high level update is thought to reflect the large changes in the coarsest graphs, whereby adding new vertices causes the coarsest graph to increase from two heavy vertices to include multiple lighter vertices which are more susceptible to force.

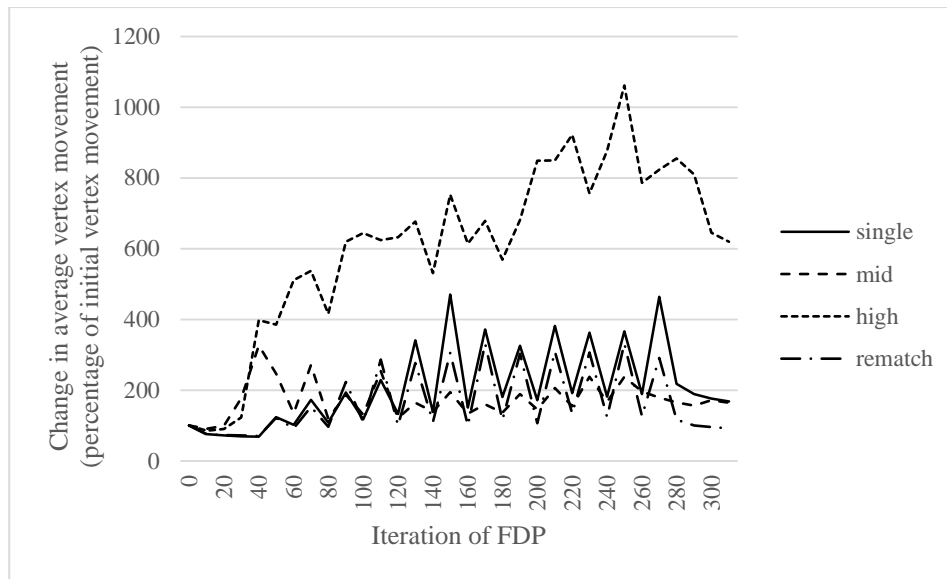


Figure 6.43. Comparison of the change in average vertex movement as a result of incorporating *growth* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

The *maintain* operations mirror the performance above for both changes in edge crossings and movement, as depicted in Figure 6.44 and Figure 6.45.

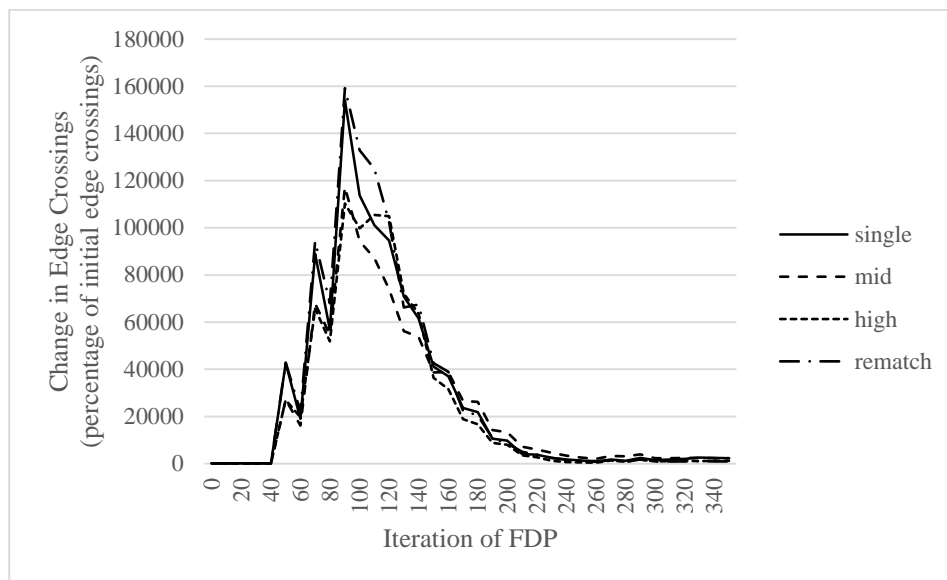


Figure 6.44. Comparison of the change in edge crossings movement as a result of incorporating *maintain* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

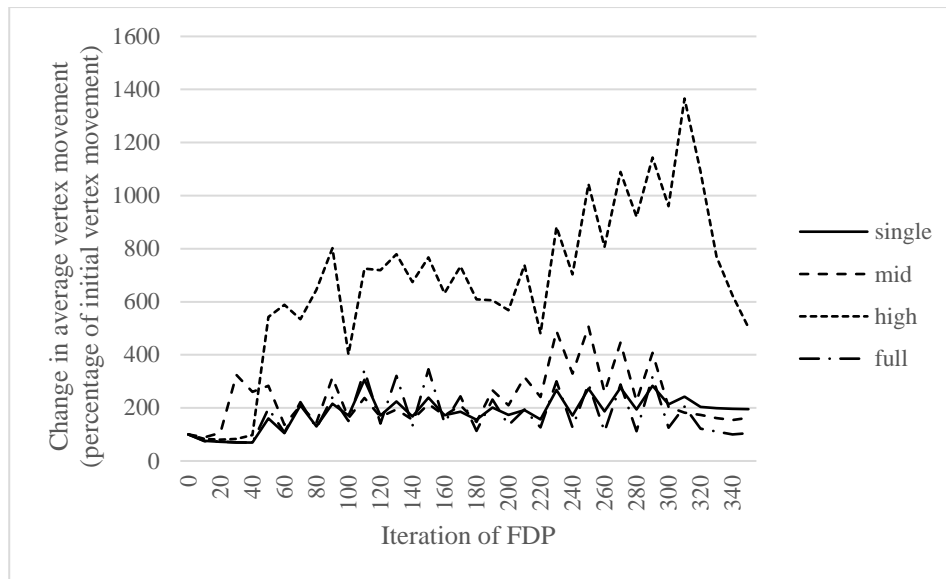


Figure 6.45. Comparison of the change in average vertex movement as a result of incorporating *maintain* operations into Multilevel Global Force approximation of the dynamic test graph collection, using the four update methods controlling the extent of the update into the multilevel scheme used

6.5.2.2 Subjective Analysis

One of the biggest observed differences of the update methods is the fluffiness of a graph, whereby the orientation of vertices can make a graph expand outward from the global centre of mass or the local centres. Sparse graphs are best for visualising this, as shown for sparse-medium in Figure 6.46, the *maintain* operations are included using mid-level (left) and high-level (left) updates.

By including changes at a local level, vertices can be placed well according to their surrounding vertices, taking into account both the global and local center of mass. When propagated to the higher levels of the approximation, the vertices are more susceptible to, and are pushed away from, the global center of mass. The two methods can be useful and applied for specific interest in local or global layouts.

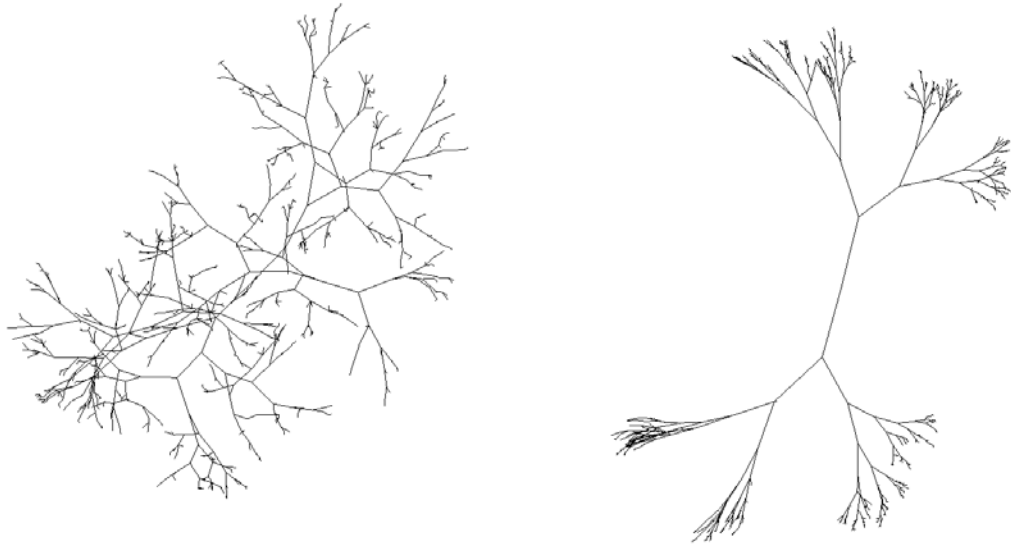


Figure 6.46. Comparison of layouts for sparse-medium using Dynamic Spring Embedder with Multilevel Global Force approximation after application of *maintain* operations, using mid-level (left) and high-level update methods, showing noticeable difference in the spread of vertices as a result of the methods

The rematch method somewhat circumvents this behaviour by removing the fragmentation and providing equal layout as would be expected in a static drawing. The changes to the matching tree will be similar to the original due to adjacent vertices being matched, however some warping can be seen in the local layout as matchings differ (as depicted in Figure 6.47).



Figure 6.47. Layouts for regular-medium, showing the minor change in vertex positions as a result of the Multilevel Global Force structure being altered during the rematch update method. Edges are coloured based on their length, with the warping shown as edge lengths changing (and colours changing)

The stacking behaviour described previously is a noticeable problem in growth operations on regular graphs, depicted in Figure 6.48. The addition of vertices with only single level updates results in very little placement being applied to vertices due to vertices being added

too quickly (left), in contrast using the mid-level updates provided some layout but still suffers from warping as forces cannot push the layout out enough (right).

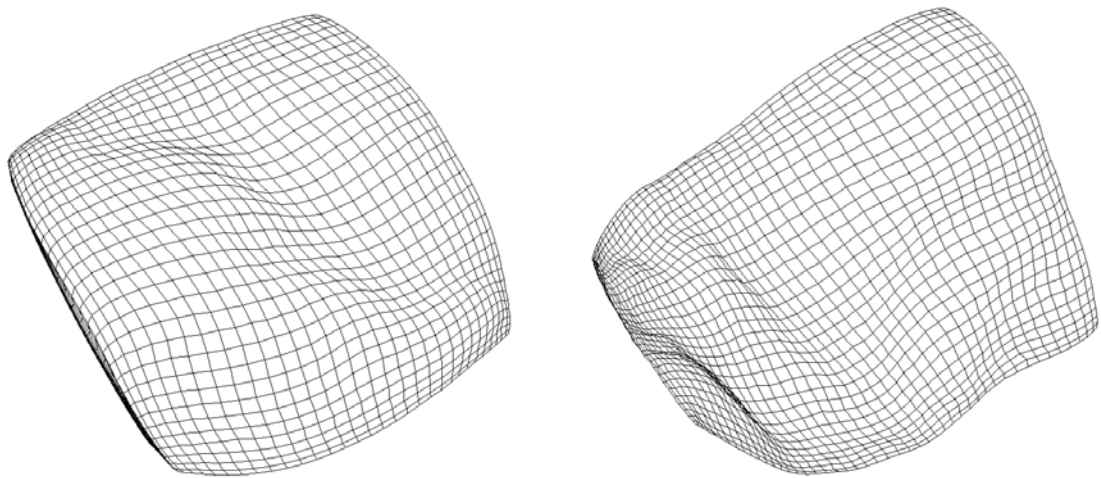


Figure 6.48. Comparison of layout generated for the graph regular-medium using Dynamic Spring Embedder with Multilevel Global Force approximation, using single-level (left) and mid-level (right) update methods for incorporating amendments into the multilevel scheme and associated approximation

Full and rematch updates improve the incorporation of vertices into the *MGF* approximation structure resulting in improved expansion of the layout.

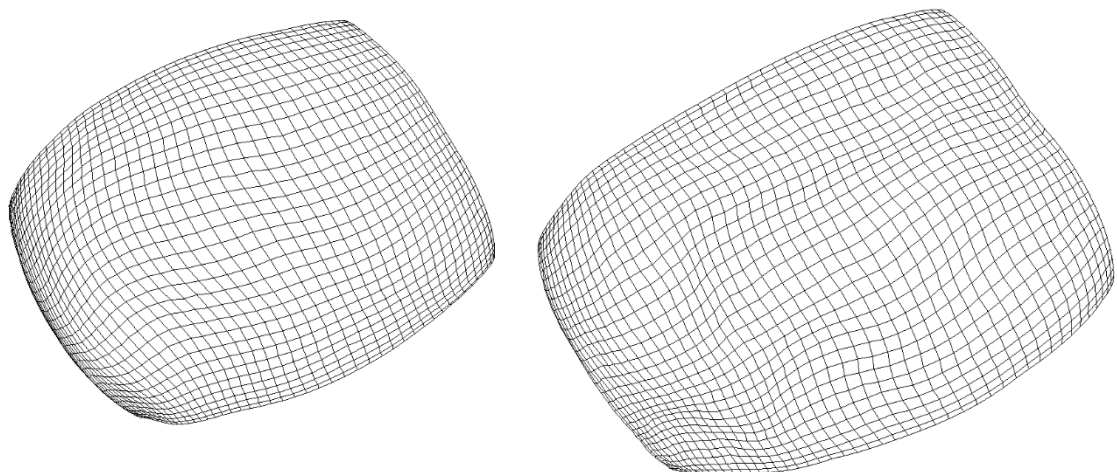


Figure 6.49. Comparison of layout generated for the graph regular-medium using Dynamic Spring Embedder with Multilevel Global Force approximation, using high-level (left) and rematch (right) update methods for incorporating *growth* amendments into the multilevel scheme and associated approximation

Although medium and large graphs are provided with moderate layouts, the smaller graphs show little to no benefit from using Multilevel Global Force. As shown for sparse-small

below, the layouts generated after the maintain operations have folds regardless of the update methods used.



Figure 6.50. Comparison of layout generated for the graph *sparse-small* using Dynamic Spring Embedder with Multilevel Global Force approximation, using single-level (left) and rematch (right) update methods for incorporating *maintain* operation amendments into the multilevel scheme and associated approximations

On denser graphs such as *dense-medium* which have a largely disorganised layout, the updates methods can provide flexibility in how to display some types of updates (particularly growth)

The high level update results in fragmentation which causes the looser vertices to be pushed outwards from the existing global layout (as shown in Figure 6.51). In contrast, using the single level operations causes the new vertices to be treated as their own subgraphs, with the repulsive forces of the original layout pushing them outwards (Figure 6.51 (left)). Rematch generates the least visually appealing layout (from the authors perspective), with the new subgraphs being affected by both the global and local layouts, somewhat conflicting the layout.

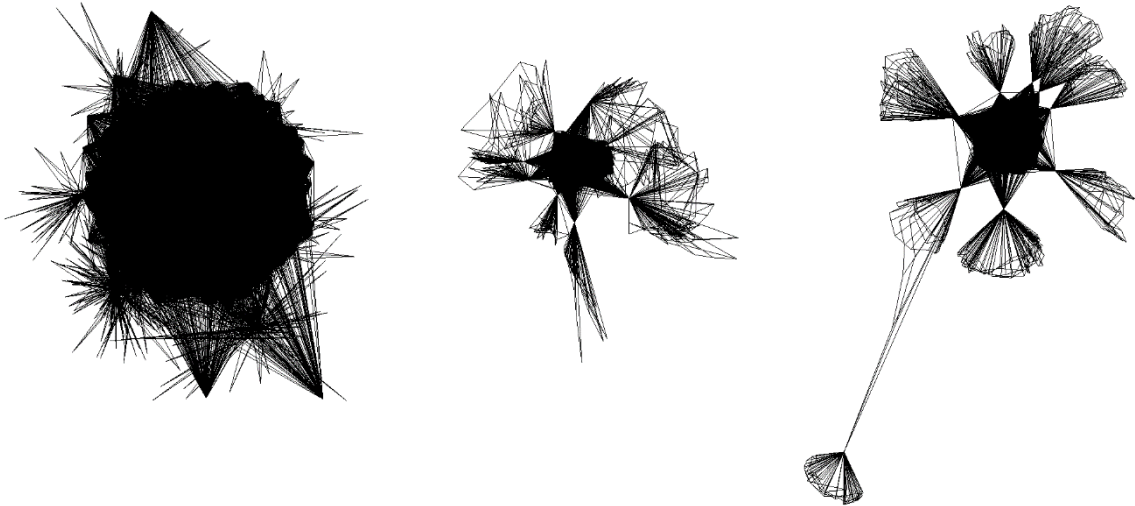


Figure 6.51. Comparison of layout generated for the graph *dense-medium* using Dynamic Spring Embedder with Multilevel Global Force approximation, using high-level (left), rematch (center) and single-level (right) update methods for incorporating *growth* operation amendments into the multilevel scheme and associated approximation

6.6 Summary

The experimental methodology is designed to test the fitness for purpose of each of the new algorithmic approaches, investigate configurations which provide improved results and provide comparison to state-of-the-art techniques. Experimentation is performed on each of the algorithms in regard to a benchmark to identify whether the techniques are applicable to graph drawing, with results collected for analysis in the Results and Evaluation chapters.

Each of the experiments are performed using implementation of the proposed and state of the art works in a framework using common algorithmic design, data structures and runtime environments. The aim of which is to keep comparison of techniques as accurate as possible.

Although quantitative results are collected for accurate and measurable comparison, results assume that the metrics used coincide with the readability and usefulness of layouts. Subjective analysis is provided for a broader evaluation of layouts and algorithm performance and confirm or refute the quantitative analysis.

Multilevel Global Force approximation is shown to provide similar results to the Barnes Hut Octree, with a noticeable (41%) decrease in running time. In addition, it is believed the structural information of the layout is encapsulated and applied during approximation of

repulsive forces, making for a more accurate representation of the graph (as opposed to using an arbitrary structure placed atop the graph, approximating in N directions outwards).

Various configurations have been investigated, with a default algorithm identified and described in Conclusions. Investigation into additional coarsening methods to improve the representation of structures within the graph has also been undertaken.

During investigation of dynamic drawing, an adaptation of the Fruchterman-Reingold Spring Embedder is suggested, applying the force functions used to overcome minima in static graph drawing to the dynamic graph drawing area. Multilevel schemes have been introduced in combination with the Dynamic Spring Embedder, providing a means to improve global layout generation and reducing running time through application of the *MGF* approximation.

Following the introduction of the multilevel methods, Dynamic Matching methods for incorporating amendments to a dynamic graph into the multilevel matching (and thus global layout and approximation schemes) has been investigated, highlighting the behaviours associated with different types of operations and providing suggestions for which update methods are beneficial for dynamic drawing.

7 Evaluation

This chapter evaluates the results collected during experimentation and determines if the new algorithmic approaches have been successful in their aims, and if when combined, they are able to reduce the complexity of dynamic graph drawing and provide high quality drawings for large dynamic graphs.

7.1 Multilevel Aesthetic Analysis

Although not directly related to the reducing complexity or improving Layout Quality, the method does offer a more detailed means of measuring quality of a layout, particularly global layout.

Within the literature, there is little research regarding analysis (particularly quantitative analysis) of layouts for large graphs with other 1,000 vertices. Current methods apply the aesthetics known to impact small graphs (e.g. edge crossings by Walshaw, 2003).

Global Layout Analysis did show that changes in global layout can be measured in the approximate layout through measuring differences in edge crossings, however, the reduction of edge crossings is also apparent in the layout of the original graph, offering little benefit from measuring Multilevel aesthetics.

Analysis of the layouts for partitions generated by the multilevel scheme provides insight into the layouts for groupings of vertices, with the examples in Section 4 showing local layout quality is identified in some areas of the example graph. In combination with Global Layout Analysis, the approach can give a detailed report of a layout.

Similarly, usage with metrics for graph stability is also recorded for layout generation using algorithms for dynamic graph drawing. The example provided in Section 4 identifies differences between the Eades Spring Embedder and Dynamic Spring Embedder, showing that the former gets trapped in global minima (higher number of edge crossings in the approximate layout) while local layout continues to improve. In contrast, the latter method provides fewer edge crossings in the approximate layout, corresponding to improvements in the untangling of the graph, confirmed with subjective analysis.

Multilevel Aesthetic Analysis provides quantitative results which give insight into the changes to layout, on the local and global scale, allowing for a more thorough interpretation

of developing layouts. The example provided for static layout analysis shows little improvement however, with the relative change in edge crossings for the initial layout similar to those in the approximate layout.

The results encourage further investigation into the potential uses of Multilevel Aesthetic Analysis, especially in comparing developing layouts (such as those in Dynamic Graph Drawing). Although useful for providing numerical comparison, the similarity in results for layouts and approximate layout makes usage redundant in combination with subjective analysis, which generally provides more detailed conclusions regarding perceived global layout.

7.2 Multilevel Global Force (*MGF*) Approximation

The aim of *MGF* is to reduce the running time associated with the Spring Embedder (or other force directed placement method). Previously it has been noted that limiting the distance at which repulsive forces take effect (Fruchterman-Reingold, 1991), and use of space decomposition approximation (Fruchterman-Reingold (1991) and Hu (2003) may result in reduced to running time, and therefore Multilevel Global Force is assessed in comparison to these.

7.2.1 Running Time and Complexity

The primary investigation comparing running time between Multilevel Global Force and Octree usage indicates that a reduction of 40.1% is achieved (Section 5.2.1.1). The results suggest the reduction becomes more prominent in larger graphs, for example one of the smaller test graphs (data) shows a reduction of 27.8% in contrast to one of the larger test graphs (*dime20*) showing a reduction of 50.52%.

Although several causes may exist, the controlled environment (Appendix 10.22.1) and decomposition of running time (Appendix 10.1) suggests the change in complexity is the primary cause. The change reduces complexity from $O(n \log n)$ associated with Octree usage (Barnes and Hut, 1986) to $O(n \cdot (L-1) \cdot m-1) + (/G_c/-1))$ for Multilevel Global Force, where n is the number of vertices in a graph, L is the number of graphs in the multilevel scheme, m is the matching number, and G_c is the coarsest graph in the multilevel scheme. This can be simplified to $O(n \cdot L)$ when $/G_c/ = 2$.

Other attributes are additionally expected to impact running time and complexity. For example Octree incorporates limits for the number of splits generated to prevent an infinite tree Hu (2006), a feature Multilevel Global Force does not include. In contrast, the generation of the Octree requires additional time to be generated, unlike Multilevel Global Force which can be exploited from an existing multilevel scheme. The impact of such attributes can be seen in Results and Appendix 10.1 (deconstructing running time), and act by altering the value of L , m and $|G_c|$ above.

7.2.2 Layout Quality

Analysis of numerical results for layout quality identifies a 13.5% increase in edge crossings as a result of using Multilevel Global Force over use of an Octree, implying that layouts are more difficult to read, however also show that the range in edge crossings between repeated drawings is reduced by 92.48% than the Octree, showing layouts generated using Multilevel Global Force are more regular (repeated layouts are measurably similar).

The increase in edge crossings is associated with the difference between the mapping of vertices to approximation (approximation tree structure) used by the Octree and Multilevel Global Force. The impact on force calculation is approximation of long range repulsive forces in reduced directions (the Octree approximates forces in four directions per split and Multilevel Global Force approximates in two), changing the directions of expansion within the layout and introducing warping.

Larger graphs show a lesser effect from warping (a result of vertices with high weight in coarser graphs overcoming the high weights of the coarsest graph) but more prominent Peripheral Effect (Hu, 2005), whereby the repulsive forces cause expansion outwards from points of high mass within the layout (large groupings of vertices) and compression of the precipice of sub graph structures (observed as a more defined global structure).

Further changes in edge crossings come from overlaps in the layout, however, subjective analysis identifies a likeness between drawings using Octree and Multilevel Global Force approximation methods.

7.2.3 Configuration

As with much literature (Fruchterman and Reingold (1991), Walshaw (2003), and Hu (2006)), experimentation is performed to identify and suggest an optimal configuration for a proposed algorithm. The methods used to achieve this are largely focused on the structure and attributes of the Multilevel Global Force structure and further reducing complexity of repulsive force calculation.

7.2.3.1 Limiting approximation

Typical implementations of force directed placement (Walshaw (2003) and Hu (2003)) include some limitation on global n-body forces (Fruchterman and Reingold, 1991), preventing weak interactions between distant vertices. In order to determine the impact on layout, two methods are compared for incorporating such function, a limitation on spatial distance (r), and a limitation on approximations (a limit on the traversal of the approximation tree).

Comparison of the limitations shows a common and expected behaviour, the larger the limitation the greater the impact on layout and running time (disproportionately reduced, see below). Both spatial and approximation limit follow the same rate of change as indicated in Appendix 10.7, with some difference as a result of spatial distance and traversal of the approximation tree.

Between values of 0 and 30 (representing the extent of traversal from none to 30 levels), an optimal value of 15 is suggested for approximation limit (corresponding to traversal to the 15th coarsest graph), resulting in a 40% decrease in running time and 20% increase in edge crossings. A value of 14 is suggested for spatial limit (corresponding to distance equal to $14 \cdot k$).

Values provided in other publications (most notably Hu, 2005) provide different configurations for using spatial limitation, however, are incomparable due to differences in implementation. As such, the values suggested may have different impact between implementations.

7.2.3.2 Coarsening Tolerance

Limiting the number of graphs within a multilevel scheme has had little exposure in previous research, described briefly as a means to prevent successive abstractions of a graph with the

same structure being generated as part of a multilevel scheme (Walshaw, 2003). Investigation into the values suggests that reason for low exposure is the disproportionate increase in edge crossings (decrease in layout quality) relative to any benefit obtained (reduction in running time).

A tolerance of 2 vertices between successive abstractions is suggested, providing negligible difference to running time and edge crossings across test graphs on average. The value is applied during the coarsening scheme (see Section 0) and prevents graphs with a difference less than 2 being generated. Higher values show little decrease in running time and noticeable increase in edge crossings (a value of 20 results in 5% decrease in running time and up to 240% increase in edge crossings).

Due to no other function being altered, the cause of the increase in edge crossings is attributed to the coarsest graph which is generated, and the number of directions in which approximation occurs. For example, if a coarsest graph of 100 vertices is generated and provided a layout which exhibits a fold, this layout then becomes the foundation for all finer graphs. Further subjective analysis supports the conclusion showing folds and overlaps for larger graphs. The values provided match the usage described in Literature (Walshaw, 2003).

A more adaptive method is suggested in Primitive Graphs (see below).

7.2.3.3 Multimatching, Primitives and *MGF* Structure

The structure of Multilevel Global Force is dependent on the multilevel or multiscale scheme used, which here uses edge contraction as described by Walshaw (2003). Other multilevel and multiscale methods may use alternate approaches for generating a hierarchy of approximate graph structures (such as identifying related vertex clusters, Hachel and Jünger (2005) or identifying a Maximal Independent Vertex Subset, Gajer et al (2001)) providing alternate approximation tree structures if utilized as Multilevel Global Force.

Investigation here examines the impact of altering the structure of a multilevel scheme described by Walshaw (2003) by adapting the edge contraction method used to generate it.

7.2.3.3.1 Multimatching and Brute Coarsening

Changing the number of vertices being matched during the coarsening process alters the number of vertices represented by coarse vertices, and so increases the number of repulsive forces applied to a vertex (while simultaneously reducing the number of levels in the multilevel scheme). The aim was to determine if such mechanism would prevent the warping effect of using Multilevel Global Forces and provide a more uniformly expanded layout similar to Octree generated layouts.

Analysis of the results suggests that Multimatching is able to provide such effect, with some layouts showing more equilateral expansion. However, the numerical results suggest that there is little benefit from using the techniques for many graphs, with all values increasing the number of edge crossings and running time by a minimum of 26.9% and 27.8% respectively (achieved using a matching value of 4).

The increase in edge crossings can be attributed to the change in the approximation of repulsive forces in greater number of directions, supported by the subjective analysis which shows layouts are expanded outwards and exhibit more overlaps. Although higher number of edge crossings, the layouts are still readable and comparable to those achieved using standard edge crossings, providing little benefit other than a more equilateral expansion of the layout.

Runtime is altered as a result of changing (see above) whereby the larger number of vertices represented by a vertex in a coarse graph increases the number of approximations.

Due to the higher running times and increase in edge crossings, Multimatching is not suggested for use in generating a multilevel scheme for use with Multilevel Global Force approximation.

More advanced methods may be able to identify improved matchings and therefore improve layout quality and so further investigation into matching and clustering techniques is suggested.

7.2.3.3.2 Matching Maximal and Average Degree

Analysis reveals there is little to no improvement to layout quality or running time as a result of using matching values equal to the average or maximum degree within each graph.

Such a result is surprising as it was expected that better representation of graph structure and connectivity would be indicated in layouts, however, matching the average degree increases running time and edge crossings by 108.7% and 165.3%, suggesting that the change in the number, weight and direction of approximations causes decreased readability. Subjective analysis attributes the increase in edge crossings to overlaps within layouts as seen in the Multimatching above.

More notably, matching the maximal degree (brute matching) increases running time by 1001.5%, caused by a massive increase in the number of calculations for larger graphs in the test collection. The calculations come from a high number of levels in the multilevel scheme, and higher numbers of vertices being represented by coarse vertices. Although there is a high increase in running time, edge crossings increase by a lesser 38.3%.

Similar to above, brute and degree matching are not suggested for use with Multilevel Global Force generation due to the increased running time and edge crossings. Although subjective analysis highlights similar layouts to standard, the rise in running time dampens the little benefit to be gained.

7.2.3.4 Pattern Placement

Placement of primitive structures moves from the drawing of general graphs to providing layout to specific structures, and can be considered a similar mechanism to those used in layouts provided to hierarchical graphs. Although specific, investigation here looks at the use of identifying and removing such structures in order to reduce the complexity of the Multilevel Global Force structure when drawing general graphs.

Four methods are used to provide guidance over usage and configuration, testing the removal and placement of leaf vertices using a Vertex Placement Scheme (Leaf Placement), refinement of the calculated layout (*FDP* refinement), and the impact on these changes to only the original graph (Single Level) or throughout the multilevel scheme (Multilevel):

- A – Single Level Leaf Placement
- B – Single Level Leaf Placement with *FDP* refinement
- C – Multilevel Leaf Placement only
- D – Multilevel Leaf Placement with *FDP* refinement

Results are recorded for the collection of general graphs and also the collection of leafy graphs. It is noted however, that the leafy graphs have much fewer vertices and edges than the tested general graphs.

7.2.3.5 General Graphs

For general test graphs running time is increased by 25.8% and 3.3% for single and multilevel leaf placement (A and C). The 13.1% difference between the single and multilevel usage may most noticeably be caused by the $O(L)$ and $O(L \cdot (L-1))$ reduction in calculations per iteration. The significant difference between them correlates to the number of leaf vertices per graph, many of the general graphs have no leaf vertices, whereas coarser abstractions may include many.

Although increasing running time, the layout quality remains largely unaffected by leaf placement, increasing edge crossings by 1.26% for single level (which can occur naturally between layouts). The lack of impact is due to general graphs featuring few to no leaf vertices, in contrast, the multilevel leaf placement shows an increase in 15.9% suggesting the existence of leaf vertices in coarser abstractions. The increase comes from the observed fluffier layouts as a result of finer vertices being placed in higher energy positions as a result of the leaf placement in coarse graph.

Methods using *FDP* refinement results in a more noticeable application of running time of 66% and 36% for algorithms B and D. Running time is increased due to application of further *FDP* to general graphs exhibiting leaf vertices (i.e. add32). The reduction supports the suggestion above, that coarser graphs exhibit more leaf vertices providing opportunity for further saving.

Although noticeably increasing running time, the number of edge crossings drops by 7.3% and 22.8%, suggesting initial leaf placement with refinement thereafter improves the measured quality of layouts. Observed layouts can be identified as being fluffy (like above) with fewer tangles and overlaps.

7.2.3.6 Leafy Graphs

For leafy graphs running time is decreased by 29% and 23% for single and multilevel leaf placement.

Edge crossings increase by 56.1% and 65.9% for single and leaf placement, a result of high number of leaf vertices overlapping the existing layouts. Edge crossings are also increased using multilevel placement, contrary to the general graphs above.

Leaf placement with *FDP* refinement shows an increase in running time by 15% and 21% in running time (B and D). This is different to the results for general graphs which saw a reduction for the multilevel usage and is attributed to the increase in leaf vertices, resulting in larger number of calculations in *FDP*.

Edge crossings are reduced by 12.2% and 6.7% for single and multilevel leaf placement with *FDP* refinement. Although less than the values above, the impact is still higher than exhibited for the general graphs, however, the multilevel method provides fewer edge crossings (similar to the general graph measurements).

7.2.3.7 Overall Usage

It is known through the Vertex Placement Scheme, leaf vertices are likely to be provided with poor placement relative to their local layout due to orientation using the centre of mass, often overlapping other part of the layout. In addition, the VPS does not provide the compression and elongation of edge lengths seen in Force Directed Placement, and so layouts do not necessarily fit in.

The running time features an increase or decrease, associated with the search for leaf vertices and the saving when removed from force directed placement (shown as an increase for general graphs and decrease for leafy graphs).

In general, multilevel usage appears beneficial for the drawing process, providing improved layout when refined using further Force Directed Placement. The running time associated with multilevel usage is less than single level, however still increased above that of the default edge contraction algorithm.

Single level provides some optimization for known leafy graphs only, multilevel provides some suggestion of optimization in the multilevel scheme and so where quality is more important than running time, multilevel leaf placement is suggested. For smaller leafy graphs, the leaf placement algorithm provides an effective way of highlighting the leaf vertices and can be modified to allow users to modify the placement of leaf vertices – adding a level of interactivity.

7.2.3.8 Primitive Graphs

Use of primitive graph structures as a potential stopping mechanism for multilevel generation has been shown beneficial during investigation. Compared to the default value of 2 for coarsening tolerance, there is little impact on running time (1.1% increase) and layout quality (1.7% increase in edge crossings) during numerical analysis.

Although it was anticipated some saving would be achieved in running time (as a result of the coarsening scheme finishing early), the minor rise indicates that the time spent identifying a graph as primitive negates any saving achieved by ending the coarsening scheme early.

Similarly, it was expected that layout would be effected as a result of improved initial positions and improved expansion as a result of the altered *MGF* structure, however the results suggest little difference. Subjective analysis indicates that the cause of the change is different between graphs, with reduced edge crossings achieved by improving global layout, and in others increased as a result of additional directions of approximation.

Layouts are observed to be fluffier and show less warping due to the layout/direction of forces in coarser graphs. As such, primitive graphs are suggested as a stopping mechanism and are included as part of the default algorithm.

7.2.4 Summary of *MGF*

The most notable feature of *MGF* is the decreased running time achieved with minimal change to layout quality. Layouts are comparable to those described in Walshaw (2003) and Hu (2005), however exhibit warping and more noticeable Peripheral Effect due to the change in the direction and weight of repulsive force approximation. Although it is believed approximation better represents the structure and connectivity of a layout, determining the impact on readability requires studies measuring individuals' perceptions of a layout.

Implementation requires only reuse of existing multilevel structures, using a similar traversal scheme as that used for the Octree for application of coarse vertex positions as approximation, making the scheme easy to implement and transferable with the Octree method. A key difference between the two is the use of graph structure as opposed to space, this frees approximation from requiring an additional structure with which to map vertices.

Although not implemented, it is expected that other multilevel and multiscale methods can utilize the approach (Hadany and Harel, 2001, Harel and Koren, 2001, and, Hachel and Jünger, 2001) to generate their own approximation structure – specific to the multilevel refinement used.

Default usage combines Multilevel Global Force with the Fruchterman-Reingold Spring Embedder (using standard parameters seen in research, i.e. $C=0.2$). The generation of the multilevel/*MGF* structure is advised to include Primitive Graph as a method for preventing similar abstractions (few vertices different) being generated and providing an accurate and calculated layout for the coarsest graph.

If reduced running time is a priority, approximation limit is suggested to prevent only 15 calculations of repulsive force per vertex (that is, traversal of 15 levels in the *MGF* tree), providing reduced running time and slight increase in edge crossings. Similarly, a spatial limitation on the distance of repulsive forces can be used with a value of 14, providing similar results.

In contrast, if high quality layout is considered priority, it is suggested that placement is provided using Vertex Placement Scheme in addition to multilevel schemes and *FDP* refinement (Algorithm D). For more interactive modes, vertex placement is suggested on its own, allowing users to alter the placement of leaf vertices – if applicable.

7.3 Dynamic Graph Drawing Methods

The aim of investigating dynamic drawing methods here is to extend the approaches used in static drawing, and provide visualisation of large dynamic graphs. In order to accomplish this, an adaptation of the Fruchterman Reingold Spring Embedder is suggested for use of the strong force functions used to overcome minima in dynamic drawing. Comparison is made to the Eades Spring Embedder to determine the effectiveness.

Following the adaptation, the multilevel and approximation methods become easier to transfer, and are compared in order to identify their effectiveness at improving global layout and reducing running times. The introduction of such mechanisms, particularly the use of a multilevel scheme, introduces the need for methods to update matchings representing graph structures as amendments are made.

7.3.1 Dynamic Spring Embedder

An adaptation of the spring embedder method described by Fruchterman and Reingold (1991), altering the method of calculating displacement (and corresponding equilibrium of forces) to provide gradual vertex movement similar to that seen by the Eades Spring Embedder (Eades, 1984).

7.3.1.1 Configuration of Displacement

Displacement by spring and repulsive forces is calculated separately, allowing for the equilibrium of forces to be modified such that one force has priority. More specifically, it provides a means to balance the displacement such that vertex movement is gradual as opposed to large distances observed in static drawing.

The experimentation shows that by limiting vertex displacement from repulsive forces, while keeping the displacement of spring forces unchanged, movement is increased or decreased providing freedom to overcome minima or more gradual vertex movement. Although a default value of 0.7 is suggested (0.7:1.0 ration between repulsive and spring displacement), the nature of dynamic graph drawing (an ongoing process) allows for values to be changed during the drawing process to best suit a dataset.

The default parameter value of 0.7 minimises movement providing smooth vertex trajectories, while retaining the force required to overcome minima (providing layouts with minimal edge crossings. Due to differences in connectivity, different graph types provide different performance between values (see Appendix 10.11).

7.3.1.2 Comparison of Spring Embedders

Cooling schedules gradually limit vertex movement per iteration of Force Directed Placement, providing a layout (optimal or not) within a finite number of frames, after which vertex movement is reduced so much it appears to stop entirely. In contrast, methods which do not use cooling schedules allow for an infinite number of frames, allowing for layout to continue being developed after the cooled algorithms have stopped, at the cost of having reduced movement required to overcome minima quickly.

The proposed Dynamic Spring Embedder aims to improve upon this by using force functions described by Fruchterman and Reingold to better overcome minima and find layouts with minimal edge crossings quickly. Configuration of displacement can be altered to enhance

this effect and better overcome minima, but at a cost of vertex movement being increased (and so harder to follow).

Although reducing edge crossings and better overcoming minima in comparison to an implementation of the Eades Spring Embedder, layouts generated using the Dynamic Spring Embedder exhibit high range in edge lengths (Peripheral Effect, Hu, 2005). Further to this, the observed layout shows large amounts of movement initially, untangling the layout and removing edge crossings but making such changes difficult to follow. The improvement of layout from randomised vertex placement is believed to represent a dramatic layout adjustment, identifying an algorithms ability to incorporate graph amendments into layout (which are not specific to shrink or growth). The results suggest the Dynamic Spring Embedder will react quickly to changes, correcting layout as adjustments are made. This may better represent the impact of amendments on a graph, but it is expected the change will not preserve layout.

Due to the resulting high quality results and the ability to overcome minima, the Dynamic Spring Embedder is considered useful for dynamic graph drawing, particularly for the untangling of large graphs. Although promising, it is noted layouts can appear stretched and warped as a result of strong forces at the centre of the graph (Peripheral Effect). In addition the results only cover layout generation, further investigation into graph and layout adjustment is evaluated below.

7.3.1.3 Layout Adjustment

Following layout generation, adjustment is tested by amending the graph and measuring the resulting changes to the quality of the layout (edge crossings and average vertex movement). No known collection of dynamic graphs exist and therefore example datasets are provided, featuring three structural types (differing connectivity) with amendments to reduce or continue the structural attributes of each (for example, extending an 10x10 grid square to a 10x20 grid rectangle).

The results indicate similar performance as indicated for layout generation above, with the Dynamic Spring Embedder being better able to overcome minima (represented as spikes in edge crossings as amendments are performed followed by an immediate drop to some minimum as low energy positions are found). Comparison to the Eades Spring Embedder similarly indicates an improvement in overcoming minima.

As mentioned above however, the ability to overcome minima results in large amounts of movement observed during subjective analysis. Although remaining easy for the author to follow what is occurring in the graph, the difference to Eades Spring Embedder (*SE*) is noticeable, with a larger amount of movement in reaction to changes as vertices move into lower energy positions. The quantitative analysis of movement suggests relative movement is higher for Eades Spring Embedder, providing a false suggestion that movement is greater.

As such, it is believed that the Dynamic Spring Embedder (*FRD*) method better incorporates amendments into the layout as a result of overcoming any minima that is introduced, whereas the *SE* method better preserves the initial layout of a graph. However the results are only accurate for the types of operation performed and provide only a general behaviour. Although frequency and types of operation are investigated (see Section 6.5), it is not possible to emulate every possible combination of operation and so the detailed results should be used only as a guide.

7.3.2 Multilevel Generation

The analysis of the metrics for graph stability shows little improvement in layout as result of multilevel refinement during generation of layouts for the test graphs. Investigation reveals that the size of a graph impacts the incorporation of multilevel layout with higher numbers of edge crossings measured in smaller graphs, observed as warping within the layout as multilevel layout overpowers local refinement.

Such differences result in poorer measured layout quality, whereby the Dynamic Spring Embedder is able to provide adequate layout without multilevel refinement. In contrast, its use in larger graphs, which would otherwise remain tangled, has improved measured layout as a result of improved global layout, in addition to exhibiting little to no warping.

Subjective analysis supports this with improvement of layout in larger graphs and warping of layouts in smaller graphs. Observing the application of force directed placement reveals that the improvement in edge crossings comes from related vertices being moved into lower energy global positions, after which the local layouts are refined.

Due to the increase in edge crossings in smaller graphs, multilevel layout is suggested for only medium and above graph sizes (>1000 vertices). Displacement appears to be a viable

method of incorporating global layout into a graph in contrast to methods utilising springs between levels (Veldhuizen).

It is noted however, that the global layout provided may not be optimal due to folds still prevalent in the layout. This suggests that although displacement provides some global layout, a minimum energy global layout may not be found with folds being exhibited in layouts (for example, regular-large in Figure 6.29).

Methods for incorporating multilevel layout already exist within dynamic drawing (Veldhuizen and Frishman), however little discussion is provided regarding the limitations of the methods used. The results here identify a simple technique for incorporating global layout into dynamic layout generation, showing improvement in layouts for large graphs and the limitations encountered.

7.3.3 Multilevel Global Force

Multilevel Global Force aims to provide reduced running time, similar to the reductions seen through its usage in static graph drawing. As expected, a massive decrease in running time is observed, from 95% reduction for small graphs to 99.8% for large graphs in comparison to the Dynamic Spring Embedder (n-body calculation). Other approximation methods are likely to provide similar savings to this, for example the Octree space decomposition, however the results of Section 5.2 suggest further reduction in running time can be achieved using Multilevel Global Force.

The saving in running time also identifies a requirement for minimum and maximum frame rate for visualisation of the drawing process as an animation. Running time is decreased to such an extent that the FPS exceeds that for typical animation, identifying a need to incorporate delays between frames in order to prevent accelerated graph drawing.

As a result of approximation, edge crossings are increased in all layouts – most notably 462% for smaller graphs and 133% for medium graphs. The cause of such massive increase comes from differences in the calculated repulsive force displacement. Altering the default values of equilibrium of displacement reduces these edge crossings providing layouts similar to those achieved by Dynamic Spring Embedder (see Appendix 10.24).

Although higher edge crossings are reported across all graphs, the change in running time is far more advantageous than the increase in edge crossings (1 frame of Dynamic Spring

Embedder takes the same time as 631 frames using the same method with Multilevel Global Force), allowing for large graphs to be visualised in a fraction of the time (observed as a faster convergence of layout).

7.3.4 Dynamic Matching

Dynamic matching investigates the effect of altering/updating multilevel matching used in the multilevel layout refinement and approximation methods above.

Due to the large number of variables within the experimentation (such as operation types, graph structures and graph size), results for the impact on multilevel layout and approximation are analysed in respect to the update methods and operation types only. Additional variables are summarised to highlight their impact on results.

7.3.4.1 Graph Structure

The graph structure (average degree and repeated sub-graph structures) is shown to effect the impact of operations and update methods. More so due to operations sets being dependant on the structure – continuing or reducing the connectivity (degree) within the graph. For example, sparse graph has the smallest average degree of all the test graphs, making it easier for vertices to pass one another and overcome folds (minima) by reducing the number of edges and vertices it must move past.

It is noted the results are generalised across the graph types, however different structures may have a different impact and so not follow the general behaviour (similar to being unable to emulate all types of operation, it is infeasible to emulate every possible graph).

7.3.4.2 Frequency of Amendments

The frequency of operations is the speed at which the changes occur relative to the number of frames being generated, with high frequency referring to more amendments being made per frame, expected to require longer for a layout to be corrected. The results follow such expectation, with higher frequency causing operations to stack over time as force directed placement is unable to amend layout before additional operations occur. Such effect can be noticed in all sizes of graphs and types of operations, and escalates with graph size due to a larger layout to be corrected.

7.3.4.3 Fragmentation

Fragmentation refers to alterations made to the multilevel scheme which leaves vertices unmatched, the result of which is multiple vertices existing within coarser graphs which represent few vertices in the finer graph (in contrast to other vertices in the coarse graph representing significantly higher numbers in the finer graph). The impact of this is significant forces being applied to these “light weight” vertices, pushing them far from the centre of the graph and causing warping in the layouts.

Fragmentation is caused by amendments introducing or removing vertices, and update methods which are unable to match them within the multilevel scheme (due to a full matching already being made). As the number of fragmented vertices increases, theoretically the impact would become less as the odds of anchoring vertices being matched decreases, allowing for new matches to be made, however are not observed in the results.

The effect of fragmentation is different between operation types, for shrink operations, the reduced weight and connectivity of vertices in coarser graphs causes higher numbers of edge crossings as more minima opens up. In contrast, growth operations show reduced number of edge crossings as a result of improvement global placement of vertices (light weight vertices are easier to move through minima).

Fragmentation is observed in both multilevel layout refinement and multilevel global force approximation.

7.3.5 Multilevel Updates

Multilevel refinement is used for generating global layout, the update of which is expected to incorporate graph amendments within the layout by differing extents.

Of all the update methods, the rematch method best preserves the layout between operations. Most notably, the rematch method reduces the fragmentation within the multilevel scheme (unmatched vertices with low weight in coarser graphs), unlike that seen in mid and high level updates.

In contrast, the single-level update causes the opposite effect, whereby shrink operations reduce edge crossings as layout is preserved, but increase edge crossings as growth

operations expand the graph, unable to provide good global layout. Mid-level update generally performs between single and full update methods.

Movement throughout all methods shows little difference (<5%) to movement, however, is relative to initial movement. More notably, sparse graphs shows erratic movement and changes through all update methods and operation types.

Subjective analysis confirms the suggestions above, with single preservation of layout but poor adjustment of new vertices and edges and warping in layouts using mid and high updates in regular structures as a result of fragmentation and placement of vertices in coarse graphs.

Layouts for single level and rematch are quite similar (regarding spread of vertices) suggesting both provide accurate representation of the layout. However, rematch provides improved layout for growth operations.

7.3.5.1 Multilevel Global Force Updates

Multilevel Global Force approximates the positions of vertices, the extent of which is dependent on the distance as described by a multilevel scheme. The update of the approximation is necessary in order to retain an accurate representation of the graph as amendments occur, providing calculation of forces which incorporates the amendments immediately.

Unlike above, both shrink and growth operations are impacted by update methods. For shrink operations, rematch provides least change in edge crossings and single generates highest, whereas for growth operations, mid shows least increase and rematch shows highest.

The cause of the more noticeable impact than using multilevel layout is due to the approximation in repulsive forces. Multilevel refinement provides guidance for global layout but refinement is still local, however, updates for *MGF* are incorporated directly into local and so updates impact the layout much more.

Most noticeably, the rematch method provides least change in edge crossings for shrink and greatest change in growth, caused by good preservation of layout during vertex removal but

larger incorporation of changes in growth (resulting in noticeable changes in edge crossings as different minima is identified).

Movement is affected due to change in approximation (weights and direction), with high level updates providing increased movement for all operations. Mid-level update provides similar movement from shrink operations but not for growth, and single and rematch methods remain with least changes to movement. Initially it is believed that the movement is a result of updating the matching, however, rematch does not provide such movement, and so the cause is thought to stem from fragmentation.

Subjective analysis somewhat confirms this, with warping (similar to that seen in Multilevel updates) observed for mid-level and full updates. In contrast, single level updates do not cause such warping, albeit layout is not achieved for the introduced vertices.



Figure 7.1 Subtle warping of the graph 3025 as a result of regenerating the multilevel scheme for 're-matching' updates.

Similarly, rematch has a reduced warping effect, but movement is noticed as edges expand and contract in reaction to changes in the multilevel matching and the centre of mass of approximations. The effect is more noticeable for dense graphs whereby large movement are observed when new matchings caused radical changes to the centre of mass used in approximations, a result of the high connectivity in the graph generating an entirely new matching.

Small graphs appear more distorted by *MGF* as a result of the approximation of forces into two directions, and large graphs (and consequently large numbers of operations) show slow impact on layout.

Due to the variety in behaviours, changes should be incorporated using update methods best suited for the graph type. For dense graphs, full-level update is suggested to incorporate

changes and preserve layout (by preserving the matching), whereas for regular graphs the rematch method reduces fragmentation and resulting warping of the layout.

7.4 Summary

Some key points are identified:

Experimental Methodology

- The methodology aims to infer performance by measuring the test data and metrics in comparison to implementation of existing algorithms
- Implementation provides use of similar structures and algorithm design in the same environment, providing a true comparison
- Implementation only follows the interpreted design described in respective publications – the authors of each publication may therefore not have included parts essential for achieving the results published
- Collection and analysis of the results is standard, comparing average performance of multiple repeats

Test Data and Metrics

- Test data is only used as an example – static graph collections used previously
- Few large dynamic graphs exist therefore an example collection is used with varying connectivity (degree)
- Edge crossings are shown to be good indicator of the planarity of layouts, which can then be compared across algorithms
 - Although not necessarily identifying poor layouts, it provides a means of identifying similarity of layouts for a given graph
- Metrics for graph stability has been useful in mapping change within a layout as a result of amendments or layout development
 - Comparison to subjective shows that it does not necessarily encompass all aspects of what is happening – specifically the cause of movement in the layout and whether it can be followed

Multilevel Analysis

- Possible use for monitoring development of global layout in dynamic graphs or comparing the global layout of two algorithms
- Beneficial for numerical comparison but similar conclusions can be made through subjective analysis
- Experimentation shows that the results of local analysis is comparable to global analysis, and so similar conclusions regarding a layout can be made without multilevel analysis
 - Analysis of individual partitions provide identify good local layout but similarly, conclusion regarding the final graph may provide similar insight

Multilevel Global Force Summary

- *MGF* reduces running time and provides high quality layouts comparable to those generated by current methods (Walshaw and Hu)
- Changing structure of multilevel scheme (Multimatching and brute matching) offers little benefit – highlighting power of weighted matching
- Primitive processing is beneficial to graphs which feature leaves but requires additional running time and offers a useful method for ending matching schemes before abstractions become too similar

Dynamic Graph Drawing

- Dynamic Spring Embedder offers a way to visualise graphs and provide high quality results which can better overcome local minima than the Eades Spring Embedder
- Other methods exist which incorporate additional features (i.e. magnetised springs or intelligent placement) or utilise methods which intend to preserve a user's initial perception of a layout (multilevel dynamic drawing)
- Multilevel method useful for improving global layout in large graphs as seen in static drawing
- *MGF* useful for improving running time
- Dynamic matching provides some guidelines for multilevel updates, generally, stick to single or rematch in order to avoid fragmentation of the matching (which results in warping as forces and vertex placement is altered)

In summary of the evaluation, it is clear that there is scope for further experimentation – particularly in the analysis of dynamic drawing algorithms which requires additional comparison to other methodologies, and a wider range in test graphs. Similarly, the experimentation in graph amendments and impact on multilevel schemes requires additional investigation.

8 Conclusions

This dissertation has investigated two notable areas, identifying a new use for multilevel schemes to approximate long range repulsive forces using structure as opposed to spatial information, and an adaptation of the Fruchterman Reingold Spring-Embedder for iterative dynamic drawing, providing smooth vertex trajectories are force functions able to better overcome local minima to provide layouts with minimal edge crossings. Brief investigation is also provided to identify metrics for monitoring development of global layout in larger graphs.

The aims of the investigations were to assess methods for:

- An efficient force calculation offering minimal cost, intending to reduce the running time associated with spring embedder (Multilevel Global Force)
- Optimization of layout visualization and adjustment for large graphs such that changes are visualized immediately (Dynamic Graph Drawing)
- Global layout and local layout generation simultaneously via multilevel integration (Dynamic Graph Drawing, Multilevel Layout)

8.1 Research Findings

8.1.1 Implementation

Being able to compare research through its implementation was extremely useful, both in terms of learning and of identifying progression in the development of the theory.

The shared environment provides confidence that the suggested algorithmic approaches are not just theoretical or dependent on the one implementation.

It is strongly suggested any new researchers to the area look at implementing the works of others, if even through paper-based means, to get a better understanding of the works (often leading to questions as why something is at it is).

8.1.3 Algorithm Performance

Algorithm	Description
Multilevel Global Force Spring Embedder	The algorithm works best for general graph drawing across all test cases. The algorithm outperforms the implementation of Multilevel Spring Embedder and Efficient Multilevel Spring Embedder in terms of running time with minimal cost to layout quality.
Multilevel Global Force Spring Embedder with Multimatching	There was little benefit to using Multimatching in any of its forms, and the results did not follow any predictable pattern which could be exploited.
Multilevel Global Force Spring Embedder with Leaf Placement Scheme	The algorithm was most useful during interactive drawing, whereby the placement schemes could be modified in real time to ‘redraw’ the affected vertices. However, there was little benefit generally.
Multilevel Global Force Spring Embedder with Primitive Matching	There was little benefit to primitive matching, with negligible improvements to running time or layout quality.
Multilevel Global Force Spring Embedder with Approximation Limit	Useful to reduce running times, however the negative impact on layout quality makes it useful only for cases when running time is of high importance.
Dynamic Modified Spring Embedder	The algorithm works well for smaller graphs and outperforms the Spring Embedder when overcoming minima. Best used for general drawing purposes.
Multilevel Dynamic Modified Spring Embedder	The algorithm improves the quality of layouts for larger graphs only, layouts for smaller graphs lose quality as multilevel layout overpowers the local placement.
Dynamic Spring Embedder with Multilevel Global Force	Best in terms of reducing complexity with minimal cost to layout quality, however like above, best suited for larger graphs as the approximation/multilevel overpowers the local layout of smaller graphs.

<p>Multilevel Spring Embedder with Dynamic Matching and Multilevel Global Force</p> <p>Dynamic Spring Embedder with Dynamic Matching</p>	<p>The usage of dynamic matching is dependent on the requirements of the user. Low level updates are useful for preserving the layout of the graph, whereas high level updates and rematching are useful for showing the impact of changes to the global layout.</p> <p>It is suggested high level updates are used by default, in order to avoid fragmentation of the multilevel scheme and maintaining accurate approximations. Rematch is useful for quick changes, and easier to implement, however the changes do lead to changing layout as the matchings impact global layout and approximated forces.</p>
--	---

8.1.4 Multilevel Global Force

Multilevel Global Force provides efficient force calculation reducing the cost required of approximation, providing means of generating layouts for larger graphs in reduced time, which better encapsulates the structure and connectivity of the graph as approximated by a multilevel scheme. Evaluation of Multilevel Global Force (Section 7.2) identifies a considerable drop in running time (40%) over Octree approximation, and both improvement and deterioration to layout quality (20%). Layouts are overall more consistent and isomorphic. The application of the approximation method to dynamic graph drawing is evaluated in Section 7.3.3 describing large savings in running time (99% over n-body calculations).

Further investigation in Section 7.2.3.3 found little improvement in optimising the multilevel structure using Multimatching methods aiming to provide coarser approximation of graph structure and layout, suggesting the standard weighted edge contraction

Further investigation in Section 7.2.3.4 supports usage of pattern placement techniques for graphs exhibiting leaf vertices, providing layouts with fewer edge crossings (improving readability) and the option of personalised placement of vertices. Building upon this, it is shown that identifying primitive graphs can prevent similar abstractions being generated

within the multilevel scheme, stopping the coarsening early and providing an initial calculated layout.

Overall, experimentation in Section demonstrates that Multilevel Global Force approximation has a beneficial effect on the efficiency of graph drawing, providing comparably high quality layouts to the state of the art methods in reduced time.

8.1.5 Dynamic Graph Drawing

The proposed modifications for a dynamic spring embedder evaluated in Section 7.3 provides visualisation of the development and amendment of layouts comparable to the Eades Spring Embedder. The modification of the Fruchterman Reingold Spring Embedder (Section 3.7) extends the force functions typically used in static graph drawing to dynamic graph drawing, allowing for minima to be better overcome and generate optimal layouts, reducing edge crossings in the generated layouts (Section 7.3).

Configuration of the proposed algorithm is investigated providing a default configuration and the ability to modify the forces at play in real time, generating different and more meaningful layouts in by altering values (for example, increasing vertex movement to overcome tangles in layouts), evaluated in Section 7.3.1.1 .

Building upon this, multilevel layout refinement is applied providing improved global layout to large graphs (Section 7.3.2). Although previously applied using cooling schemes and complex multilevel spring functions (Veldhuizen 2007), the algorithm provides a simple method to refine and interpolate global and local layouts simultaneously.

Further use of multilevel schemes for approximation using Multilevel Global Force is applied and evaluated in Section 7.3.3 , identifying a reduced cost of (running time) of calculating repulsive forces, and allowing for larger graphs to be drawn in reduced time, providing a faster frame rate for animating the drawing of large graphs over standard spring embedders. It is noted that amendments to the configuration of forces is required to gain improved layouts over the Dynamic Spring Embedder alone.

Experimentation of the dynamic graph drawing algorithms (Section 7.3) demonstrates the effectiveness of incorporating multilevel layout techniques into medium and large graphs for improving global layout quality, and huge reductions in running time through use of approximation, offering visualisation of much larger graphs.

8.1.6 Dynamic Matching

Dynamic matching identifies some of the problems associated with updating the multilevel matching/sparsification (used in multilevel refinement and approximation) with amendments to a graph as evaluated in Section 7.3.4 . The research shows the importance of reducing fragmentation and retaining the approximate structure of the graph, ensuring vertices in coarser graphs are provided position and weight accurate enough to approximate layout can be provided and interpolated.

The investigation shows that update methods have high impact on multilevel global force approximation (Section 7.3.5.1), a result of the graph amendments being applied to force calculations. In contrast, the impact of updates on multilevel refinement has a lesser effect (Section 7.3.5), with operations showing little effect to graph layout across the update methods.

A common behaviour across the multilevel update methods is the introduction of fragmentation which causes warping in layouts (in both layout refinement and approximation). The frequency of updates reduces this to a large extent, as noted by layouts of large graphs, whereby fragmentation is minimised due to the vast number of calculations over short periods, however small and medium graphs are impacted more due to the relative sizes of the graphs to the multilevel scheme.

Overall the experimentation shows that incorporating graph changes into multilevel layout and approximation does change the layout in comparison to update methods which do not take this into account, offering a more global interpretation of the data.

8.1.7 Metrics for Graph Aesthetics and Graph Stability

Measuring the number of edge crossings within a graph layout is a useful mechanism for gathering numerical representation of both layout quality and the regularity of drawing algorithms for providing consistent layouts. The results are (from the authors' subjective view) a good representation of the layouts general readability within a 2D drawing area.

Analysis of edge lengths provides a useful insight into the stretching (peripheral effect) within a layout, however were lesser used for comparison of algorithms as strict uniformity of edge lengths played a less significant part in the authors reading of larger graphs due to

the sheer number of edges (stretching and compression actually made some graphs easier to read as it highlighted the global structure of the graph).

Using edge crossings and vertex movement as a means of determining the development of a graph layout was very useful, and allowed for a quick and numerical comparison of drawing algorithms. Even simple charting of the values over time was useful, giving a good sense of feedback to any changes made to the algorithms during their development without needing to analyse small multiples (frames) or the animation of the drawing itself (please note, numerical analysis is not suggested to replace visual analysis but instead should be used in addition too, especially in the case of comparisons).

In addition to the test data itself, the methods for testing the metrics for graph stability showed to be invaluable in learning the behaviours of different drawing algorithms. In particular, the use of Layout Generation as extreme layout adjustment gave a window to monitor the drawing processes without distractions such as the type and frequency of operations.

8.2 Research Contributions

The research contributions include a method for reducing the complexity of force calculation in n-body systems (Section 3.6), investigation of a multilevel dynamic graph drawing algorithm (Section 3.7) and the incorporation of graph changes into a multilevel scheme representing the global structures of a graph (Section 3.7.3). Although only used on spring embedder examples, it can be adapted to any system which models particles with n-body interaction through some relationship to one another.

The most immediate impact is the ability to generate layouts for larger graphs using more structural information which better represents graph structure (Section 7.2), however, the suggested method for approximation global forces may be applied to other areas. Due to the similarity with the Octree, Multilevel Global Force can be adapted for application to other areas which work with particle systems, however the method requires some relationship the particles with which to generate the approximation structure.

A spring embedder is proposed to improve upon dynamic layouts for larger graphs (Section 7.3.1). Typically only applicable to smaller graphs, the approach allows for larger

collections of data to be visualised and allows for improved visualisation in a world whereby the volume of data is getting bigger. Multilevel layout and Multilevel Global Force improve upon this by allowing for larger graphs to be provided with improved layout (Sections 7.3.2 and 7.3.3 respectively).

The implication of this is the ability to visualise and explore larger collections of data in reduced time, providing refined global and local layout for improved readability of layouts and better representation of the data.

A more practical contribution of this thesis is the development of an experimentation framework, moreover, highlighting the importance of making comparisons between algorithms which are implemented on the same environments using as many common variables as possible. By implementing other people's works, and breaking them into smaller specialist components (individual tasks, for example, contraction of edges within the multilevel scheme) it allows for various configurations and an ease of experimentation. It also helps authors better understand one another's code, and will hopefully bring more discussion and collaboration between those who attempt to implement one another's work, making for better designed algorithms.

8.3 Limitations

A number of limitations need to be considered.

Firstly, the results collected throughout this thesis relate to attributes of layouts expected to relate to their readability (as identified in literature, Section 2.4), as such analysis is largely dependent on those attributes and if they are found to be ineffective for representing readability (Section 7.1), the results become void. Individuals have different perceptions of what makes graphs and data readable, and so the collected results may not accurately represent the readability for everyone.

In many cases, results are generalised to provide performance across all graphs. As such, some result sets may differ or not follow this.

The dynamic modified spring embedder is only compared to a basic spring embedder (Section 7.3.1.2), despite adaptations existing which aim to better preserve graph stability or improve initial positions of vertices, or aim to spread impact amendments across a graph.

Comparisons across a collection of algorithms can provide better impact to the research area but encounters problems with differences in testing environments and interpretations of the algorithms described.

Dynamic graph drawing is limited by a lack of large mainstream dynamic graphs (Section 2.5), in contrast to the large number found in static graph drawing. The generation of data sets limits the experimentation and analysis of results to “emulated” graphs, whereby amendments aim only to reduce or extend the graph.

8.4 Future Work

The new algorithmic approaches described and experimented in this thesis provide only a small window of investigation, with many avenues which can be continued.

As far as the author observes, the most predominant problems in the area are related to an uncertainty as to what makes a layout “good”. Research into graph aesthetics gives us a good indication, and we can use this to compare our algorithms and the layouts generated, however it relies on these initial definitions of “good layout” and does not include the context of the data to a reader.

Further investigation on how people read graphs and which aesthetics improve their experience with the visualisation is always welcome, particularly those testing on real people.

In addition to the difficulty of defining good layout quality, dynamic graph drawing of large graphs is such a new area that there is little to no research which describes how dynamic graphs with more than 10,000 vertices should be drawn over time. The efforts in this thesis, and the works of few others (Veldhuizen 2007) aid the drawing of such dynamic graphs, but there are no studies which monitor how people view this data, and the patterns or movements which maximises the graph stability (in contrast, readability of smaller dynamic graphs have been more thoroughly investigated). Some suggestions are made in the Evaluations here, but there is more room for work.

In terms of algorithm design and performance, there are many avenues for additional research in Multilevel Global Forces, or even other uses for multilevel scheme as an approximation of distances between vertices.

Some ideas which directly relate to this thesis include:

- Multilevel Aesthetic Analysis - as mentioned previously (Section 3.5), the approach is heavily influenced by the aesthetic attributes used to measure the layouts, and as such the development of matching techniques specific to analysis of desired attributes is proposed in order to measure approximate and partition layout quality for different aesthetic needs – providing means of optimising this in drawing algorithms.
- Comparison of Multilevel and Multiscale Schemes for generating a Multilevel Global Force Approximation – generation of the approximation scheme using other multilevel and multiscale structures and identify which of those provide improved layout readability or greater efficiency saving (regarding Section 3.6.2)
- Comparison of Dynamic Spring Embedders - further investigation to establish the advantages and disadvantages of differing dynamic drawing methods, most notably those methods used in Dynavis and other graph drawing packages and those which optimise for aesthetic criteria
- Investigation of large graph layouts – the experimentation highlights a need for more investigation into what makes “good” layout for larger graphs, and how the readability of layouts can be compared (with and without testing through crowd based studies)

8.5 Concluding Paragraph

The algorithms described and implemented in this thesis investigate a wide variety of methods for utilising multilevel schemes with simple heuristics to vastly improve the running time of current state of the art algorithms (in some cases showing 40% decrease) and also the measured readability of the layouts generated by Force Directed Placement algorithms, suggesting a route for bringing multiscale force directed placement algorithms to dynamic graph drawing. If applied, these may prompt thoughts of reuse in other problems being solved using multiscale techniques, offering insight into the substructures the techniques are creating and how they can be used to improve the solution.

9 References

- Abello, James, Stephen G. Kobourov, and Roman Yusufov. "Visualizing large graphs with compound-fisheye views and treemaps." In *Graph Drawing*, pp. 431-441. Springer Berlin Heidelberg, 2005.
- Alberts, David, Giuseppe Cattaneo, Giuseppe F. Italiano, Umberto Nanni, and C. Zaroliagis. "A software library of dynamic graph algorithms." In *Proc. Workshop on Algorithms and Experiments*, pp. 129-136. 1998.
- Alstrup, Stephen, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. "Maintaining information in fully dynamic trees with top trees." *ACM Transactions on Algorithms (TALG)* 1, no. 2 (2005): 243-264.
- Archambault, Daniel, and Helen C. Purchase. "The mental map and memorability in dynamic graphs." In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*, pp. 89-96. IEEE, 2012.
- Archambault, Daniel, Helen Purchase, and Bruno Pinaud. "Animation, small multiples, and the effect of mental map preservation in dynamic graphs." *Visualization and Computer Graphics, IEEE Transactions on* 17, no. 4 (2011): 539-552.
- Archambault, Daniel, Helen C. Purchase, and Bruno Pinaud. "Difference map readability for dynamic graphs." In *Graph drawing*, pp. 50-61. Springer Berlin Heidelberg, 2011.
- Bannister, Michael J., David Eppstein, Michael T. Goodrich, and Lowell Trott. "Force-Directed graph drawing using social gravity and scaling." In *Graph Drawing*, pp. 414-425. Springer Berlin Heidelberg, 2013.
- Barnes, Josh, and Piet Hut. "A hierarchical $O(N \log N)$ force-calculation algorithm." (1986): 446-449.
- Battista, Giuseppe Di, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. "Algorithms for drawing graphs: an annotated bibliography." *Computational Geometry* 4, no. 5 (1994): 235-282.
- Baur, Michael, and Thomas Schank. *Dynamic graph drawing in visone*. Univ., Fak. für Informatik, 2008.
- Bennett, Chris, Jody Ryall, Leo Spalteholz, and Amy Gooch. "The Aesthetics of Graph Visualization." In *Computational Aesthetics*, pp. 57-64. 2007.
- Böhringer, Karl-Friedrich, and Frances Newbery Paulisch. "Using constraints to achieve stability in automatic graph layout algorithms." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 43-51. ACM, 1990.
- Brandes, Ulrik. "Layout of Graph Visualizations." PhD diss., PhD thesis, University of Konstanz, 1999. See <http://www.ub.uni-konstanz/kops/volltexte/1999/255>, 1999.

- Brandes, Ulrik, and Dorothea Wagner. "A Bayesian paradigm for dynamic graph layout." In *Graph Drawing*, pp. 236-247. Springer Berlin Heidelberg, 1997.
- Bridgeman, Stina S., and Roberto Tamassia. "A user study in similarity measures for graph drawing." *J. Graph Algorithms Appl.* 6, no. 3 (2002): 225-254.
- Chiang, Y-J., and Roberto Tamassia. "Dynamic algorithms in computational geometry." *Proceedings of the IEEE* 80, no. 9 (1992): 1412-1434.
- Cohen, Jonathan D. "Drawing graphs to convey proximity: an incremental arrangement method." *ACM Transactions on Computer-Human Interaction (TOCHI)* 4, no. 3 (1997): 197-229.
- Cohen, Robert F., Giuseppe Di Battista, Roberto Tamassia, Ioannis G. Tollis, and Paola Bertolazzi. "A framework for dynamic graph drawing." In *Proceedings of the eighth annual symposium on Computational geometry*, pp. 261-270. ACM, 1992.
- Crawford, Carl, Chris Walshaw, and Alan Soper. "A multilevel force-directed graph drawing algorithm using multilevel global force approximation." *Information Visualisation (IV), 2012 16th International Conference on*. IEEE, 2012.
- Cruz, Isabel F., and Joseph P. Twarog. "3d graph drawing with simulated annealing." In *Graph Drawing*, pp. 162-165. Springer Berlin Heidelberg, 1996.
- Daniel Danko, "3elt 3D graph layout on GPU." Youtube video, published 27th April 2013. <http://www.youtube.com/watch?v=15eFkE-rVVk>.
- Davidson, Ron, and David Harel. "Drawing graphs nicely using simulated annealing." *ACM Transactions on Graphics (TOG)* 15, no. 4 (1996): 301-331.
- Diehl, Stephan, and Carsten Görg. "Graphs, they are changing." In *Graph Drawing*, pp. 23-31. Springer Berlin Heidelberg, 2002.
- Dwyer, Tim, Bongshin Lee, Danyel Fisher, Kori Inkpen Quinn, Petra Isenberg, George Robertson, and Chris North. "A comparison of user-generated and automatic graph layouts." *Visualization and Computer Graphics, IEEE Transactions on* 15, no. 6 (2009): 961-968.
- Eades, Peter. "A heuristics for graph drawing." *Congressus numerantium* 42 (1984): 146-160.
- Eades, Peter, Michael E. Houle, and Richard Webber. "Finding the best viewpoints for three-dimensional graph drawings." In *Graph Drawing*, pp. 87-98. Springer Berlin Heidelberg, 1997.
- Eades, Peter, and Mao Lin Huang. "Navigating clustered graphs using force-directed methods." *J. Graph Algorithms Appl.* 4, no. 3 (2000): 157-181.

- Eisner, Jason, Michael Kornbluh, Gordon Woodhull, Raymond Buse, Samuel Huang, Constantinos Michael, and George Shafer. "Visual navigation through large directed graphs and hypergraphs." *Proc. of IEEE InfoVis, Poster/Demo Session* 116 (2006).
- Ellson, John, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. "Graphviz and dynagraph—static and dynamic graph drawing tools." In *Graph drawing software*, pp. 127-148. Springer Berlin Heidelberg, 2004.
- Frick, Arne, Andreas Ludwig, and Heiko Mehldau. "A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration)." In *Graph Drawing*, pp. 388-403. Springer Berlin Heidelberg, 1995.
- Friedrich, Carsten, and Peter Eades. "Graph drawing in motion." *J. Graph Algorithms Appl.* 6, no. 3 (2002): 353-370.
- Frishman, Yaniv, and Ayellet Tal. "Multi-level graph layout on the GPU." *Visualization and Computer Graphics, IEEE Transactions on* 13, no. 6 (2007): 1310-1319.
- Frishman, Yaniv, and Ayellet Tal. "Online dynamic graph drawing." *Visualization and Computer Graphics, IEEE Transactions on* 14, no. 4 (2008): 727-740.
- Fruchterman, Thomas MJ, and Edward M. Reingold. "Graph drawing by force-directed placement." *Software: Practice and experience* 21, no. 11 (1991): 1129-1164.
- Gajer, Pawel, Michael T. Goodrich, and Stephen G. Kobourov. "A multi-dimensional approach to force-directed layouts of large graphs." In *Graph Drawing*, pp. 211-221. Springer Berlin Heidelberg, 2001.
- Gajer, Pawel, and Stephen G. Kobourov. "Grip: Graph drawing with intelligent placement." In *Graph Drawing*, pp. 222-228. Springer Berlin Heidelberg, 2001.
- Gansner, Emden R., Yifan Hu, and Stephen North. "Visualizing streaming text data with dynamic graphs and maps." In *Graph Drawing*, pp. 439-450. Springer Berlin Heidelberg, 2013.
- Gansner, Emden R., Yehuda Koren, and Stephen North. "Graph drawing by stress majorization." In *Graph Drawing*, pp. 239-250. Springer Berlin Heidelberg, 2005.
- Goodrich, Michael T., and Roberto Tamassia. "Dynamic trees and dynamic point location." In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pp. 523-533. ACM, 1991.
- Görg, Carsten, Peter Birke, Mathias Pohl, and Stephan Diehl. "Dynamic graph drawing of sequences of orthogonal and hierarchical graphs." In *Graph Drawing*, pp. 228-238. Springer Berlin Heidelberg, 2005.
- Gort, Adam, and James Gort. *Survey of Network Visualization Tools*. No. DRDC-0-CR-2007-280. NRNS INC OTTAWA (ONTARIO), 2007.

- Hachul, Stefan, and Michael Jünger. "Drawing large graphs with a potential-field-based multilevel algorithm." In *Graph Drawing*, pp. 285-295. Springer Berlin Heidelberg, 2005.
- Hadany, Ronny, and David Harel. "A multi-scale algorithm for drawing graphs nicely." *Discrete Applied Mathematics* 113, no. 1 (2001): 3-21.
- Harel, David, and Yehuda Koren. "A fast multi-scale method for drawing large graphs." In *Graph drawing*, pp. 183-196. Springer Berlin Heidelberg, 2001.
- Harel, David, and Yehuda Koren. "Graph drawing by high-dimensional embedding." In *Graph Drawing*, pp. 207-219. Springer Berlin Heidelberg, 2002.
- Harel, David, and Meir Sardas. "Randomized graph drawing with heavy-duty preprocessing." *Journal of Visual Languages and Computing* 6, no. 3 (1995): 233-253.
- Hendrickson, Bruce, and Robert W. Leland. "A Multi-Level Algorithm For Partitioning Graphs." *SC 95* (1995): 28.
- Hendrickson, Bruce, and Robert Leland. *The Chaco user's guide Version 2.0*. Vol. 4. Technical Report SAND95-2344, Sandia National Laboratories, 1995.
- Herman, Ivan, Guy Melançon, and M. Scott Marshall. "Graph visualization and navigation in information visualization: A survey." *Visualization and Computer Graphics, IEEE Transactions on* 6, no. 1 (2000): 24-43.
- Hu, Yifan. "Algorithms for visualizing large networks." *Combinatorial Scientific Computing* 5, no. 3 (2011): 180-186.
- Hu, Yifan. "Current and Future Challenges in the Visualization of Large Networks." (2013).
- Hu, Yifan. "Efficient, high-quality force-directed graph drawing." *Mathematica Journal* 10, no. 1 (2005): 37-71.
- Huang, Weidong, and Peter Eades. "How people read graphs." In *proceedings of the 2005 Asia-Pacific symposium on Information visualisation-Volume 45*, pp. 51-58. Australian Computer Society, Inc., 2005.
- Huang, Weidong, Peter Eades, and Seok-Hee Hong. "Beyond time and error: a cognitive approach to the evaluation of graph drawings." In *Proceedings of the 2008 Workshop on BEyond time and errors: novel evaluation methods for Information Visualization*, p. 3. ACM, 2008.
- Huang, Weidong, Peter Eades, Seok-Hee Hong, and Chun-Cheng Lin. "Improving force-directed graph drawings by making compromises between aesthetics." In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pp. 176-183. IEEE, 2010.
- Huang, Mao Lin, Peter Eades, and Junhu Wang. "On-line animated visualization of huge graphs using a modified spring algorithm." *Journal of Visual Languages & Computing* 9, no. 6 (1998): 623-645.

- Ibarra, Louis. "Fully dynamic algorithms for chordal graphs and split graphs." *ACM Transactions on Algorithms (TALG)* 4, no. 4 (2008): 40.
- Kamada, Tomihisa, and Satoru Kawai. "An algorithm for drawing general undirected graphs." *Information processing letters* 31, no. 1 (1989): 7-15.
- Kamada, Tomihisa, and Satoru Kawai. "A simple method for computing general position in displaying three-dimensional objects." *Computer Vision, Graphics, and Image Processing* 41, no. 1 (1988): 43-56.
- Kaufmann, Michael, and Dorothea Wagner, eds. *Drawing graphs: methods and models*. Vol. 2025. Springer, 2001.
- Kobourov, Stephen G. "Force-directed drawing algorithms." (2004).
- Kobourov, Stephen G. "Spring Embedders and Force Directed Graph Drawing Algorithms." *arXiv preprint arXiv:1201.3011* (2012).
- Koren, Yehuda, Liran Carmel, and David Harel. "ACE: A fast multiscale eigenvectors computation for drawing huge graphs." In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pp. 137-144. IEEE, 2002.
- Kumar, Aruna, and Richard H. Fowler. *A spring modeling algorithm to position nodes of an undirected graph in three dimensions*. Tech. rep., Department of Computer Science, University of Texas, Pan American, Edinburg, 1994.
- Lai, Wei, Kazuo Misue, and Kozo Sugiyama. *Preserving the mental map of a diagram*. International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.
- Lee, Yi-Yi, Chun-Cheng Lin, and Hsu-Chun Yen. "Mental map preserving graph drawing using simulated annealing." In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation-Volume 60*, pp. 179-188. Australian Computer Society, Inc., 2006.
- Lin, Chun-Cheng, Yi-Yi Lee, and Hsu-Chun Yen. "Mental map preserving graph drawing using simulated annealing." *Information Sciences* 181, no. 19 (2011): 4253-4272.
- Lyons, Kelly A. "Cluster busting in anchored graph drawing." In *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research-Volume 2*, pp. 327-337. IBM Press, 1992.
- Misue, Kazuo, Peter Eades, Wei Lai, and Kozo Sugiyama. "Layout adjustment and the mental map." *Journal of visual languages and computing* 6, no. 2 (1995): 183-210.
- Nguyen, Quan, Peter Eades, and Seok-Hee Hong. "On the faithfulness of graph visualizations." In *Graph Drawing*, pp. 566-568. Springer Berlin Heidelberg, 2013.
- North, Stephen C., and Gordon Woodhull. "Online hierarchical graph drawing." In *Graph Drawing*, pp. 232-246. Springer Berlin Heidelberg, 2002.

- Papakostas, Achilleas, and Ioannis G. Tollis. "Issues in interactive orthogonal graph drawing (preliminary version)." In *Graph Drawing*, pp. 419-430. Springer Berlin Heidelberg, 1996.
- Pfaltz, John L. "Graph structures." *Journal of the ACM (JACM)* 19, no. 3 (1972): 411-422.
- Purchase, Helen C. "Effective information visualisation: a study of graph drawing aesthetics and algorithms." *Interacting with computers* 13, no. 2 (2000): 147-162.
- Purchase, Helen C. "Metrics for graph drawing aesthetics." *Journal of Visual Languages & Computing* 13, no. 5 (2002): 501-516.
- Purchase, Helen. "Which aesthetic has the greatest effect on human understanding?." In *Graph Drawing*, pp. 248-261. Springer Berlin Heidelberg, 1997.
- Purchase, Helen C., David Carrington, and Jo-Anne Alder. "Empirical evaluation of aesthetics-based graph layout." *Empirical Software Engineering* 7, no. 3 (2002): 233-255.
- Purchase, Helen C., Robert F. Cohen, and Murray I. James. "An experimental study of the basis for graph drawing algorithms." *Journal of Experimental Algorithmics (JEA)* 2 (1997): 4.
- Purchase, Helen C., Robert F. Cohen, and Murray James. "Validating graph drawing aesthetics." In *Graph Drawing*, pp. 435-446. Springer Berlin Heidelberg, 1996.
- Purchase, Helen C., Eve Hoggan, and Carsten Görg. "How important is the "mental map"?—an empirical investigation of a dynamic graph layout algorithm." In *Graph drawing*, pp. 184-195. Springer Berlin Heidelberg, 2007.
- Purchase, Helen C., and Amanjit Samra. "Extremes are better: Investigating mental map preservation in dynamic graphs." In *Diagrammatic Representation and Inference*, pp. 60-73. Springer Berlin Heidelberg, 2008.
- Quigley, Aaron, and Peter Eades. "FADE: Graph drawing, clustering, and visual abstraction." In *Graph Drawing*, pp. 197-210. Springer Berlin Heidelberg, 2001.
- Saffrey, Peter, and Helen Purchase. "The mental map versus static aesthetic compromise in dynamic graphs: a user study." In *Proceedings of the ninth conference on Australasian user interface-Volume 76*, pp. 85-93. Australian Computer Society, Inc., 2008.
- Shannon, Ross, and Aaron J. Quigley. "Considerations in Dynamic Graph Drawing: A Survey." (2007).
- Sugiyama, Kozo, and Kazuo Misue. "Graph drawing by the magnetic spring model." *Journal of Visual Languages and Computing* 6, no. 3 (1995): 217-231.
- Sugiyama, Kozo, Shojiro Tagawa, and Mitsuhiro Toda. "Methods for visual understanding of hierarchical system structures." *Systems, Man and Cybernetics, IEEE Transactions on* 11, no. 2 (1981): 109-125.

- Tamassia, Roberto. "On-line planar graph embedding." *Journal of Algorithms* 21, no. 2 (1996): 201-239.
- Tamassia, Roberto, Giuseppe Di Battista, and Carlo Batini. "Automatic graph drawing and readability of diagrams." *Systems, Man and Cybernetics, IEEE Transactions on* 18, no. 1 (1988): 61-79.
- Tunkelang, Daniel. "A numerical optimization approach to general graph drawing." PhD diss., IBM, 1999.
- Tutte, William T. "How to draw a graph." *Proc. London Math. Soc* 13, no. 3 (1963): 743-768.
- Veldhuizen, Todd L. "Dynamic multilevel graph visualization." *arXiv preprint arXiv:0712.1549* (2007).
- Walshaw, Chris. "The Graph Partitioning Archive". <http://staffweb.cms.gre.ac.uk/~wc06/partition/>. Last accessed 17.02.2014.
- Walshaw, Chris. "A multilevel algorithm for force-directed graph-drawing." *J. Graph Algorithms Appl.* 7, no. 3 (2003): 253-285.
- Walshaw, Chris. "Multilevel refinement for combinatorial optimisation problems." *Annals of Operations Research* 131, no. 1-4 (2004): 325-372.
- Walshaw, Chris, Mark Cross, and M. G. Everett. "Parallel dynamic graph partitioning for adaptive unstructured meshes." *Journal of Parallel and Distributed Computing* 47, no. 2 (1997): 102-108.
- Ware, Colin, Helen Purchase, Linda Colpoys, and Matthew McGill. "Cognitive measurements of graph aesthetics." *Information Visualization* 1, no. 2 (2002): 103-110.
- Xu, Kevin S., Mark Kliger, and Alfred O. Hero III. "Tracking communities in dynamic social networks." In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pp. 219-226. Springer Berlin Heidelberg, 2011.

10 Appendix

10.1 Comparison of Running Time for Multilevel Spring Embedder with Multilevel Global Force and Multilevel Spring Embedder with Octree

The running time is broken into various parts in the subsections below;

- G_l – The number of graphs in the multilevel scheme
- Co – time taken to coarsen the graph
- FDP_f – full time taken for all Force Directed Placement
- FDP_r – time taken to calculate repulsive Force Directed Placement
- FDP_a – time taken to calculate attractive (spring) Force Directed Placement
- d – time taken to calculate vertex displacement
- I – time taken to interpolate layout between the graphs in the multilevel scheme
- O – time take to run updates on the approximation to maintain accuracy
- RT – full running time for the algorithm

Multilevel Spring Embedder with Multilevel Global Force

Graph	Running Time (ms)								
	G_l	Co	FDP_f	FDP_r	FDP_a	d	I	O	RT
4elt	24.0	62.80	4392.20	2923.20	1151.20	214.40	0.00	0.00	4455.00
3025	15.8	15.60	609.20	441.60	96.60	52.40	0.00	0.00	628.00
add32	77.4	15.60	1277.80	951.20	175.00	76.20	3.20	6.20	1302.80
data	14.6	9.60	675.00	409.00	234.40	19.00	3.20	0.00	687.80
dime20	33.0	1034.60	96203.40	69294.20	19295.00	5950.80	81.40	62.20	97381.60
finan512	138.0	990.20	35402.40	20950.00	12089.60	1759.20	34.60	58.80	36486.00
mesh100	36.2	640.80	43257.80	28872.20	11107.00	2255.20	28.20	31.20	43958.00
sierpinski10	26.6	334.00	31408.40	22124.60	6760.20	1682.20	28.00	19.20	31789.60

10.1.1 Multilevel Spring Embedder with Octree Running Time

	Running Time (ms)								
	G_l	Co	FDP_f	FDP_r	FDP_a	d	I	O	RT
4elt	22.8	61.16	7310.66	5815.00	1178.42	221.18	4.40	58.42	7434.64
3025	16.0	7.52	888.14	734.90	105.52	31.84	0.30	12.86	908.82
add32	61.7	14.46	2366.68	2034.04	208.76	93.44	1.86	27.86	2410.86
data	14.3	14.40	927.84	642.58	241.36	27.58	0.00	10.88	953.12
dime20	38.6	1052.04	194292.94	166241.96	20125.66	5551.80	63.26	1394.36	196802.60
finan512	122.0	905.60	59375.90	44635.80	12196.24	1729.70	18.86	407.72	60708.08
mesh100	40.1	663.42	76108.86	61722.36	10799.84	2461.34	29.68	571.46	77373.42
sierpinski10	29.2	362.48	62320.10	52578.58	6931.96	1889.54	24.20	539.28	63246.06

10.2 Coarsening Tolerance

A coarsening tolerance is included for preventing the generation of graphs which are too similar in the multilevel scheme, that is, several consecutive graphs which hold the same structural information (few vertices difference). Such graphs are likely to have similar layout and therefore application of force directed placement is wasteful. In addition, the effect on Multilevel Global Force is an increase in the number of graphs in the approximation tree, affecting running time, and similar approximations being made during calculation of repulsive forces, affecting layout.

In the literature, Hu (2005) details a hybrid coarsening scheme in order to overcome this issue, identifying when approximations of a graph become too similar and switching to an alternate coarsening scheme thereafter. The heuristic used by Hu for determining when approximations are too similar is used here (information regarding Implementation can be found in Section 10.22), however the hybrid coarsening scheme is not used as graphs that reach this limit tend to be small enough for a layout to be generated by standard Force Directed Placement, therefore further coarsening is not required.

The use of a tolerance will reduce the dimensions of the Multilevel Global Force structure, therefore impacting the number of approximations made during repulsive force calculation (approximation with G_L levels has complexity $O(L \cdot |V| \cdot (m-1) \cdot (|G_i|-1))$ where m is the matching number used (typically 2)).

The tolerance, tol , is tested with incremental values between 0 and 20, such that if three abstractions are generated with fewer than tol vertices difference n consecutive times, the coarsening phase stops early. A value of 3 is given to n (Hu, 2005). The tolerance is tested on the static test graphs and are repeated 10 times for each value of the tolerance, resulting in 210 layouts $\{10 * 21\}$ for each graph tested. Results for layout quality and running time are analysed and averaged for comparison.

10.2.1 Analysis of Numerical Results

The principle effect of altering the tolerance value is on the number of graphs and size of the coarsest graph generated through the coarsening process. In summary, as the tolerance increases from 0 to 20, the number of graphs in the multilevel scheme decreases by an average of 2.03, and the size of the coarsest graph increases by an average of 4.05 vertices. For context, the average number of graphs in the multilevel scheme is 15.9.

Due to these changes, the running time is expected to gradually decrease, shown as a 4.5% decrease between values of 0 and 20, in **Figure 10.1**. Due to the increase in size of the coarsest graph, the reduction in running time will begin to equal out and gradually increase as described by the complexity above,

however, accurately predicting such changes is difficult due to the random nature of the coarsening scheme (not every vertex will necessarily be matched with another).

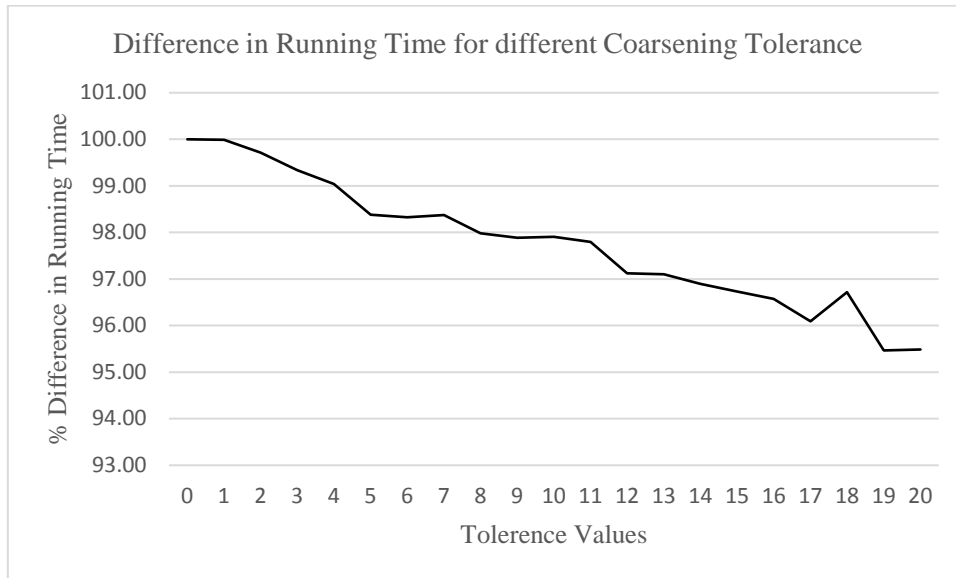


Figure 10.1. Difference in running time as a result of changing the value of the coarsening tolerance.

Analysis of the layouts shows that increasing the tolerance can result in a noticeable change in edge crossings, with over a 400% increase in edge crossings for some values, shown in **Figure 10.2** as average1. The average include erratic and dramatic increases in edge crossings which arise from folds exhibited by the graph 3025; a graph which normally has planar layout. Removing the values provides the much more regular average depicted as average2 in **Figure 10.2**, dropping the average increase in edge crossings to 50%, still

The increase in edge crossings is significant in comparison to the reduction in running time, and is largely due to the graphs dime20 and sierpinski10, which show an increase in edge crossings of over 100% between tolerance of 0 and 20 in comparison to 9% for other graphs, as shown for comparison in **Figure 10.3**. Due to the erratic behaviour associated with the graphs 3025, sierpinski10 and dime20, a default value for all graphs is best kept low to prevent depreciation of layout quality, as such, a value of 1 or 2 is suggested.

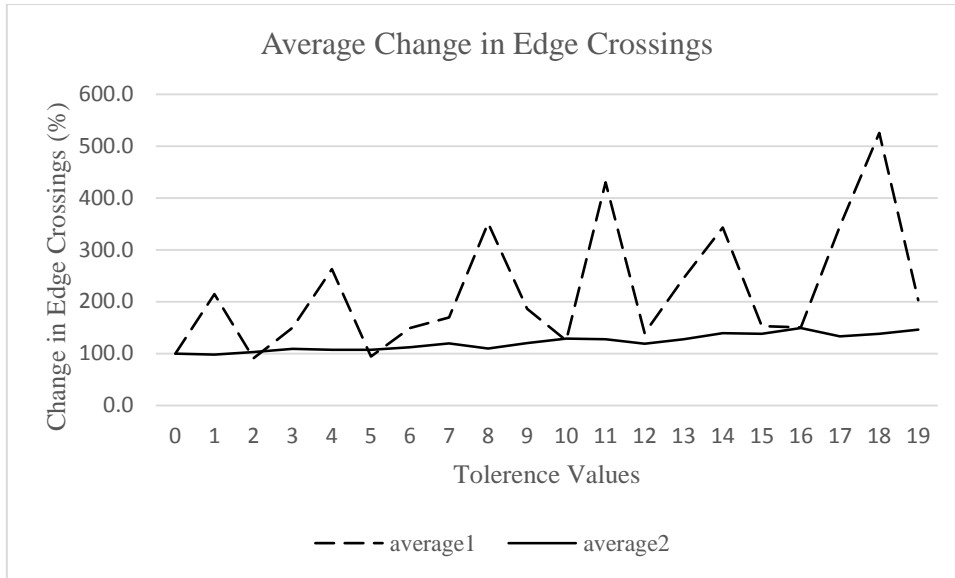


Figure 10.2. Average change in edge crossings, whereby average1 refers to all the test graph and exhibits erratic and dramatic changes, and average2 refers to all graphs except 3025 (the cause of the spikes seen in average1).

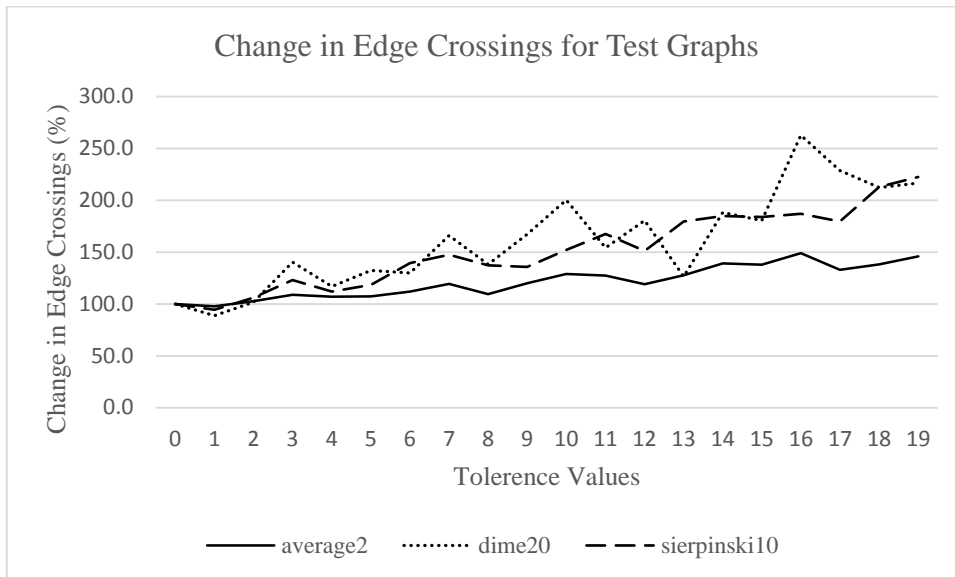


Figure 10.3. Average change in edge crossings for all graphs except 3025, represented as average2, and the changes in edge crossings for sierpinski10 and dime20, showing the significant increase over the average (of up to 150% increase).

10.2.2 Subjective Results

For many graphs, the difference between tolerances is small but gradual, for example, layouts for 4elt drawn using a tolerance of 0 and 20 have little differences upon the number of edge crossings, as shown in **Figure 10.4**, whereas changing the value to a 500 (a high value) results in noticeable depreciation of layout quality, shown in **Figure 10.5**. The difference in value appears to have high impact on global layout, suggesting that the coarser graphs, and placement of vertices within them, are affected by the tolerance.

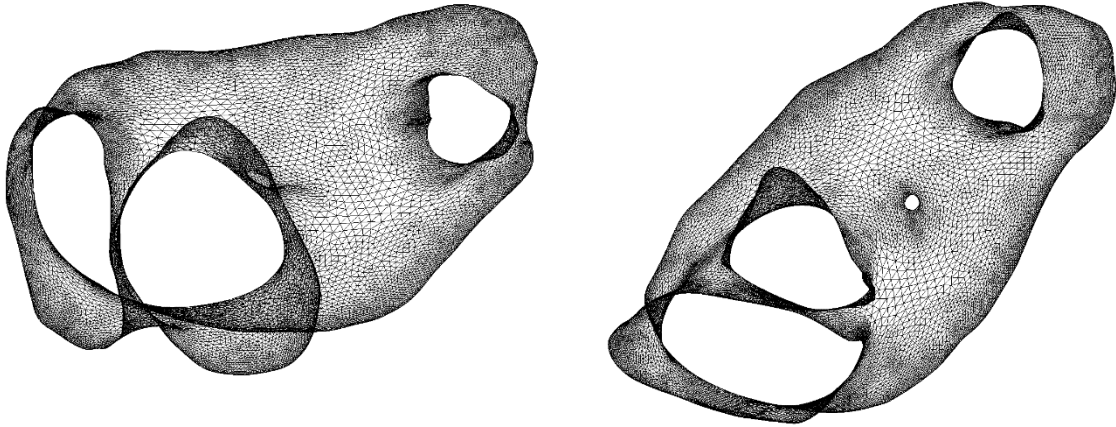


Figure 10.4. Layouts for the graph 4elt showing changes to layout as a result of changing the value of the coarsening tolerance to 0 (left) and 20 (right)

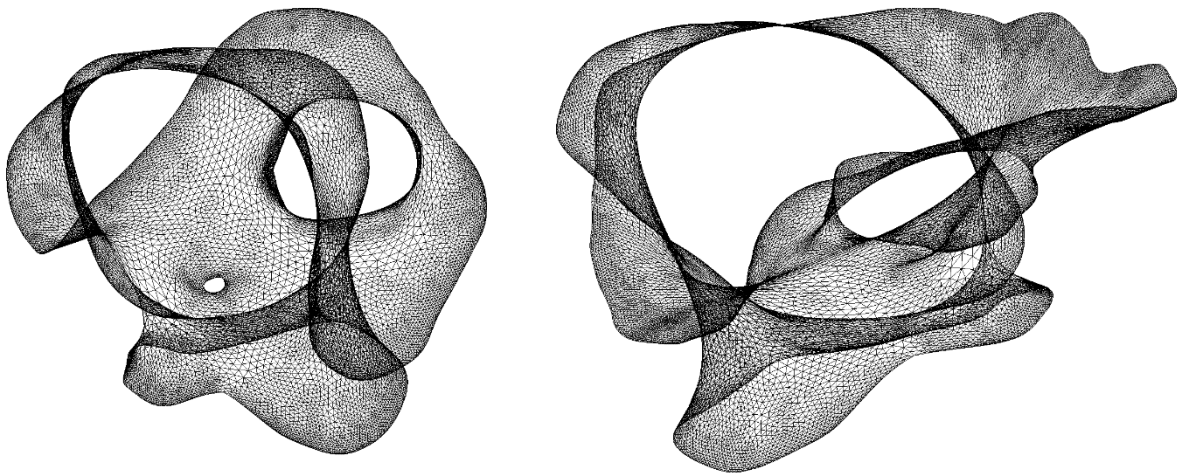


Figure 10.5. Layouts for the graph 4elt showing changes to the layout as a result of change the value of the coarsening tolerance to 100 (left) and 500 (right)

A comparison of layouts for dime20 using tolerance of 0 and 20, provided in **Figure 10.6**, shows a similar conclusion, that the tolerance impacts the global layout. Unlike the 4elt example above however, the global layout changes with a smaller value for tolerance (20 as opposed to 100), suggesting that the effect on layout is related to the weights of vertices in the coarser graph (vertices in the coarsest approximation of 4elt have less weight than those in the coarsest approximation of dime20).

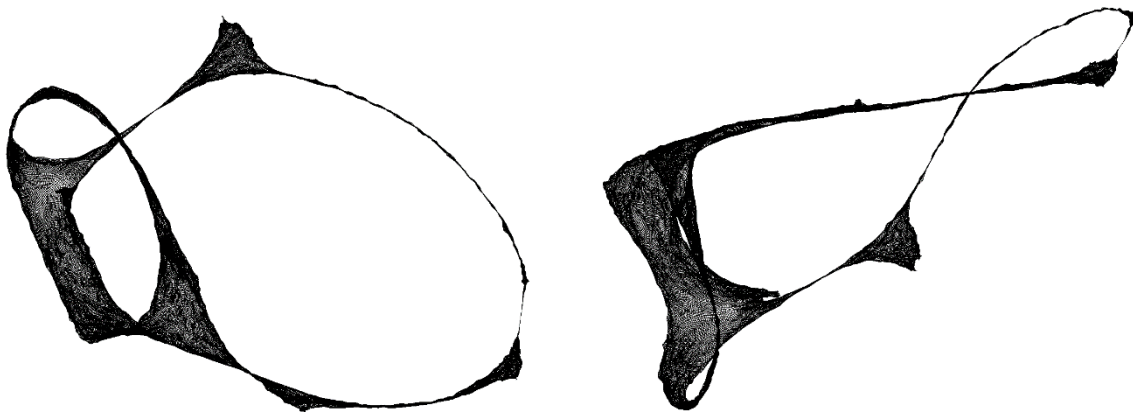


Figure 10.6. Layouts for the graph `dime20`, showing changes as a resulting of changing the value of the coarsening tolerance to 0 (left) and 20 (right)

To confirm the observation, a smaller graph (3025, see **Figure 10.7**) is provided layout using a tolerance of 100 and 500, and as expected, the higher tolerance has a reduced effect on graphs with fewer vertices. The tolerance of 500 shows that although higher tolerance has reduced effect, higher values will inevitably have an effect.

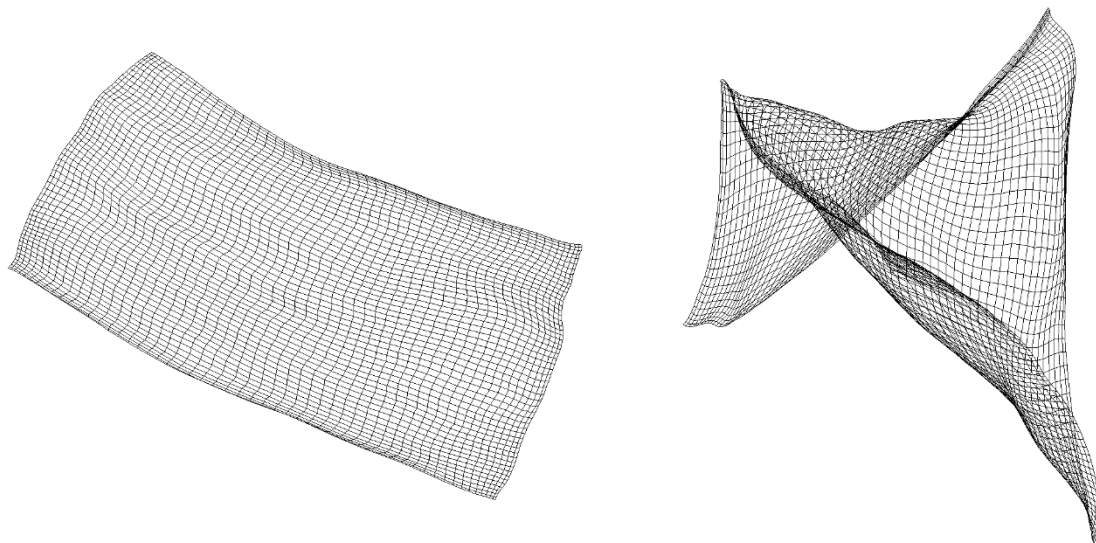


Figure 10.7. Layouts for the graph `3025`, showing changes as a resulting of changing the value of the coarsening tolerance to 0 (left) and 20 (right).

10.3 Correspondence with Daniel Danko

Correspondence with “Daniel Danko” regarding his video entitled “3elt 3D graph layout on GPU” published on Youtube on 27 April 2013.

Carl Crawford

hi daniel, the work looks great ^^ the speed you get from the GPU is incredible, quick question though; what values do you give your cooling schedule in order to get such smooth vertex movement?

Daniel Danko

0.5 for attractive forces and 0.9 for repulsive forces but i guess the values can say nothing if you dont know how exactly the final force on the node is computed

Carl Crawford

Thanks for the reply and the information, it was very helpful although I have chosen alternate values (probably due to differences in implementation) the use of individual cooling schedules for both forces, having previously used only one for the combined movement, works brilliantly leading to much smoother convergence of layout though one more question, if I may, do you have any sources or publications which suggests the use of two schedules that would be beneficial to read? thanks again!

No further replies were given and no indication on sources were provided.

Video available at: <http://www.youtube.com/watch?v=15eFkE-rVVk>

Last checked online: 04.02.2014

10.4 Configuring Multilevel Global Force

To ensure high quality layouts, a parameter is introduced to alter the equilibrium between repulsive and spring forces. The aim is influence layout generation such that one force is given priority, and to find which alteration of this equilibrium provides the highest quality results. A similar parameter is used in the methods described by Hu (2005) and Walshaw (2003), with earlier and more extensive parameterisation in methods modelling Simulated Annealing (Davidson and Harel, 1996).

The parameter, C , is introduced into the calculation of repulsive forces as described by Walshaw (2003) and Hu (2005) as indicated below. A default value of 0.2 is used by Walshaw and Hu, with investigation of values for use with Multilevel Global Force through experimentation with values of: 5.0, 1.0, 0.5, 0.05, 0.005 and 0.0005. It is expected that values below 1.0 reduce the effect of the repulsive forces, giving priority to spring force displacement, and values greater than 1.0 resulting in more expansion of the layout.

$$fr(w, k, d) = \frac{Cwk^2}{d}$$

Layouts for static test graphs are generated for the differing values, repeated 10 times for each graph. The results are averaged and plotted as line charts, with the chart for edge crossings expected to have a prominent peak as a result of instability between forces.

10.4.1 Numerical Results

In order to provide some context for the parameters, initial investigation tests 4 medium sized graphs to determine how such parameter usage effects both Multilevel Global Force and Octree data structures, as well as an n-body Multilevel scheme. The edge crossings of layouts are then generalised (to show the difference in edge crossings per parameter as a percentage of edge crossings for the first parameter value) in order to compare the drop in layout quality and when such drop occurs across the methods.

Figure 10.8 shows that for the four graphs, the number of edge crossings exhibited by layouts generated by the multilevel method increases dramatically when using a value of 0.032 or above. The change in edge crossings for Octree and *MGF* show a slight increase, however, not as substantial suggesting approximation is less effected by the parameter. The value which provides layouts with least edge crossings is 0.004 for all methods. The value differs from those used by Walshaw (2003) and Hu (2005), both of which use a value of 0.2 after some experimentation, a difference which likely comes from differences in the size of the drawing area and calculation of the values of k (which here uses the area as opposed to a function of some initial placement described by Walshaw, 2003).

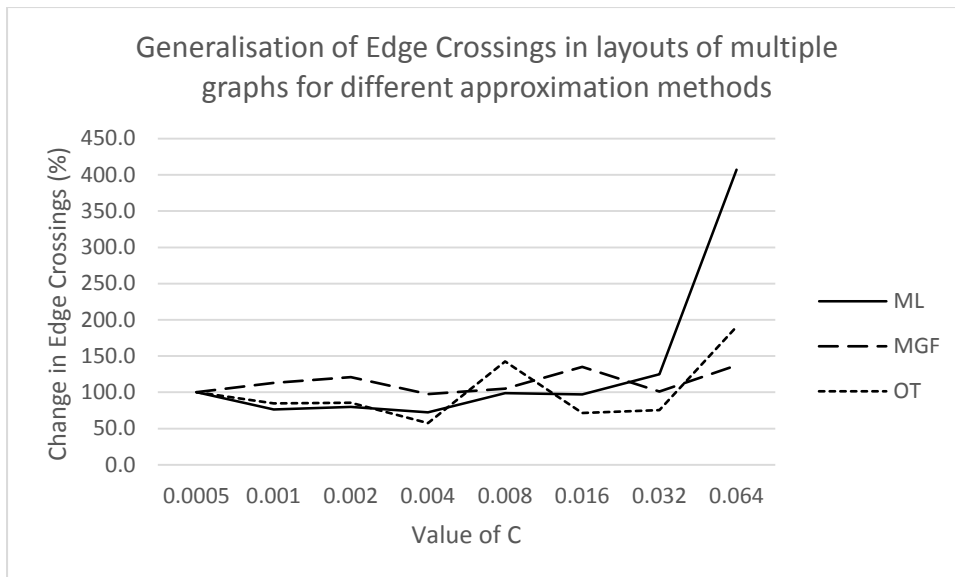


Figure 10.8 Generalisation of edge crossings in layouts of multiple graphs for different approximation methods

Further testing investigates values for only Multilevel Global Force, with the number of edges crossings for a larger selection of test graphs provided in Table 10.1. The values show similar rises in edge crossings as the precursory investigation above, identifying 0.005 as the parameter value with offers layouts with fewest edge crossings.

Specific testing for Multilevel Global Force supports a reduced value of 0.005.

	Parameter C values					
	0.0005	0.005	0.05	0.50	1.00	5.00
3025	1095.3	605.3	788.5	1394.6	1808.7	3863.6
data	36836.4	35579.7	35645.6	36344.5	39961	50384
add32	16591.9	17553.3	17364.8	16897.4	16028.3	16841.3
4elt	27763.6	25620.3	26861.2	32897.4	36549.7	51011
sierpinski10	37189	40144.4	40449	40824.8	40122	40643.8
finan512	5043279	6068587	6035649	5841397	5930307	5851012

Table 10.1 The number of edge crossings exhibited in a layout for a collection of graphs for differing values of the C parameter used in Modified Spring Embedders. The figures show that values in the range of 0.0005 and 0.05 provide fewest edge crossings for the development environment

It should be noted that for implementation of space decomposition approximation, higher parameter values will cause expansion or movement within the layouts of graphs, and therefore the data structure must react if the graph expands beyond the area governed by the structure. If not, vertices outside the area will not be accurately modelled by the approximation and layouts may exhibit stretching as a result,

for example, Figure 10.11 provided in Subjective Analysis below, which shows part of the graph data outside of the approximation grid, with those vertices outside featuring compression.

10.4.2 Subjective Analysis

The effect on layout matches the number of edge crossings measured for layouts in the numerical analysis above, as shown in Figure 10.9. For higher values of C , for example an extreme value of 200.0 (not included in the testing above), global layout decreases in quality. Local layout is still exhibited however which suggests that the parameter has larger effect in the coarser graphs in which vertices have higher weights. For the extreme value of 200.0, repulsive forces are so high that springs are unable to pull much of the graph into low energy position, but instead of constant expansion.

Reduction of the value shows quick improvement in local layout and gradual improvement in global layout.

It should be noted however, that edge crossings are expected to indicate layout quality due to their impact in smaller graphs, therefore layouts which exhibit fewer edge crossings (such as that generated for $C=0.0005$) are expected to be better than those which may be easier to read for some users (for this author, that generated at $C=0.05$).

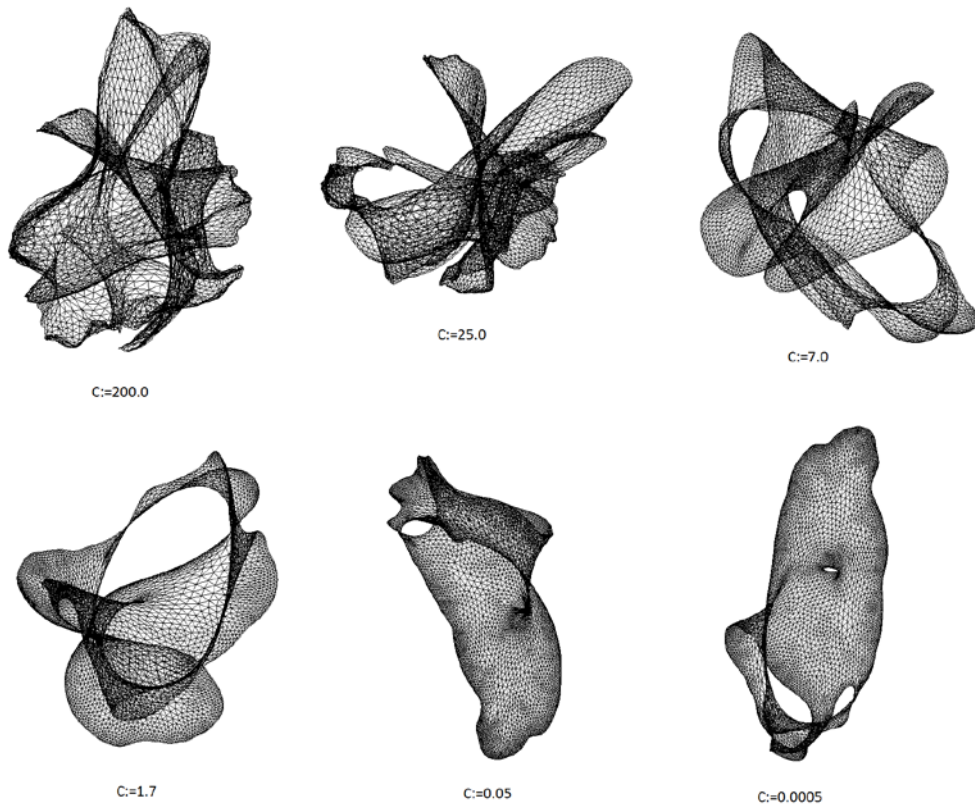


Figure 10.9 Impact on changing the value of C on graph layout of 3elt

Another effect of the parameter is that seen during usage of approximation, wherein if the values are not sufficient to repulse forces, parts of the graph are unable to escape minima as a result of the grid structure as depicted in Figure 10.10. A benefit of gradient based space decomposition, and more so of Multilevel Global Force, that the grid type structure differs per vertex and therefore does not provide such limit.

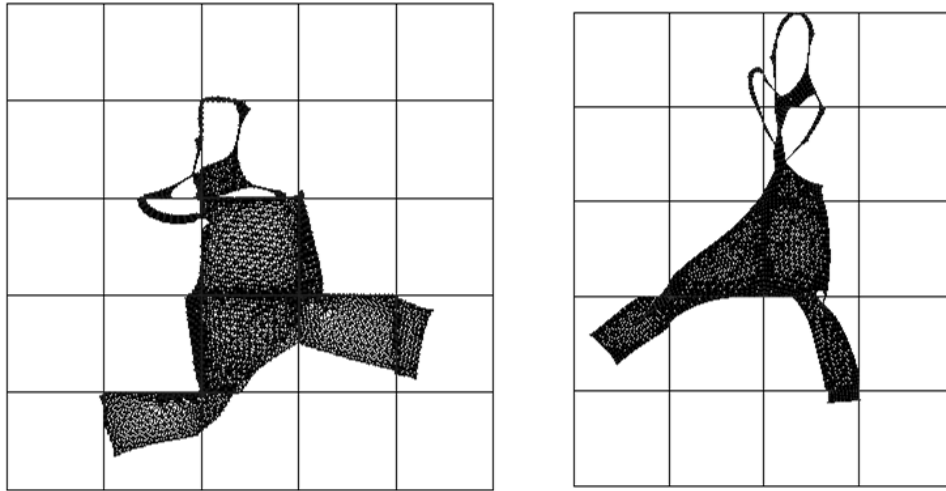


Figure 10.10 Approximation with differing scaling values (C) on the forces, showing the difference in layout as a result of the approximated forces, highlighting the need for configuration

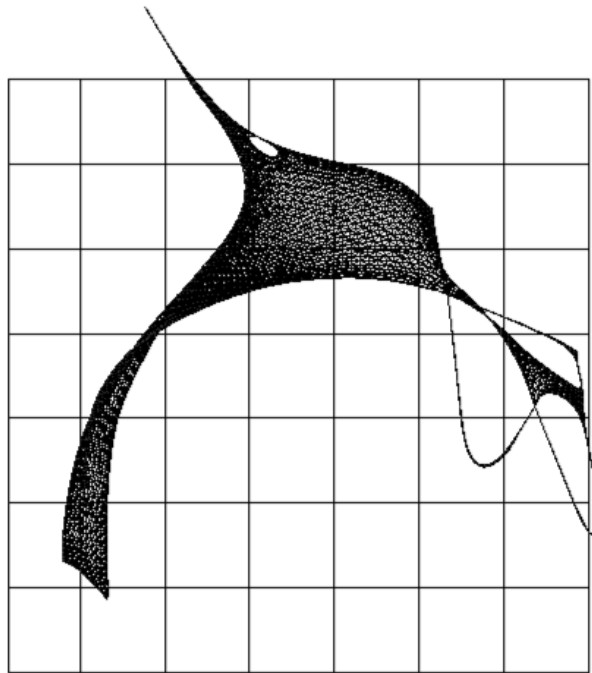


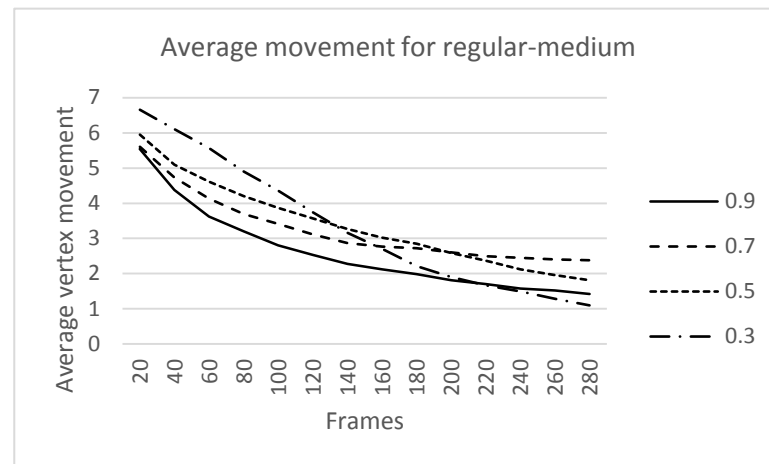
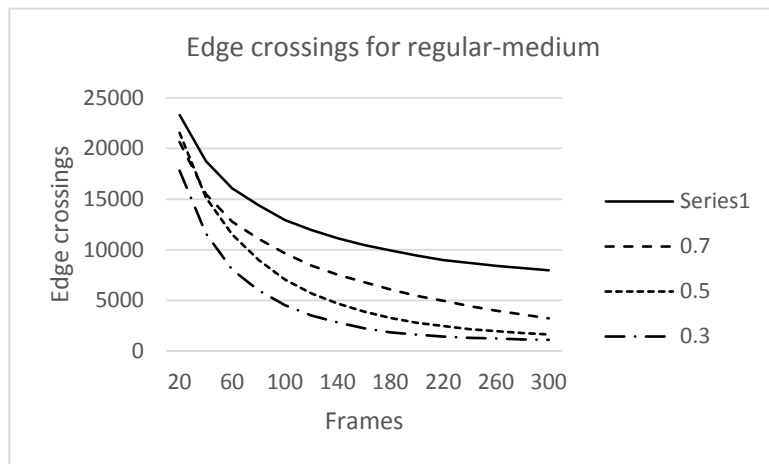
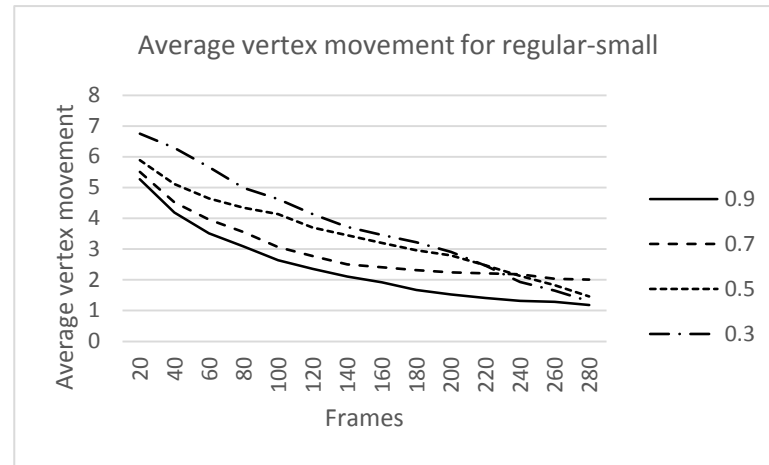
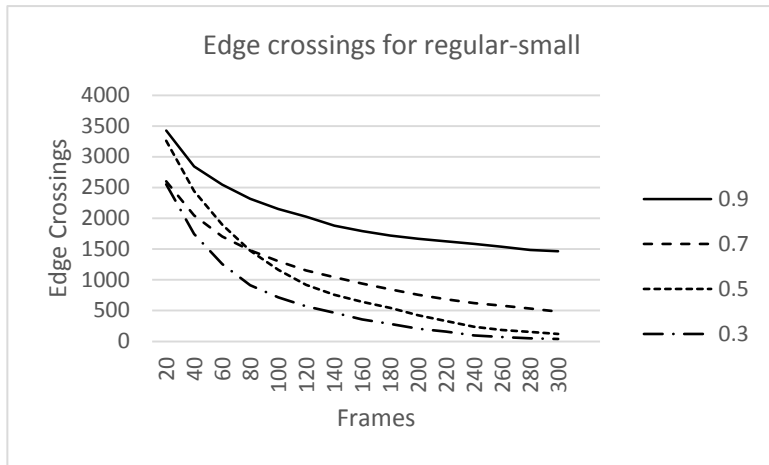
Figure 10.11 Layout of data showing part of the graph outside of the approximation area

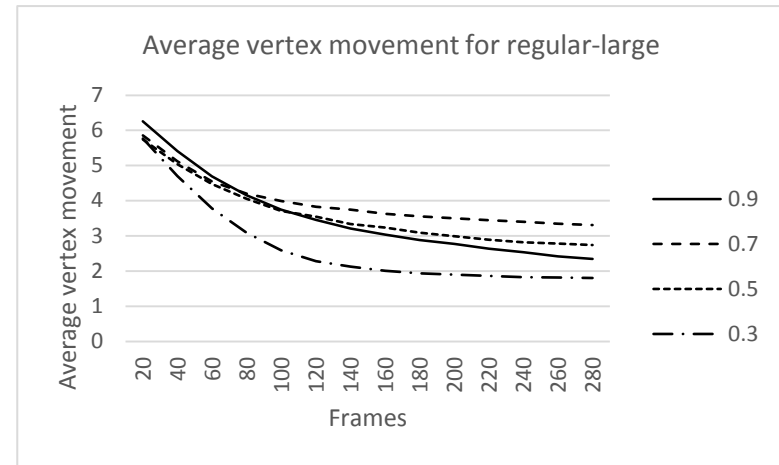
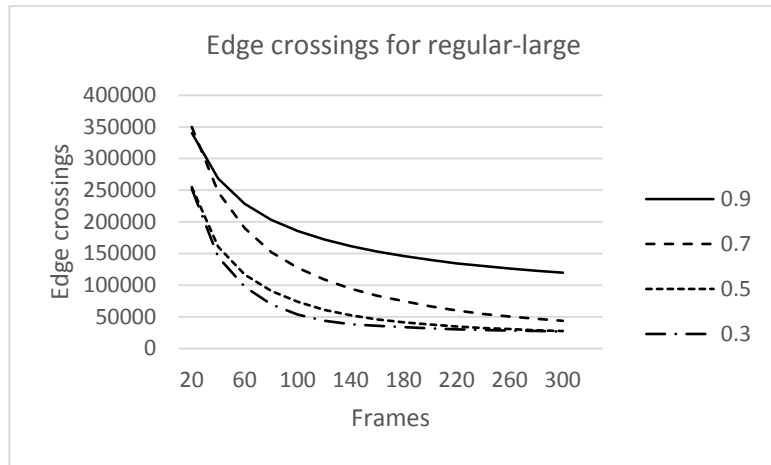
10.5 Configuring Dynamic Spring Embedder: Multilevel Global Force

Due to the large number of results for parameter testing, a summary of the generalized results is provided here. Unlike the results for *FRD*, *MGF* allows for larger graphs to be drawn in less time per frame (as with any approximation). As such, the results provide a combination of all graphs sizes for each graph type (small, medium and large), identifying whether the values identified previously are still appropriate for larger graphs.

Regular graphs show a consistent pattern through the graph sizes; edge crossings are reduced for lower values of tr , as shown in the charts below, values of 0.3 provide fewer edge crossings with 0.5 coming close second. Values of 0.9 show to have higher number of edge crossings for the small, medium and large graph. Given lower edge crossings for lower values of repulsive force, it suggests the regular graph structure is more dependent on spring forces for overcoming minima.

In contrast, the average vertex movement is minimal for higher values in the smaller graphs, with 0.9 showing least movement for small and medium graphs, and second least movement for the large. Across the results, the movement decreases in a parabolic fashion, dropping quickly in early stages and slowing in the later frames, however, a value of 0.3 shows a more linear decrease in vertex movement over time.

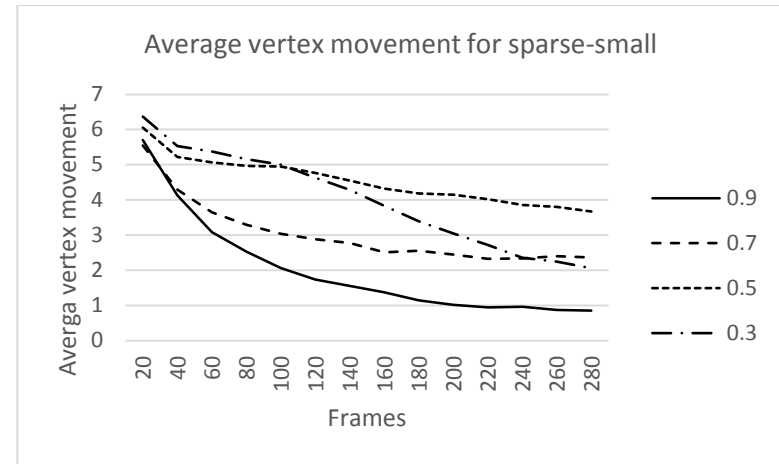
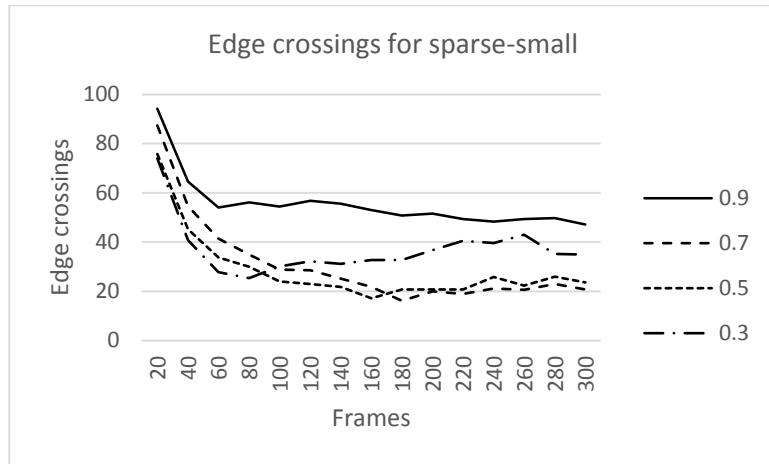


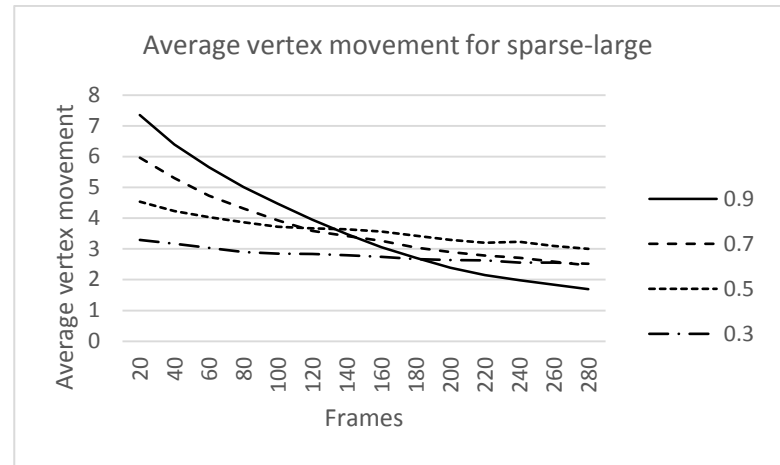
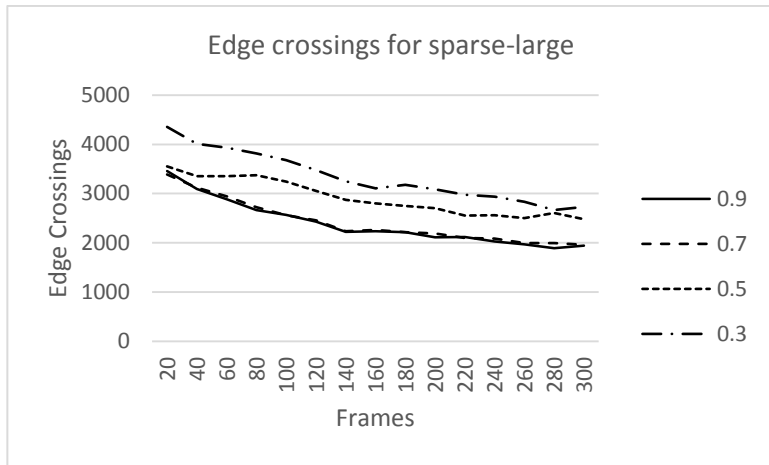
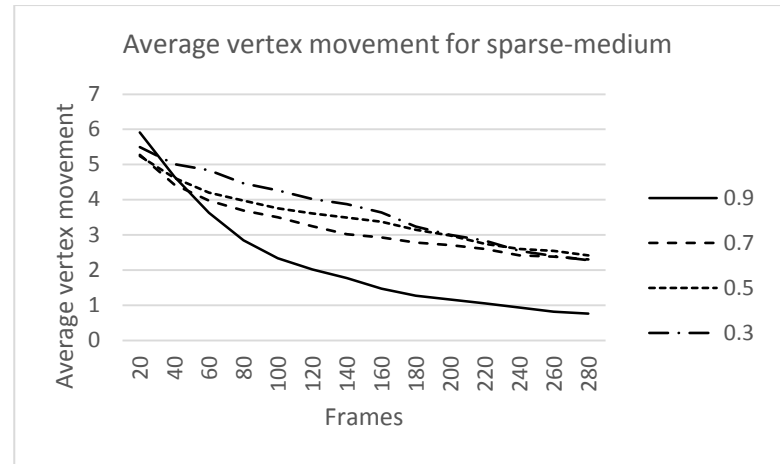
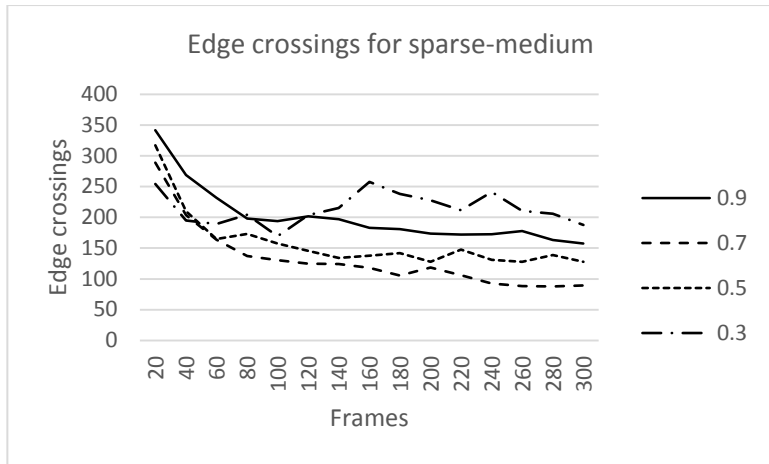


Sparse graphs show a less obvious choice for default values of the graph. The number of edge crossings greatly drop for smaller graphs but very gradual difference across frames for larger graphs. Across the results for edge crossings, a value of 0.5 appears to provide better quality layouts for the different sizes, however, the results for other values change throughout, with the edge crossings for 0.9 showing higher counts for the smaller graph and lower counts for the larger graphs. This confirms that graph size, as well as graph structure, are impacted by these values and finding a best value will depend very much on the data being visualised.

For average vertex movement the charts show lower values of tr having a more gradual reduction in movement between frames. This is a bit more sporadic in the smaller graphs due to fewer vertices and movement of a single vertex having a higher impact on the result, but is better described in the chart for sparse-large, which shows lower values (0.3 and 0.5) having a much more gradual change in movement and 0.9 having a much more noticeable drop.

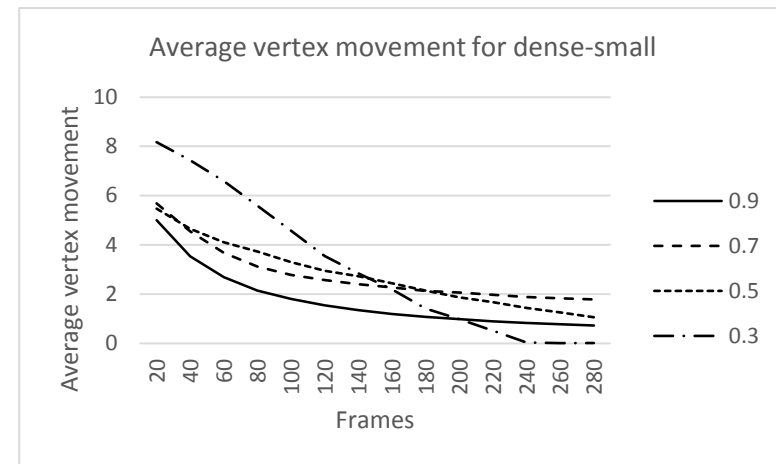
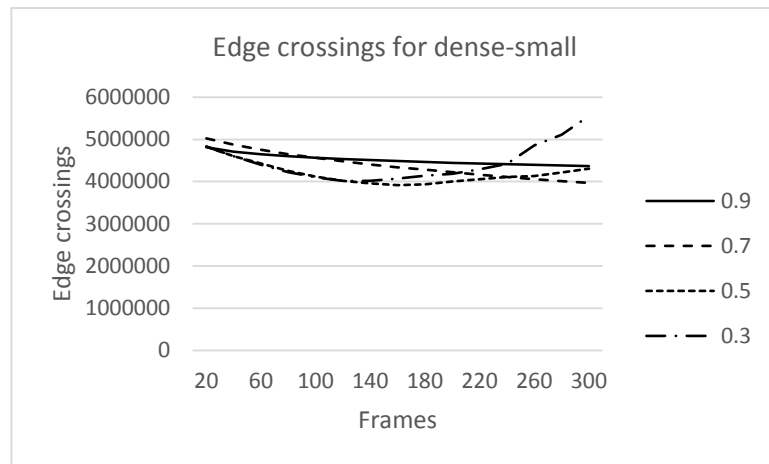
The pattern suggests that the higher repulsive forces push vertices into lower energy positions faster, resulting in the decreased movement. However, if this was so, one would also expect lower edge crossings as a result, which is not always shown for the values of 0.9 (for small and medium graphs in particular). Once again values are dependent on the size of the graphs, and so a value of 0.5 appears to work universally for sparse graphs.





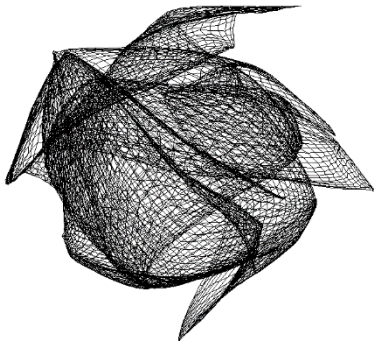
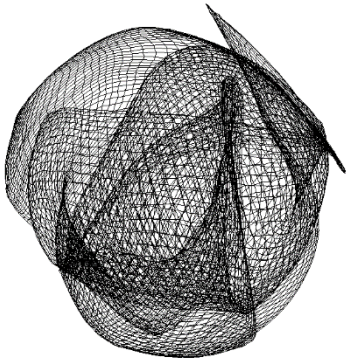
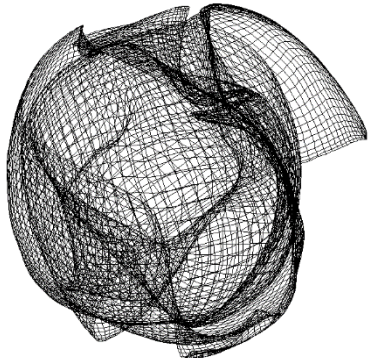
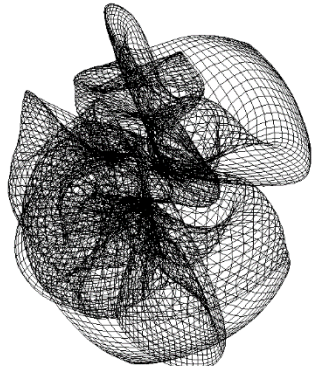
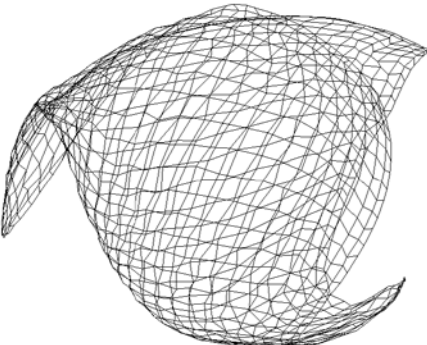
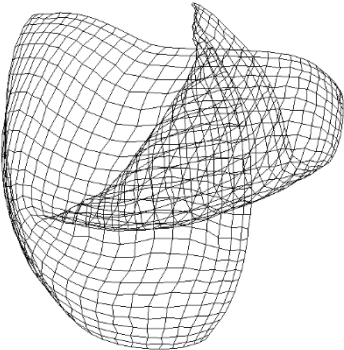
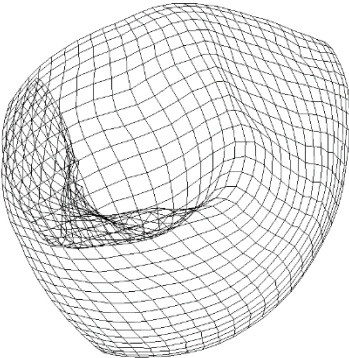
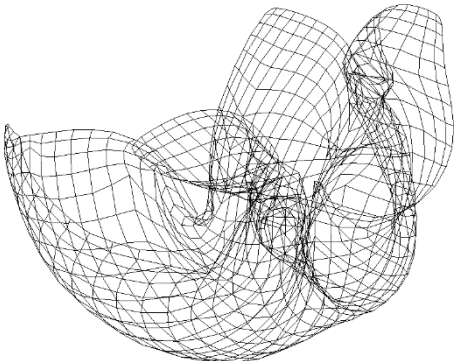
Dense graphs have little impact from the change in force equilibrium due to the natural density of edges within the graph, as shown in the charts below – the edge crossings show relatively little change as a result of the change in forces. That being said, the difference at its largest is still 1 million edge crossings, for which the values of 0.3 and 0.5 provide layouts with highest quality.

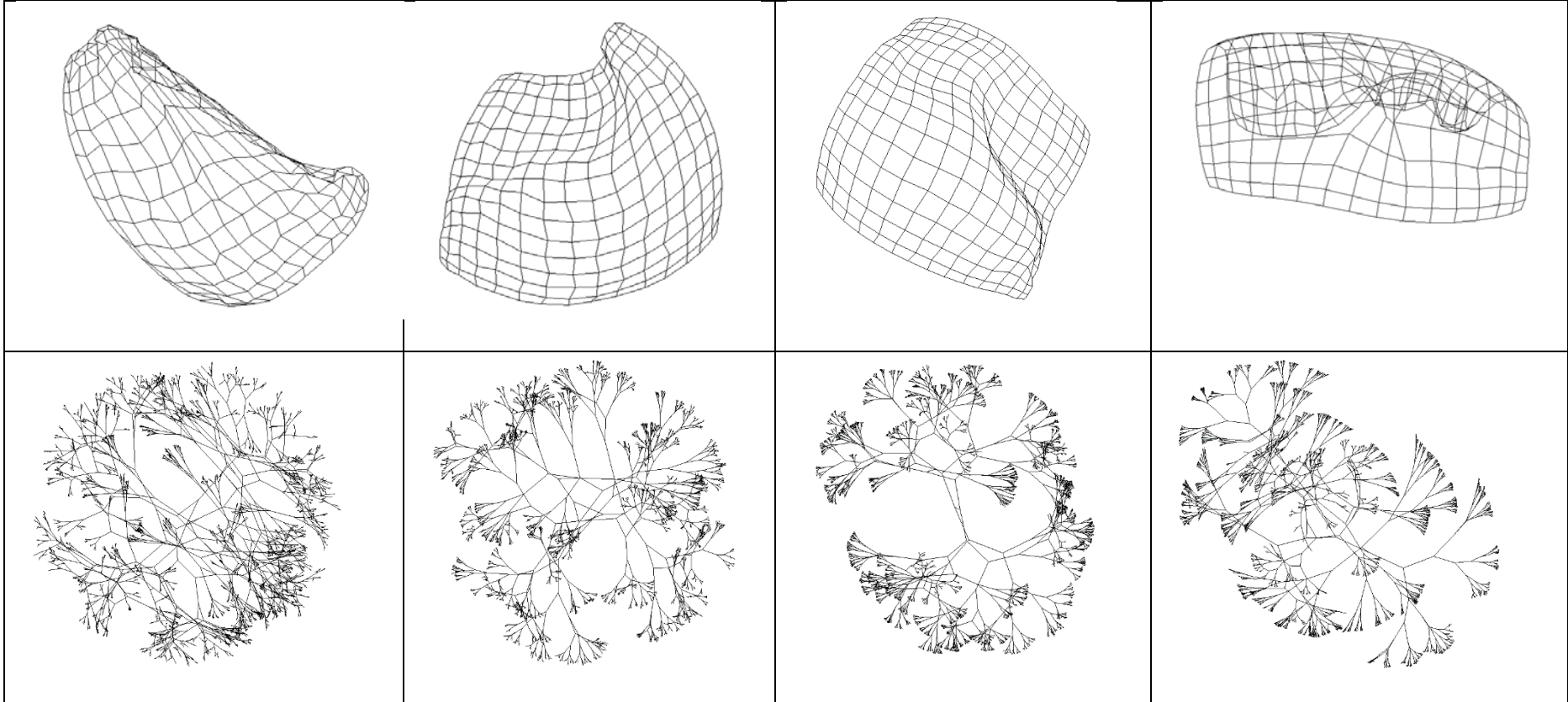
The average vertex movement is shown to be more affected, with a value of 0.9 showing continual and stable reduction in movement throughout. Although showing continual movement (gradual change), the lowest movement is achieved using a value of 0.3, which shows largest movement initially and lowest movement towards the end of the visualisation, suggesting large movements to find lower energy positions. For this reason, a value of 0.3 and 0.9 are suggested, however, subjective analysis is required to validate these.

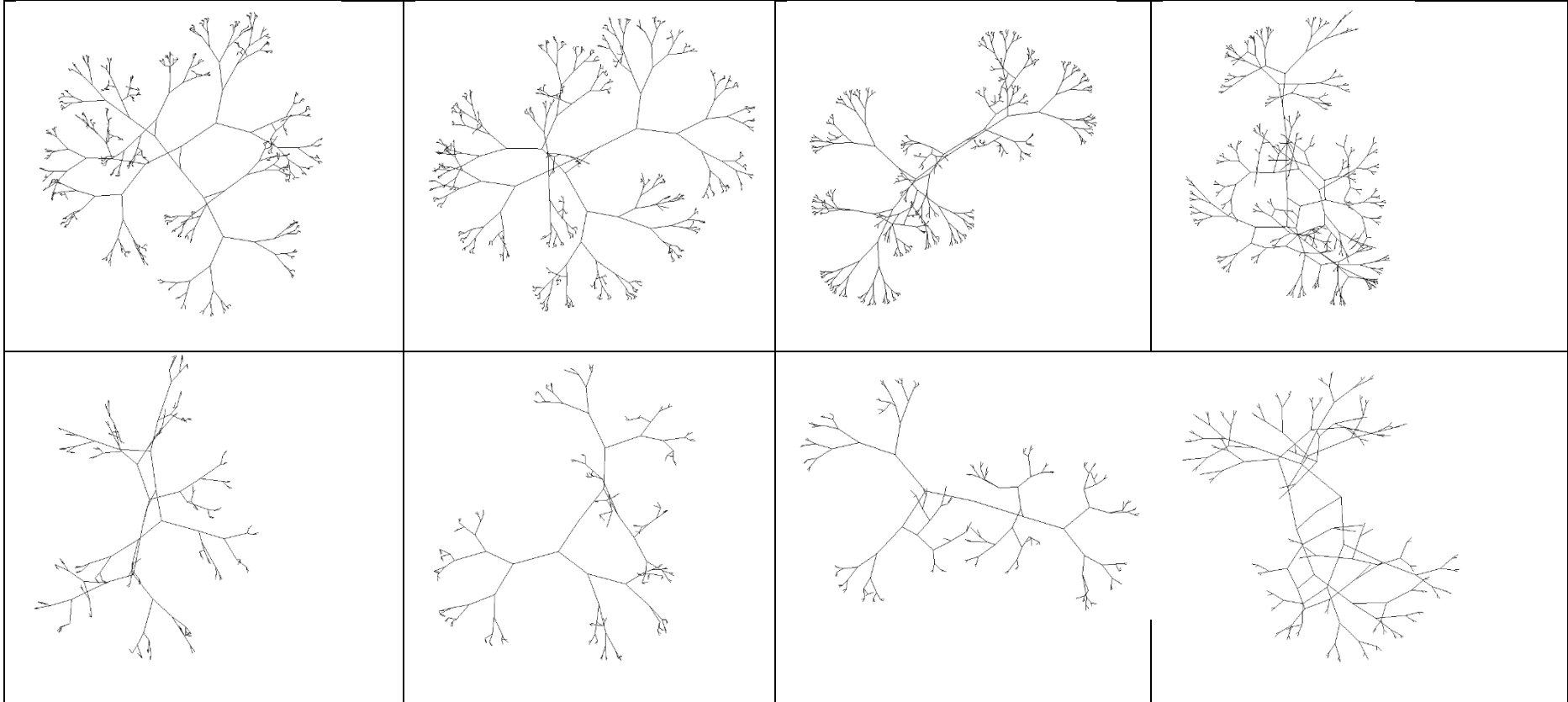


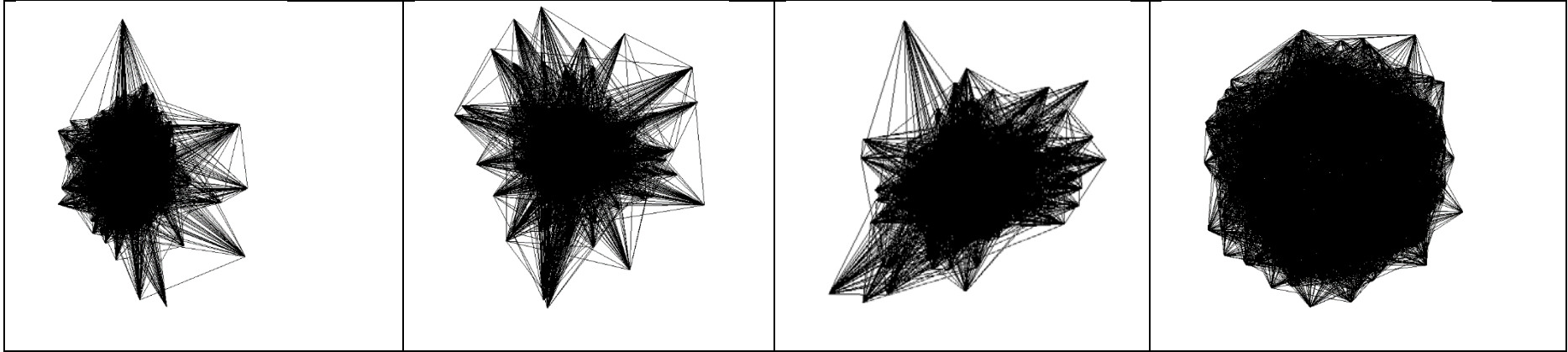
For all the graphs tested, values for the equilibrium of forces are sporadic and very much dependent on the type of graph, structure, degree of vertices and the density of edges. There does not appear to be one single setting which can be applied for all, however defaults are required, and so an average of 0.5 for *tr*

forces are suggested. Subjective analysis below identifies values which appear better quality or are easier to follow, and identifies the links and differences between the authors perception and the measured metrics for graph stability above.

0.3	0.5	0.7	0.9
			
			







10.6 Multilevel Aesthetics: Analysis of Layout Generation

In addition to regular-medium, analysis of edge crossings in original and coarse graphs for the graph sparse-medium suggest similar findings. The edge crossings exhibited in the layout for the original graph is shown in Figure 10.12, showing a similar slower convergence to a layout from *SE*. Due to difficulty reading the chart after frame 120, the chart is continued in Figure 10.13 showing the change in edge crossings for the last 150 frames. The chart shows the difference in edge crossings, confirming the above observation that *SE* provides layouts which suffer being trapped in minima.

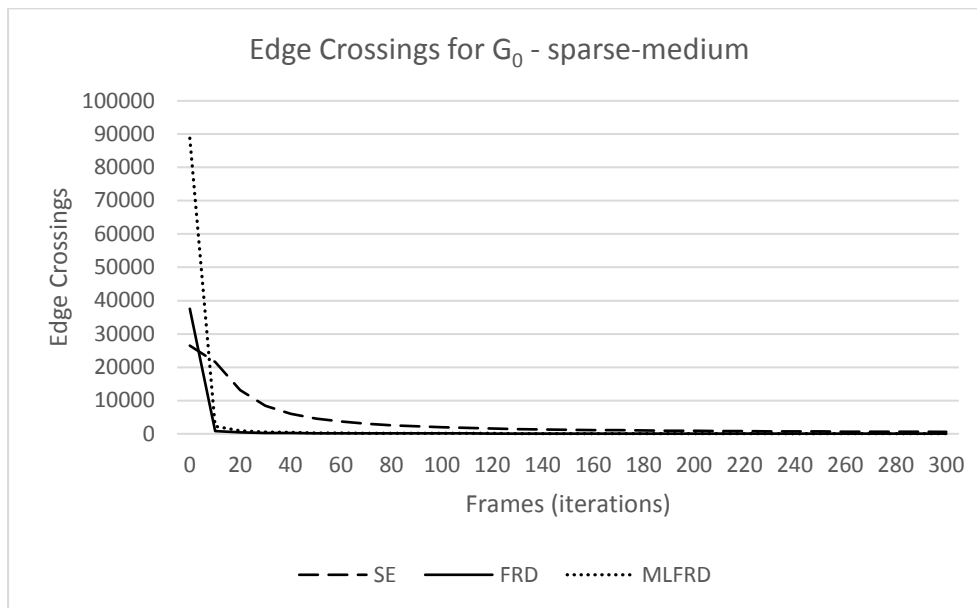


Figure 10.12 Comparison of the change in edge crossings for the graph sparse-medium over the course of 300 iterations using Eades Spring Embedder (*SE*), Dynamic Spring Embedder (*FRD*) and Multilevel Dynamic Spring Embedder (*MLFRD*)

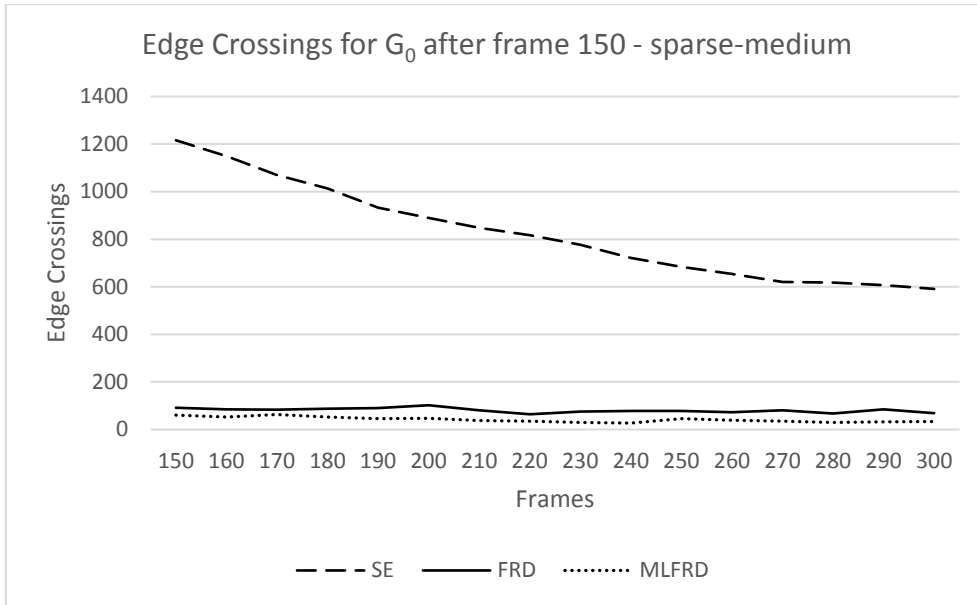


Figure 10.13 Comparison of edge crossings for the graph sparse-medium over 150 frames of iteration, after an initial 150 frames of force directed placement using Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD)

As above for regular-medium, multilevel analysis shows that the cause of the edge crossings is likely due to poor global layout, with many more edge crossings exhibited in an approximation of the graph than by the *FRD* and *MLFRD* methods.

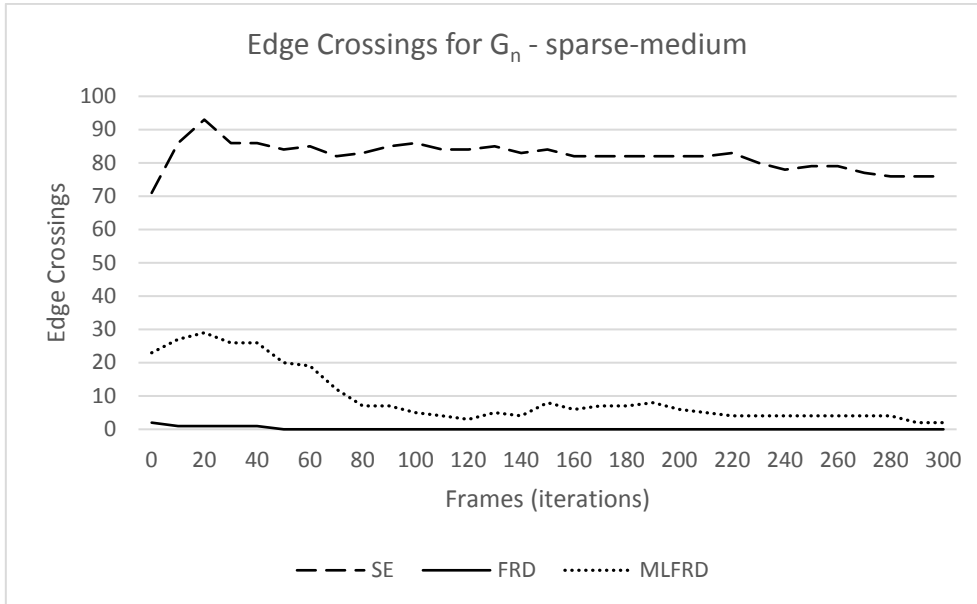


Figure 10.14 Comparison of edge crossings in layouts for a coarsened abstraction of sparse-medium, showing the global layout as generated for Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD)

Unlike the example graphs used above, *dense-small* is not expected to have a structured global layout due to the density of edges in the layout. Figure 10.15 shows that the *SE* method provides fewer edge crossings with the *FR* and *MLFRD* methods with a slower convergence (opposite to the behaviour above). Multilevel analysis in Figure 10.16 suggests differing results, with the *MLFRD* method providing improved global structure through reduction of edge in the coarser graph. In addition, the chart shows that the number of edge crossings exhibited in coarse layout through use of *SE* fluctuates, suggesting constant movement in the layout – to be expected in *dense-medium*.

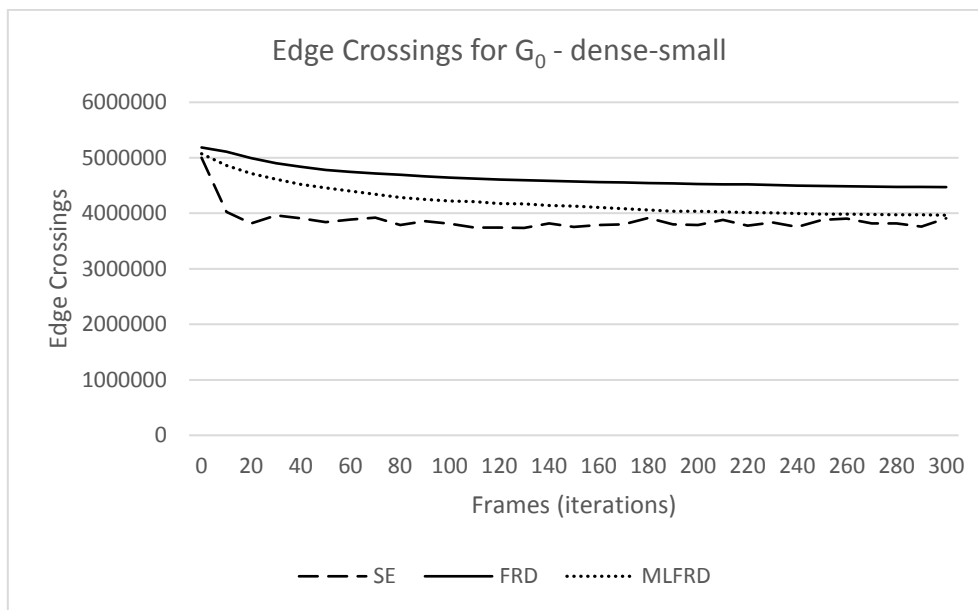


Figure 10.15 Comparison of the change in edge crossings for the graph *dense-small* over the course of 300 iterations using Eades Spring Embedder (*SE*), Dynamic Spring Embedder (*FRD*) and Multilevel Dynamic Spring Embedder (*MLFRD*)

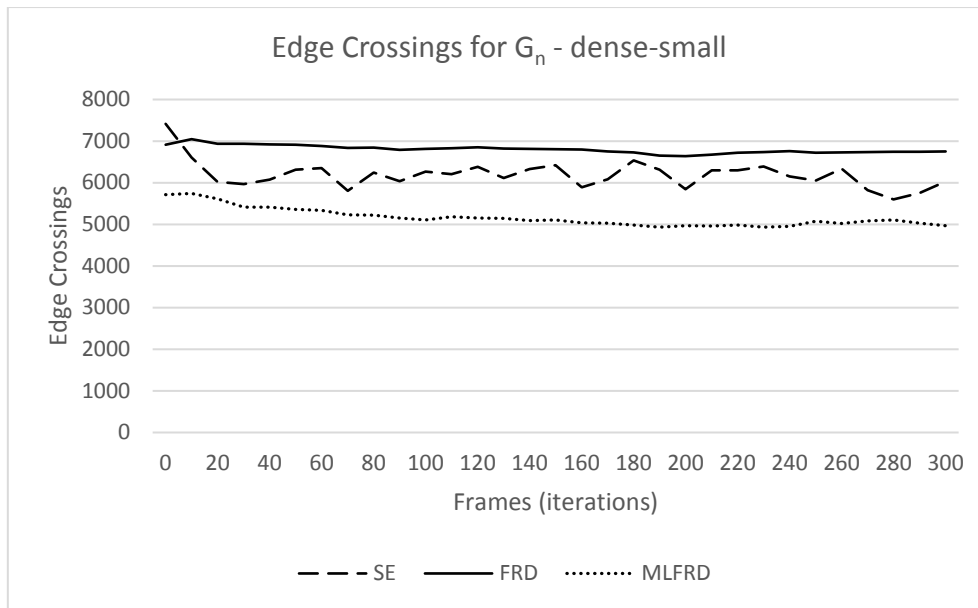


Figure 10.16 Comparison of edge crossings in layouts for a coarsened abstraction of sparse-medium, showing the global layout as generated for Eades Spring Embedder (SE), Dynamic Spring Embedder (FRD) and Multilevel Dynamic Spring Embedder (MLFRD)

Analysis of displacement between frames (average movement) can also be applied, however, due to differences between the algorithms a fair comparison cannot be made for the example used here (*SE* uses a value of 1.0 for ideal spring length k following description by Eades, 1984) whereas *FR* methods use a function of the drawing area, see Appendix 10.22). Multilevel analysis of vertex displacement is expected to provide irregular results due to differences in vertex position, therefore may only be beneficial to analyse the layout development of a singular graph (and not useful for comparative purposes).

10.7 Approximation Limit and Distance Limit in Calculating Repulsive Forces

In addition to approximation of global forces, the distance at which repulsive forces take effect is limited to remove calculation of weak and unnecessary interactions (Fruchterman and Reingold, 1991). The method results in fewer calculations by only calculating forces within a radius of a vertex. Similarly, experimentation here examines an “approximation limit” to restrict the traversal of the *MGF* tree during repulsive force calculation, limiting the distance at which repulsive forces take effect, using the connectivity of a graph as approximated by the multilevel scheme.

The method aims to answer the following questions;

Can a limit be imposed on the calculation of long range global forces, preventing interactions between weakly connected vertices? What is the effect on the quality layouts?

Use of an approximation limit assumes that vertices with a high graph conceptual distance between them (edges) are spatially distant within the layout, achieved by interpolation and refinement of coarse layouts. The limit is enforced by limiting the traversal of the Multilevel Global Force tree during force calculation, stopping further calculation when the limit is reached. For example; a limit of 3 means only three levels of the tree are traversed, preventing approximations made by further traversal. Details regarding Implementation can be found in Implementation.

Traversal of Multilevel Global Force means that the most distant approximations are achieved at the L^{th} level of the traversal (the number of graphs in the multilevel scheme (G_L)), and therefore a maximal limit is derived. The approximation limit is tested with incremental values between 1 and 30, with a value of 50,000 used for graphs which result in a multilevel scheme with more than 30 levels. Values larger than L result in the entire approximation tree being used as if no limit is in place.

The limit is compared to a spatial limit, r , using incremental values of k such that $r = n \cdot k$, where n is given values between 1 and 30. Limits are tested on static test graphs, with each value tested 10 times (repeats) for each graph. The results are analysed and averages compared to determine the change in running time and layout quality, identifying whether there is any benefit to using structural information to limit the distance at which forces take effect.

Results are collected using an un-weighted coarsener (one which does not organise by weight), and therefore the number of graphs generated for the multilevel scheme is higher than one which may use a weighted coarsener.

10.7.1 Numerical Results

Analysis is split into two parts, examination of the approximation limit, and comparison to spatial limitation.

10.7.1.1 Approximation Limit

Due to the large number of results, a general summary is provided here. Results are described as a percentage of those recorded from an algorithm using a value of 50,000 for the approximation limit, allowing them to be combined to form a general behaviour for all test graphs (for example, a graph with 200 edge crossings, which changes to 300 using a lower approximation limit, will be recorded as 50% increase – 150% in total).

Figure 10.17 illustrates the change in runtime as the value of the approximation limit and number of calculations is reduced. An initial drop between a value of 50,000 and 30 can be observed due to many of the graphs tested having a multilevel scheme with greater than 30 graphs. After the initial drop, the difference in runtime is steadily reduced until a lowest decrease of 70% is recorded for the smallest value of 3 (limiting to 3 approximations of repulsive forces per vertex – $O(3V)$).

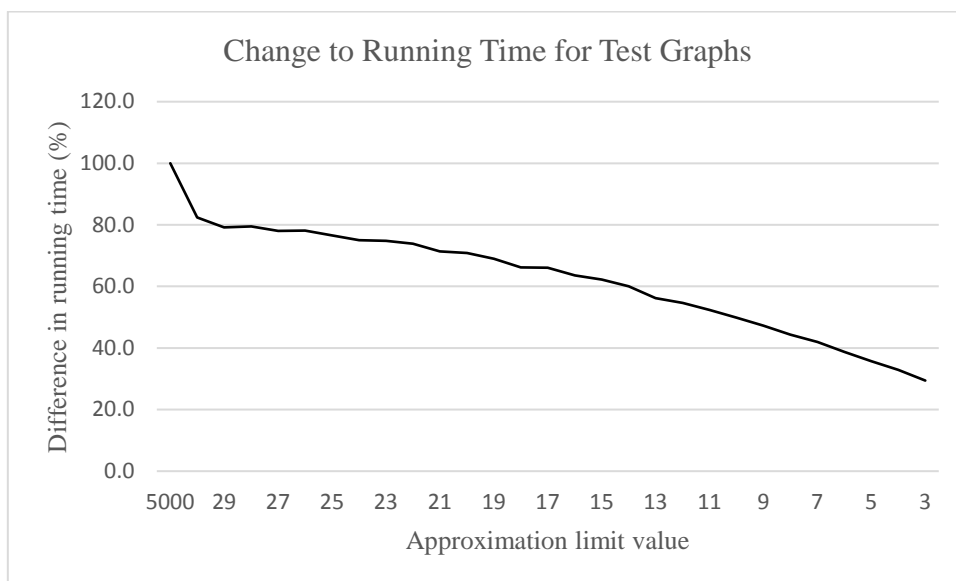


Figure 10.17. Average change in running time for the test graphs as a result of changes to the value of the approximation limit, with change measured as a percentage of the initial running time to provide a general comparable performance.

The near linear decrease in running time is expected due to the change in complexity, however the effect on layout quality is of greater interest due to the ability to reduce running time with marginal difference to layout quality for some values. The chart in **Figure 10.18** shows that as the value of the limit gets smaller, the quality of layouts gradually deteriorate until a value of 15, where after the number of edge

crossings increase dramatically from 40% increase to 500% increase (read as values of 140% and 600%).

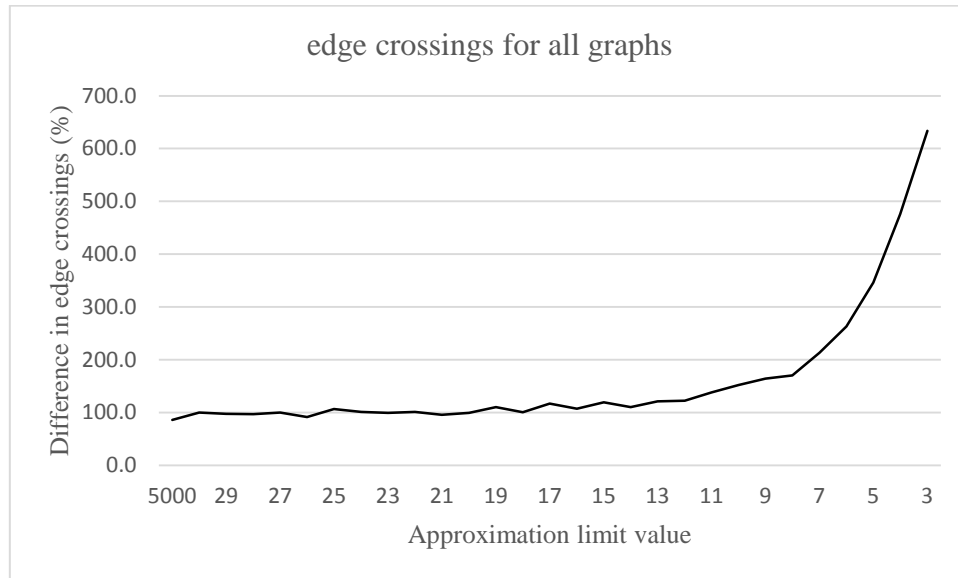


Figure 10.18. Average change of edge crossings for layouts generated for the test graphs as a result of changes to the value of the approximation limit, with change measured as a percentage of the initial running time to provide a general comparable performance

The charts suggest that generally, running time can be decreased by sacrificing layout quality, similar the R limitation used by Fruchterman and Reingold (1991), Walshaw (2003) and Hu (2005). Due to different graphs resulting in multilevel schemes with differing number of abstractions, analysis of one graph is also provided to add context to the usage. The chart in **Figure 10.19** shows the change in running time and edge crossings for the graph data, which is coarsened to only 17 graphs.

As a result, values for running time remain constant, with some changes identified for edge crossings (attributed to randomness across repeats). The results show that an approximation limit of 9 can be used to provide layouts of equal quality to layouts using a maximum value of 17 (or 50,000), providing a noticeable decrease in running time of 33.8%.

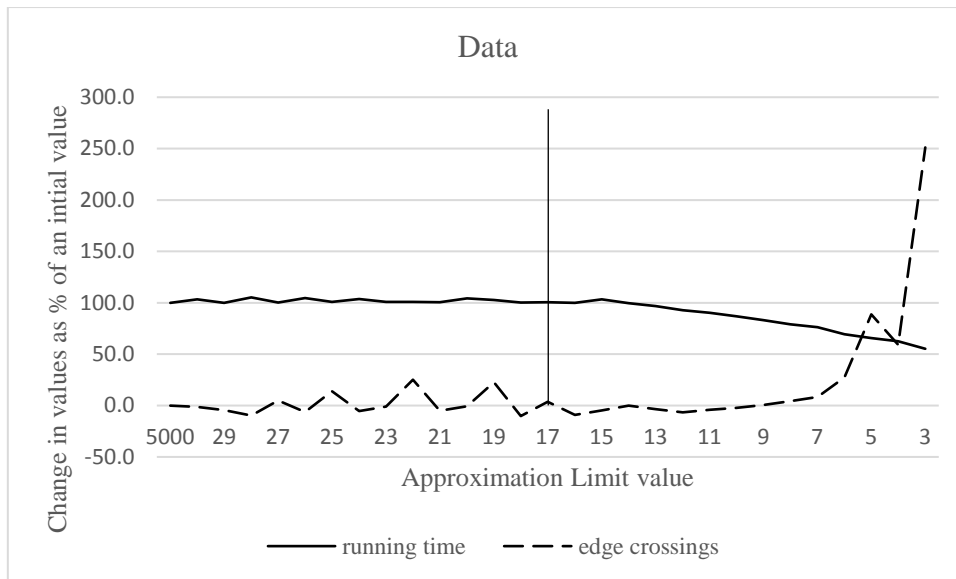


Figure 10.19. Change in edge crossings and running time for a test graph called data, showing the number of levels in the multilevel scheme (the maximum value of the approximation limit that will affect either running time or layout quality). Note the change in edge crossings fluctuates naturally, and the gradual change in both running time and edge crossings after the maximum value is reached.

10.7.1.2 Comparison to R

Similar to the analysis above, results are collected and generalised for comparison to a spatial limit of r used in existing algorithms (Fruchterman and Reingold (1991), Walshaw (2003), Hu (2005)). **Figure 10.20** shows the difference in running time between usages of r or an approximation limit.

The chart suggests that the saving of running time is more erratic and unpredictable for use of r than it is the approximation limit. The reason is down to differences in vertex positions relative to one another between the repeated experimentations, suggesting that use of approximation limit provides an added benefit of being able to accurately predict the reduction of running time and resulting changes to layout quality for different values.

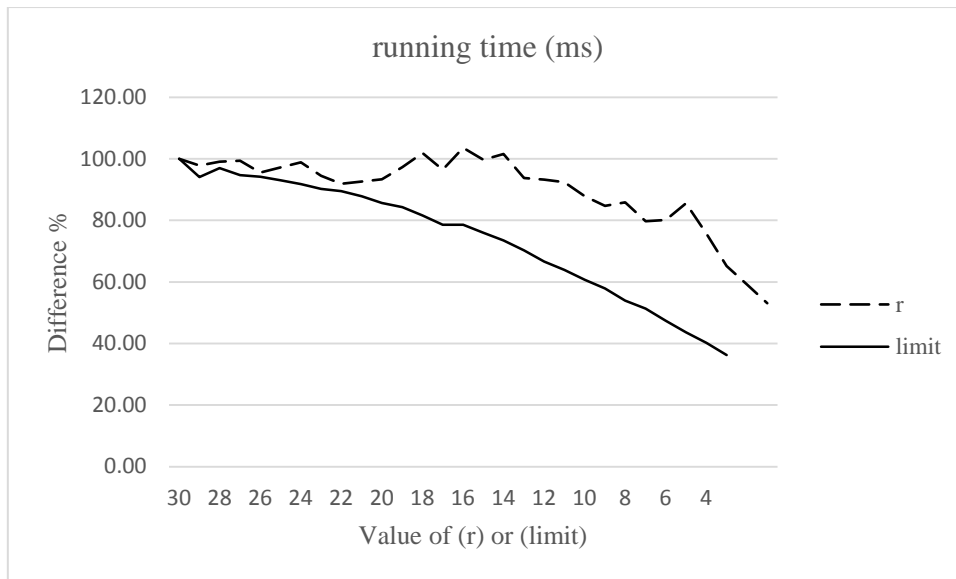


Figure 10.20. Average change in running time for the test graphs as a result of changes to the value of the approximation limit and spatial limit (r), with change measured as a percentage of the initial running time to provide a general comparable performance.

The difference in edge crossings show less erratic results, with both methods providing a gradual change in quality until a value of 15, where after the quality depreciates quickly. There is a noticeable difference in that r provides fewer edge crossings for smaller values than the approximation limit, due to the difference between using space or connectivity to approximate forces (the limit uses the connectivity of vertices as described by the multilevel scheme, whereby a value of 2 will result in 2 calculations, whereas a value of 2 for r may allow for many more calculations dependant on how many vertices are within $2 \cdot r$).

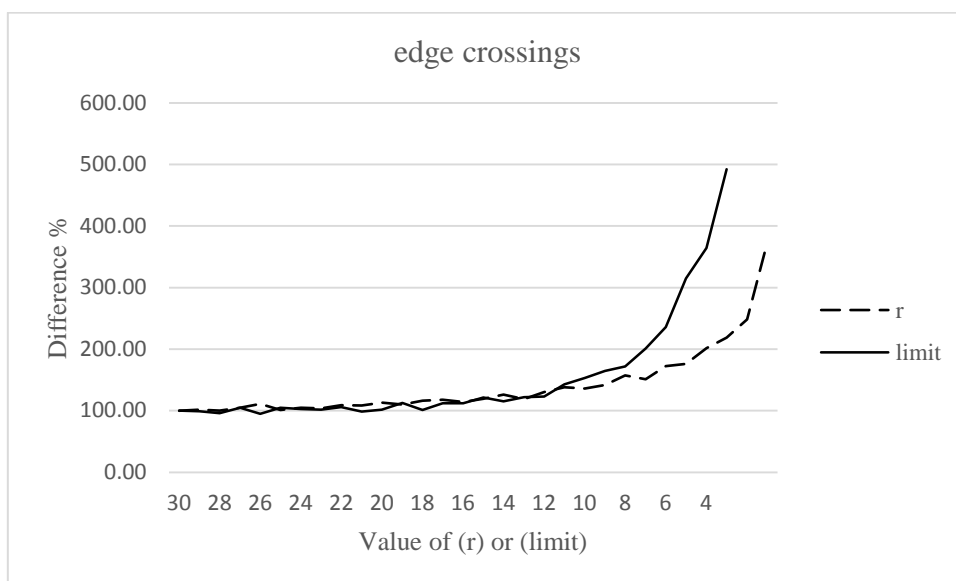


Figure 10.21. Average change of edge crossings for layouts generated for the test graphs as a result of changes to the value of the approximation limit and spatial limit (r), with change measured as a percentage of the initial running time to provide a general comparable performance

The results provided are general for the graphs tested, many of which result in over 30 levels in the multilevel scheme. For graphs with less, the same behaviour is expected however in reduced number of graphs and therefore the reaction of layout quality can be estimated before usage.

Use of approximation limit provides a maximum and minimum value for the limit, with a predictable decrease in running time (each value decreases traversal by 1, therefore removes $1 * (m-1)$ calculations), unlike r which is dependent on vertex positions relative to one another. Together the charts in **Figure 10.20** and **Figure 10.21** show that use of approximation limit is similar to use of spatial limits on repulsive forces, however, the rate of change is less for values of r due to differences between spatial and relational approximation.

10.7.2 Subjective Results

For both approximation limit and distance limit r , the effect on layouts is similar: the lower the value, the poorer the layout quality. **Figure 10.22** and **Figure 10.23** show the effect of approximation limit and r value on the graph 3025. There is little difference between the two, however, the change in quality is shown to correlate to the numerical analysis above, whereby lower values lead to more drastic deterioration of layout quality whereas as the values increase, the changes are more subtle.

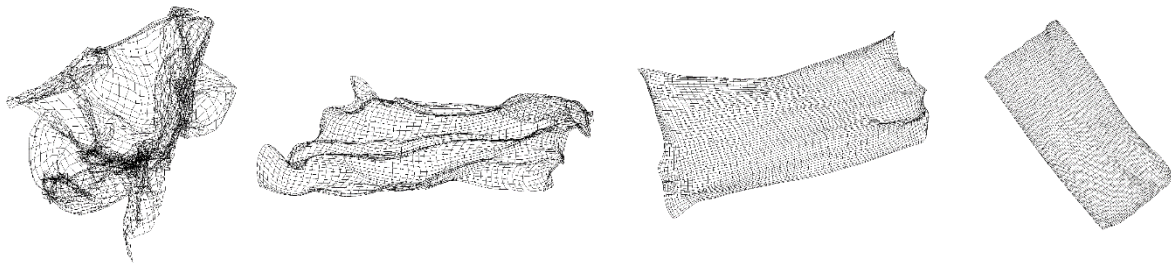


Figure 10.22. Approximation limit on layouts for 3025, showing the effect of using values of 2, 5, 7 and 10 for the approximation limit (left to right).



Figure 10.23. Spatial limitation (r) on layout for 3025, showing the effect if using values of 2, 5, 10 and 15 for the spatial limit (left to right).

The difference between the two is primarily the extent of the tangling for lower values of either limit. The approximation limit shows large amounts of tangling initially, whereas the lower value for r (Figure 10.22 – value of 2) is similar to that achieved using higher value of limit (Figure 10.22 – value of 5). Differences are a result of spatial distance (k) being different to relational distance, whereby a value of 1 for r can result in several calculations, whereas for limit it results in only 1 (dependant on relative positions).

Figure 10.24 shows the effect of approximation limit on global and local layout of sierpinski10, whereby smaller values result in some loss of global structure and a much more noticeable loss of local layouts. Typically one might expect a reduction in global layout before loss in local layout, however, due to layout being generated through a multilevel scheme (whereby distance between two vertices in the coarser graphs represents greater distances in finer graphs), global layout is still achieved. However, due to refinement being limited to small areas around each vertex, local layout is shown to be compressed whereby forces are not strong enough to expand the graph, causing folds and overlaps in the layout.

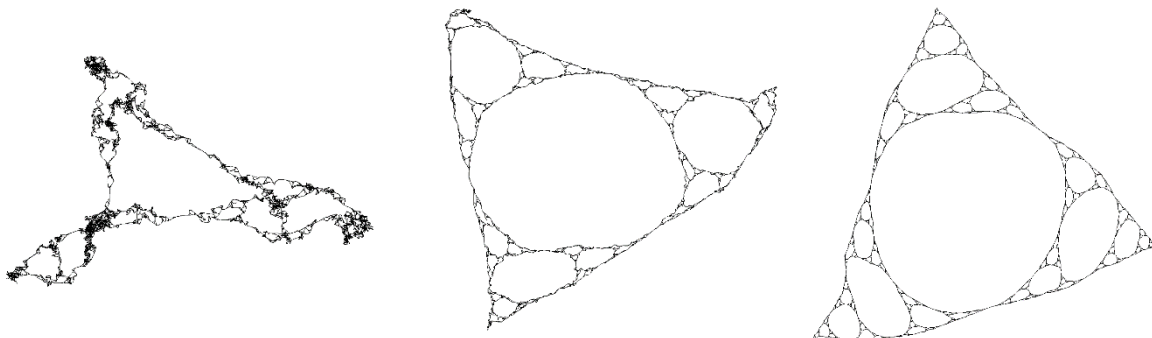


Figure 10.24. Approximation limit on layout for sierpinski10, using values of 2, 5 and 15 from left to right.

Similarly for larger graph, Figure 10.25 and Figure 10.26 show the effect of approximation limit and spatial limit r on the layouts for finan512. The layouts show improvement in global layout as the values increase and allow for greater number of repulsive force calculations, in addition to improvement on a local level.

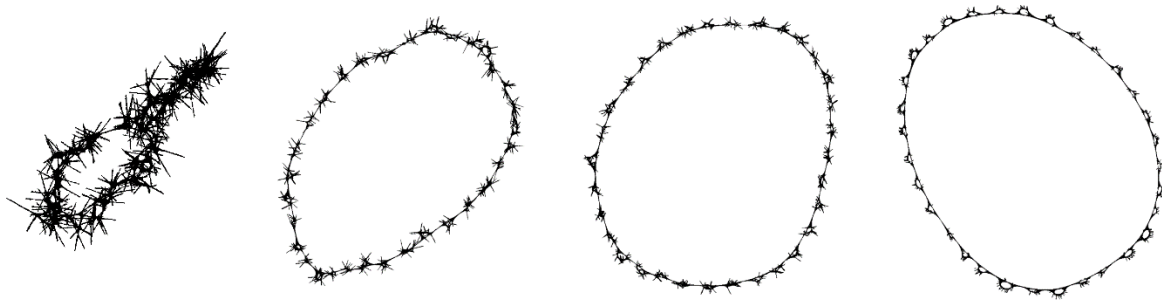


Figure 10.25. Approximation limit on layout for *finan512*, using values of 2, 5, 7 and 15 from left to right.

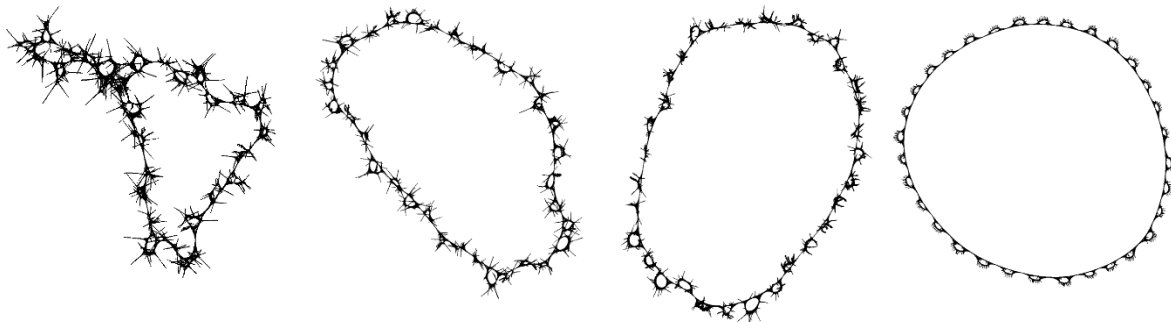


Figure 10.26. Spatial limitation (r) on layout for *finan512*, using values of 1, 2, 5 and 50,000 from left to right.

10.8 Brute Matching

The larger a graph, the more running time it takes to apply Force Directed Placement on it to generate a layout. The cause of this is due to the high number of calculations associated with the high number of vertices, and as such, reducing the number of calculations is key to reducing the running time.

One attempt at doing so is by reducing the number of calculations performed in the repulsive force calculation by reducing the size of the multilevel structure. For *dime20*, the number of graphs is extremely high as a result of brute matching, with 96% of consecutive graphs generated being having only 2 vertices different. If stopped at this point, only 19 graphs would be generated (an ideal place to utilise the coarsening tolerance). The number of graphs increases the height of the *MGF* tree and causes increased number of approximate calculations per vertex.

graph	max degree	M value	GL	rt
3025	4	5	19	74
data	17	18	5	92
add32	31	32	15	124
4elt	10	11	9	251

finan512	54	55	21	1296
sierpinski10	4	5	38	1064
dime20	3	4	545	2189
mesh100	4	5	446	2445

Table 10.2 Table showing the maximum degree of a selection of test graphs, showing the choice of matching value for Brute Matching and the resulting number of graphs in the Multilevel Scheme using the value. The running time for generating the multilevel scheme is also provided.

If the matching number m above is greater than the maximum vertex degree of a graph, the Multimatching scheme is in a state of “brute matching”. Experimentation of brute matching is performed to identify if such extremely coarse matchings have any benefit on the running time or layout quality of graphs, answering the following question;

The approach uses a matching number larger than the maximum degree of a vertex within the graph. Each test is run 10 times for each graph with analysis of both layout quality and running time with comparison to the best and worst case results of Multimatching and standard edge contraction methods. The expectation is a much coarser graph layout exhibiting poorer layout quality in comparison to standard edge contraction, however, reduced warping and running time as a result of Multilevel Global Force structure and multilevel size are anticipated.

Unlike the Multimatching method described in Appendix 10.20 , the method matches all unmatched vertices, and any unmatched vertices are matched with themselves (the method is not repeated with decreasing values of m).

10.8.1 Numerical Results

Results suggest that running time is extended when using brute matching, a result of the *MGF* tree being much broader and so increasing the number of repulsive force calculations. In contrast, the number of edge crossings is relatively similar to standard edge contraction, with some graphs having noticeable increases (data and add32 for example) and others have reduced edge crossings (sierpinski10 and mesh100 for example). Similarly, on average there is little difference between ranges of edge lengths.

Graph	Brute			Standard		
	RT	EC	ELR	RT	EC	ELR
3025	1362.0	4.0	0.3958	742.5	0.0	0.4963
data	913.8	43285.2	1.0979	788.0	36617.5	0.8887
add32	8613.4	20641.4	3.0049	1297.5	16240.0	2.2534
4elt	8806.4	23468.6	0.6695	5530.8	25695.0	0.7073
finan512	562617.3	4937331.0	1.0944	35595.0	5043279.0	0.9266
sierpinski10	55991.7	34034.7	0.3189	38768.3	37189.0	0.3399
mesh100	1492556.0	948369.0	0.4133	61184.0	979875.5	0.5329

dime20	3072137.0	141034.3	0.3207	87148.0	168011.0	0.5416
--------	-----------	----------	--------	---------	----------	--------

Table 10.3. Results comparing the running time (RT), edge crossings (EC) and range in edge length (ELR) as a result of using brute coarsening and standard edge contraction methods

The cause of the increase in running time for larger graphs can be attributed to the number of graphs generated for the multilevel scheme and the number of child vertices per coarse vertices. Overall there appears to be little benefit to using brute matching over standard matching techniques. Subjective analysis confirms similarity of layout quality to those achieved in multimatching, and suggest that some multimatching values provide more appealing layouts.

10.8.2 Subjective Analysis

For most of the graphs, the layouts are similar to those generated using Multimatching with a value greater than 2, exhibiting similar expansion as a result of approximation in multiple directions. **Figure 10.27** and **Figure 10.28** provide examples of layouts generated for 3025 and 4elt, with comparison to those generated using standard edge contraction (m2) and an alternate layout generated using Multimatching (values of 6 and 7 for 3025 and 4elt respectively).

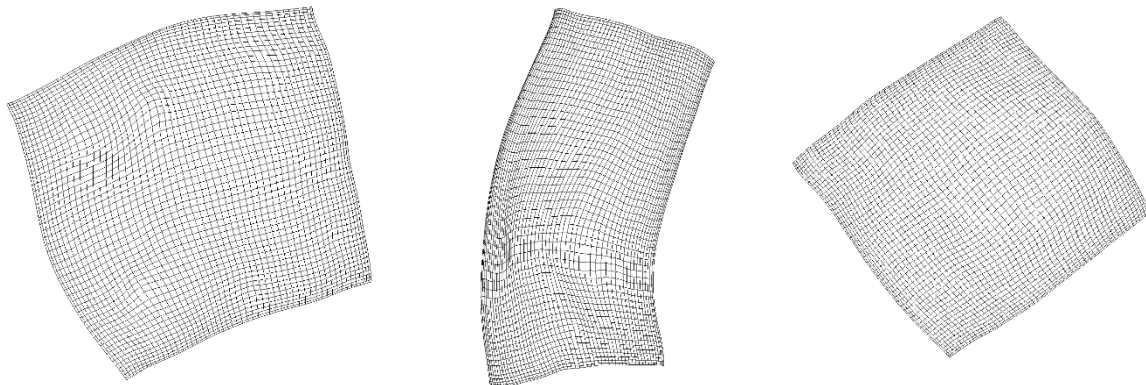


Figure 10.27. Layouts for 3025 generated using differing matching techniques: brute matching (left), standard edge contraction (centre) and multimatching with a matching value of 6 (right).

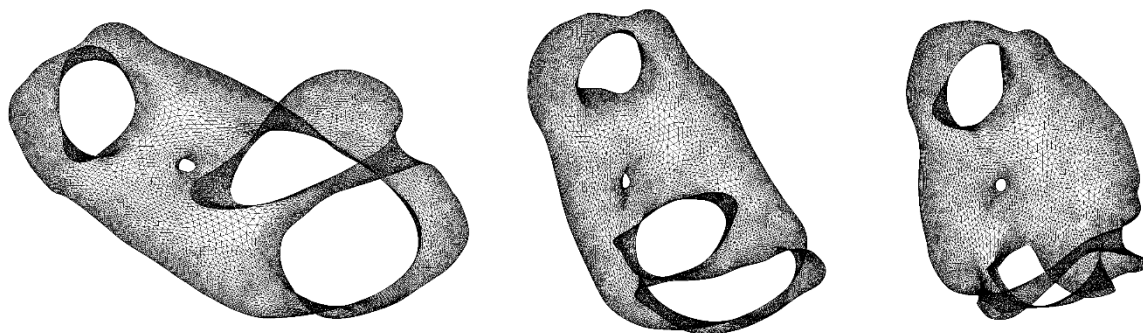


Figure 10.28. Layouts for 4elt, using differing matching techniques: brute matching (left), standard edge contraction (centre) and multimatching with a matching value of 7 (right).

It may be noted that for 3025 and finan512, the layouts generated using brute coarsening feature less warping and contraction as a result of *MGF* structure. However, these layouts are accompanied by extended running times, and are not guaranteed for all graphs, as shown for finan512 in **Figure 10.29** which shows poor global layout generated through brute coarsening (despite numerical analysis identifying reduced edge crossings).

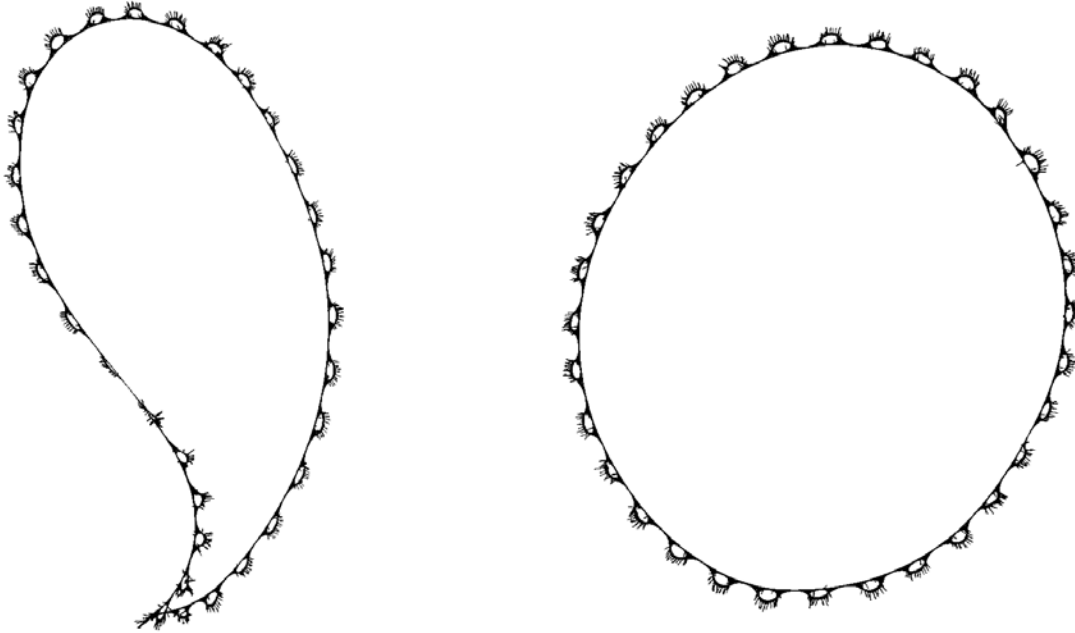


Figure 10.29. Layouts for finan512, using differing matching techniques: brute matching (left) and multimatching with a matching value of 7 (right).

In general, there is little regular improvement over standard edge contraction, and the extended running time makes usage unattractive.

10.9 Leaf Placement Scheme

Primitive Coarsening is investigated by experimenting with a simple primitive, a leaf. A leaf is defined as a vertex with a degree of 1, meaning it is connected to only one other vertex, and therefore can be given a known layout with the edge pointing outwards from the centre of mass (repulsive forces repulse vertices outwards, the single spring connecting it to the graph would prevent it moving away too far). In addition to this, a secondary rule of “vertex chains” is used, a collection of vertices connected to one another to form a chain of vertices, which similarly, can be given a known layout. **Figure 10.30** provides an example of leaf vertices (coloured black) attached to a small example graph (left), with an example of chains attached to the same graph (right). Primitive coarsening contracts these primitives with the anchoring vertex generating a coarsened vertex with weight equal to the weights of the combined vertices.

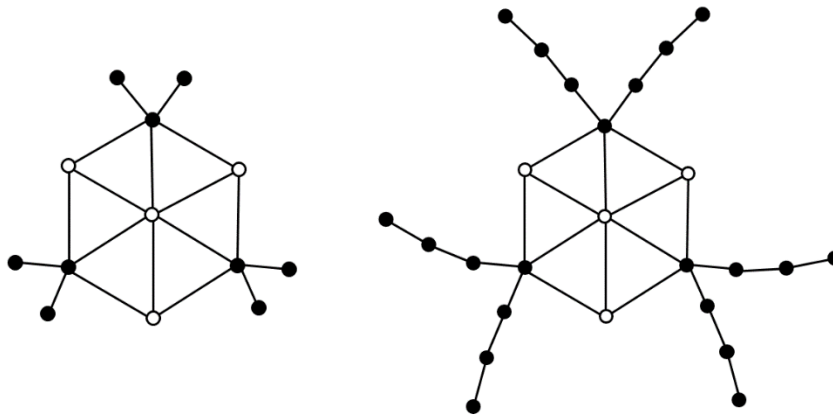


Figure 10.30. Example of leaf vertices (left) and chains (right) exhibited on an example graph.

It should be noted that although chains may appear within a graph, if connecting two clusters of vertices, the length of the chain is regarded as an important feature which should be preserved. Experimentation specifically identifies “loose chains”, those which are anchored on one end and are less connected to the structure of the graph.

The use of primitive processing is included as part of the coarsening phase, whereby before coarsening takes effect, a graph is first reviewed to identify any primitives. If found, they are coarsened with their “anchoring vertex”, generating a new coarsened graph. Coarsening continues as normal thereafter, treating the primitive-free coarser graph as the new original. During layout generation, graphs generated through primitive coarsening are processed with leaf vertices given a calculated layout using a Vertex Placement Scheme (VPS), see Implementation 10.22 for more information.

Four algorithms are used to test the effects of primitive coarsening;

- Algorithm A - Leaf pre-processing of the original graph only applies the leaf coarsening as a pre-process on the original graph (G_0), generating a new graph which is then treated as the original. After applying force directed placement, the layout is passed back to the original graph and leaves are given layout using the VPS.
- Algorithm B - Leaf pre-processing of the original graph as above but with application of Force Directed Placement after leaves have been given an initial layout from the VPS.
- Algorithm C - Leaf pre-processing at each level of the coarsening scheme as the graph is being coarsened. As the graph is being uncoarsened, force directed placement methods are applied at each level to the processed graph and then the VPS is used to place the leaf vertices (as in Algorithm A but on all graphs in the multilevel scheme).
- Algorithm D - Leaf pre-processing at each level of the coarsening scheme with force directed placement applied after leaves have been given an initial layout using the VPS (as in Algorithm B but on all graphs in the multilevel scheme).

Testing is primarily performed on the leafy test graphs described in Experimental Methodology 10.22.1, with testing on some static test graphs for impact on non-leafy graphs. Each of the algorithms above is tested on each of the graphs and repeated 10 times. Results are analysed and compared to determine the effect of the algorithms on layout and running time for graphs with few leaf vertices and chains, and on graphs featuring many.

Two sets of analysis provided;

- which if any of the methods provide improved layout of the graph – that is, does any of the methods provided improved layout of the graph without leaves (does the coarsening impact layout at all)
- Use of the VPS to provide good layout for vertices, been discussed many places before and is more specialized for leafy graphs (the aim here is general graphs), but impact on multilevel scheme is of interest, and running time

10.9.1.1 Numerical Results

Graph layouts are analyzed in two steps, firstly, the effect of the placement algorithms on running time, and secondly the effect of the algorithms on layouts which have had leaves coarsened. The reason for this is that the placement algorithm can control edge length and positioning of leaf vertices, and can therefore be altered to reduce edge crossings and alter peripheral effect.

10.9.1.1.1 Running Time

Collection of running time shows that on average, all algorithms increase running time, the primary cause of which is the additional work required to identify and coarsen leafy graphs, and the resulting increase in approximate graphs being generated. For example, given an approximate graph of 205 vertices, 5 of which are leaves, primitive coarsening will coarsen those 5 leaves and generate a new graph of 200 vertices. Typical coarsening may coarsen those leaves in addition to the rest of the graph resulting in a graph with at least 103 vertices.

Some leafy graphs however, show a reduction in running time, achieved by replacing Force Directed Placement with the Vertex Placement Scheme. In particular, sn6121 shows a large decrease in running for each algorithm, a result of instant layout being provided for a large number of leaf vertices, suggesting that a graph which is identified as having many leaves can have running time reduced using the algorithms described.

Graph	Running Time (ms)				
	0	A	B	C	D
3025	557.8	911.6	1194.4	872.4	1056.2
data	841.2	918.8	1296.6	1012.8	1094.6
add20	597.8	869.6	1206.8	797.4	1287.6
add32	2002.2	1816.0	2428.2	1929	3582.6
4elt	7778.8	7829.0	11179.2	8790.8	8097.2
finan512	47227.2	68664.4	78442		
uk	1763.2	1572.6	2130.8	2330.2	4075.8
anna	75.4	50.6	112.8	69.8	122
david	39.0	33.6	53.6	34.6	47.6
dyads_lc	73.6	39.0	71.6	28.2	70.4
dyads2_lc	166.8	104.0	191.2	102.8	156
gd96D	99.4	75.6	110.8	40	63.4
websiteCMS	234.6	223.2	341.6	272	320.8
sn6121	6289.8	2288.6	3118.2	2704.4	4433

Table 10.4. Comparison of running time for generating layouts using different Pattern Processing algorithms (algorithms A-D) and standard edge contraction (algorithm 0)

Due to the increase in running time for graphs without leaves, none of the algorithms appear beneficial for reducing running time in general, suggesting usage through some decision during the coarsening scheme (for example, if a graph has a large number of leaves relative to its size, use Algorithm A – or other at users request). The usage however is dependent on the quality of layouts generated.

10.9.1.1.2 Layout Quality

Layouts are analysed in two parts: quality of layouts for leafless graphs (top separation of **Table 10.5** and quality of layouts for leafy graphs (bottom separation of **Table 10.5**).

Due to leafless graphs not being given layout by the Vertex Placement Scheme, analysis is not provided for Algorithms A and B (however, running time above still increased due to identification of leaves used in the coarsening scheme). Algorithms C and D show little difference in edge crossings, with 1.6% increase and 4.8% decrease respectively, values which can occur naturally through *FDP* (for example, Algorithm A provides layouts with 23.3% decrease in edge crossings, even though layout is provided using standard *FDP*).

For the leafy graphs, Algorithm A is shown to provide layouts with 47.7% more edge crossings, with Algorithm B showing a decrease of 11.4%. This suggests that the VPS places vertices such that leaf vertices cross parts of the graph, however, refinement allows for improved position of leaf vertices and compression of connecting edges. A modified algorithm using local centers of mass (instead of the global centre of mass), may provide improved layout.

Algorithms C and D provide similar results, with a 58% increase and 7.5% reduction in edge crossings respectively, supporting the suggestion that placement by VPS followed by refinement by *FDP* is beneficial for layouts.

Graph	Edge Crossings				
	0	A	B	C	D
3025	0.0	0.0	0.0	2.4	0.0
data	41243.0	41673.8	39094.8	42738.0	42140.4
4elt	26396.4	20514.6	22483.4	26296.4	23339.0
finan512	4766745.2	4792462.8	4691131.6		
add20	556737.6	623590.6	503615.2	627530.8	458563.8
add32	11731.8	13642.4	10902.8	16408.6	11320.2
uk	697.8	650.0	600.4	689.8	353.4
anna	11512.0	16983.0	7760.8	15195.0	8987.8
david	11203.0	12287.8	6944.0	12901.0	6833.2
dyads_lc	11.4	45.6	9.8	51.8	18.8
dyads2_lc	16101.6	17093.8	15426.6	19940.6	14808.2
gd96D	217.8	323.8	211.4	307.6	214.2
websiteCMS	66420.2	94431.6	66673.8	102049.0	66461.6
sn6121	2039964.2	2069559.2	2193063.0	2203736.0	2069184.8

Table 10.5. Comparison of edge crossings for layouts generated using different Pattern Processing algorithms (algorithms A-D) and standard edge contraction (algorithm 0)

Similarly to above, the range in edge length is not affected by Algorithm A or B for the leafless graphs as they do not exhibit leaf vertices (and are therefore unaffected). Algorithms C and D however show a significant decrease of 23.9% and 25.5% respectively, suggesting that by placing leaf vertices in coarser graphs using the VPS (alone or for refinement) decreases the range in edge length and possibly the Peripheral Effect (see Subjective Analysis for confirmation). In contrast however, leafy graphs are shown to extend the range in edge crossings by 18.8% for both Algorithm A and B, and 14.6% and 4.9% for Algorithms C and D.

Graph	Range in Edge Length				
	0	A	B	C	D
3025	0.4304	0.3771	0.4296	0.4366	0.3881
data	0.8815	0.7921	0.9059	0.8921	0.8933

4elt	0.6715	0.6947	0.6793	0.6834	0.7160
finan512	0.7687	0.7750	0.8015		
add20	0.8976	0.9165	0.8703	0.8275	0.7987
add32	3.0819	2.8898	2.9289	2.4215	2.4846
uk	0.5015	0.4630	0.4904	0.5189	0.4908
anna	0.8991	0.9807	1.0708	0.9258	0.7130
david	0.4016	1.1817	0.9701	1.1003	0.8630
dyads_lc	1.3372	1.1844	1.4343	2.0904	1.9929
dyads2_lc	1.3633	1.3378	1.5624	1.1844	1.0838
gd96D	1.0321	0.9999	1.1554	0.9903	0.7885
websiteCMS	1.2028	1.3402	1.1230	0.7999	1.0430
sn6121	1.3427	1.3660	1.4845	1.1944	1.2792

Table 10.6. Comparison of the range in edge lengths for layouts generated using different Pattern Processing algorithms (algorithms A-D) and standard edge contraction (algorithm 0)

Although layouts are important, the aim here is to alter the coarsening scheme and simplify graphs for multilevel layout refinement and optimising the Multilevel Global Force structure. As such, layouts with leafy vertices removed are investigated, provided in **Table 10.7**.

The number of edge crossings identifies that for many of the graphs, Algorithm D and Algorithm B provides consistently improved layout quality (11.4% and 10.8% reduction in edge crossings respectively) for the main body of the test graphs. This suggests that the coarsening of leaf vertices may improve the representation of the graph structure, however, refinement using *FDP* is required to reduce edge crossings resulting for placement through the VPS (resulting in 28.8% and 55% for Algorithms A and C).

Graph	Edge Crossings of G0 with leaves removed				
	0	A	B	C	D
3025	0.0	0.0	0	2.4	0
data	41243.0	41673.8	39094.8	42738	42140.4
4elt	26396.4	20514.6	22483.4	26296.4	23339
finan512	4766745.2	4792462.8	4691132		
add20	556526.8	620978.4	503464.2	626324.6	458362.4
add32	11659.2	13119.4	10841.6	16220.6	11259.2
uk	692.2	641.4	598.4	678.4	351.4
anna	9779.0	16343.6	6601.2	14486.4	7323.8
david	9929.0	11273.0	5555	11741	5958
dyads_lc	6.2	19.0	5.6	33.6	7.8
dyads2_lc	14892.2	16898.8	13725.8	19090.6	13619.6
gd96D	174.4	244.0	177.6	207.6	161
websiteCMS	65466.8	93096.0	63187	97803	65096.4
sn6121	2008486.0	1928729.6	2126240	2067915	1984830

Table 10.7. Edge Crossings exhibited in the main connected body of graphs (graphs with leaf vertices removed).

10.9.1.2 Subjective Analysis

10.9.1.2.1 Algorithms A and B

As above, Algorithms A (Leaf placement on the original layout only) and B (Leaf placement on the original graph with *FDP* refinement) are analysed only in regard to leafy graphs due to leafless graphs not being given layout by the VPS. Leaf processing is specific for leafy graphs for these two algorithms, however, the analysis here aims to look at the effect on global layout and readability in a general sense – specifically impact on the multilevel scheme and representation of graph structure.

The layouts for the graph *dyad_lc* has some of the most noticeable reactions to the Vertex Placement Scheme due to its high number of leaf vertices and small size of the graph, making placement of leaves more noticeable, as shown in **Figure 10.31**. The layout provided by Algorithm A (**Figure 10.31** (left)) shows the noticeable placement of leaves facing outward from the centre of mass, making the leaves clearer than the layout provided by standard edge contraction in **Figure 10.31** (right). The layout provided by Algorithm B appears as a mid-way layout of Algorithm A and standard edge contraction, providing leafy layout which has been contracted as a result of *FDP*.



Figure 10.31. Comparison of layouts for *dyads_lc*, using leaf coarsening algorithms A (left), B (centre) and standard edge contraction (right).

Larger graphs appear lesser effected by Algorithm A and B, with leaf placement less noticeable for *add32* as depicted in **Figure 10.32**. A noticeable difference between the algorithms is the spread of the layouts, with a wider spread of leaves and chains exhibited in layouts generated by standard edge contraction.

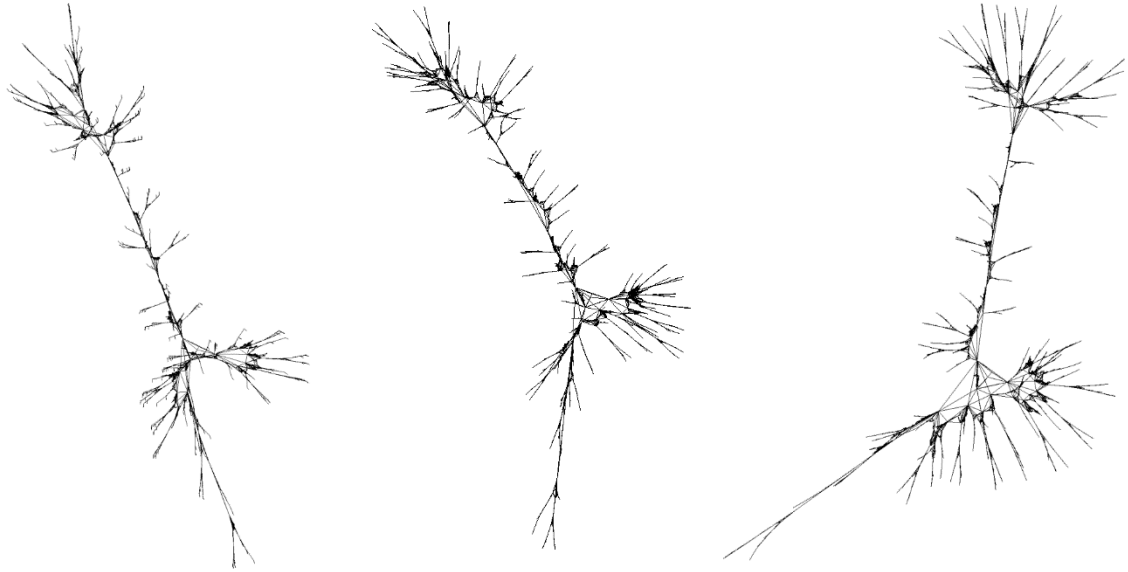


Figure 10.32. Comparison of layouts for add32, using leaf coarsening algorithms A (left), B (centre) and standard edge contraction (right).

In contrast to the examples above, layouts provided for the graph sn6121 show similarity between Algorithm A and standard edge contraction (albeit rotated), with both layouts showing a dense network of edges with a concave shape. In contrast, layouts provided by Algorithm B show a different shape.

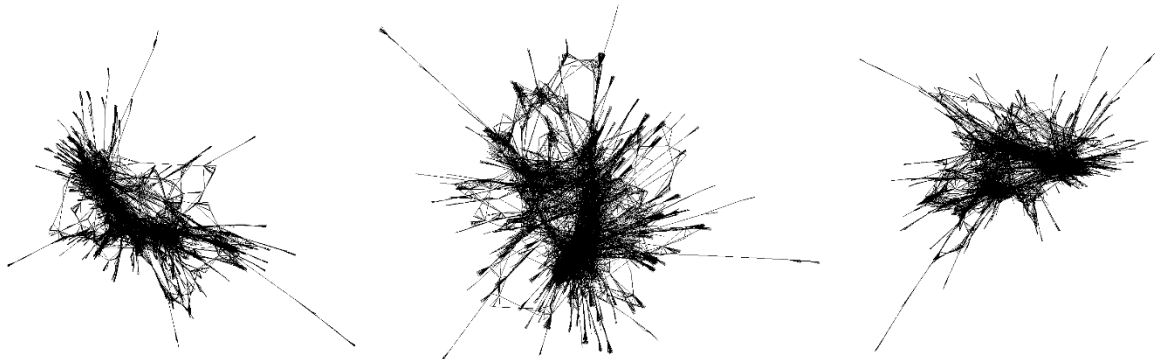


Figure 10.33. Comparison of layouts for sn6121, using leaf coarsening algorithms A (left), B (centre) and standard edge contraction (right).

Overall, the effect of the algorithms is different between graphs.

10.9.1.2.2 Algorithms C and D

Algorithms C (multilevel leaf placement) and D (multilevel leaf placement with *FDP* refinement) are based on the multilevel scheme and therefore are analysed in regard to the effect on global layout more

than visualisation of leaf vertices. The effect looks to identify any reaction in layouts caused by the processing, expected to result in changes in edge crossings and ranges in edge length.

Using the same examples above, use of the multilevel scheme can be seen to reduce the impact of leaf based layout on smaller graphs. In particular, *dyads2_lc* is shown to have less noticeable leaf placement than no leaf placement at all, a result of continuing leafs between levels of the multilevel scheme (that is, placement of leaf vertices as chains). The resulting layout is shown in **Figure 10.34** (left), whereby chains are drawn as elongated edges. Similarly to above, Algorithm D provides some refinement using *FDP*, however, due to the limited application, the layout still resembles that generated by Algorithm C.

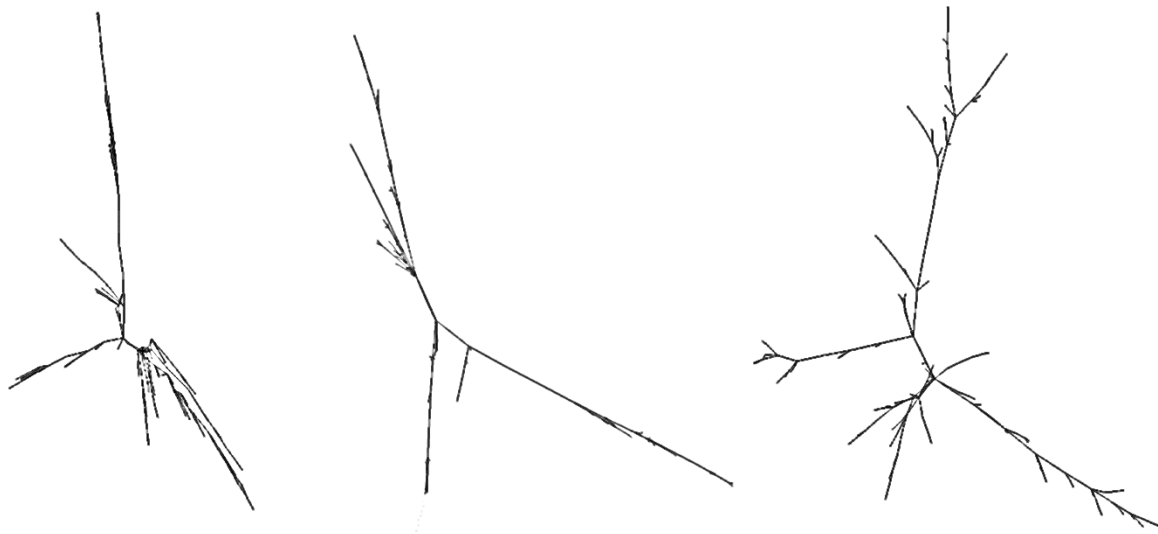


Figure 10.34. Comparison of layouts for *dyads2_lc*, using leaf coarsening algorithms C (left), D (centre) and standard edge contraction (right).

In contrast, larger graphs which have a more connected body such as *add32* and *sn6121* in **Figure 10.35** and **Figure 10.36**, show layouts which include improved expansion of branch structures (see **Figure 10.35** (left) or highlight some structure not seen in layouts provided by standard edge contraction (see **Figure 10.35** (left) compared to **Figure 6.45** (right)). Additional layouts are provided in Appendix 10.9 further suggesting that the use of leaf placement in the multilevel scheme provides layouts which are subjectively easier to read.

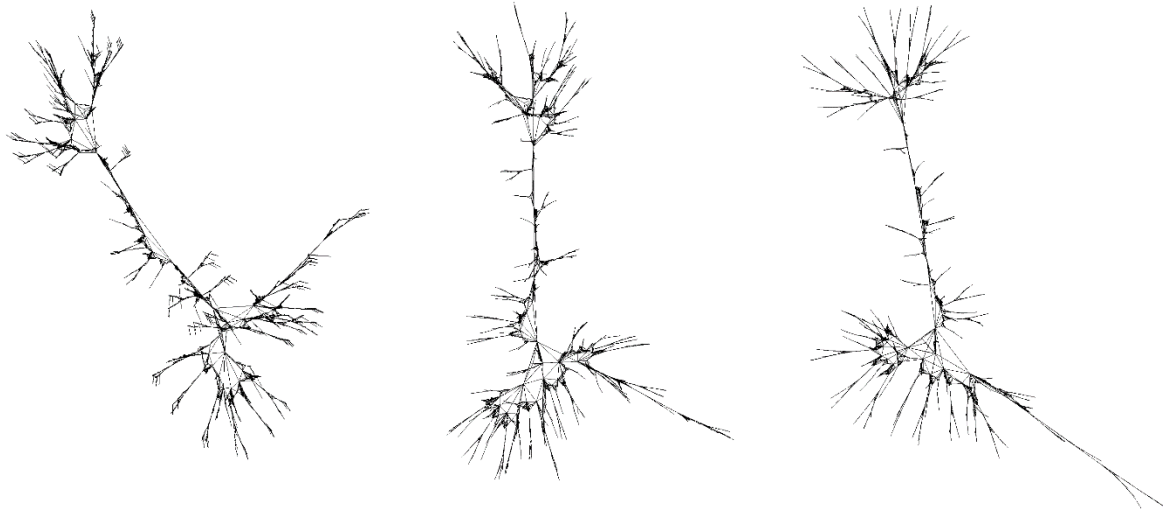


Figure 10.35. Comparison of layouts for add32, using leaf coarsening algorithms C (left), D (centre) and standard edge contraction (right).

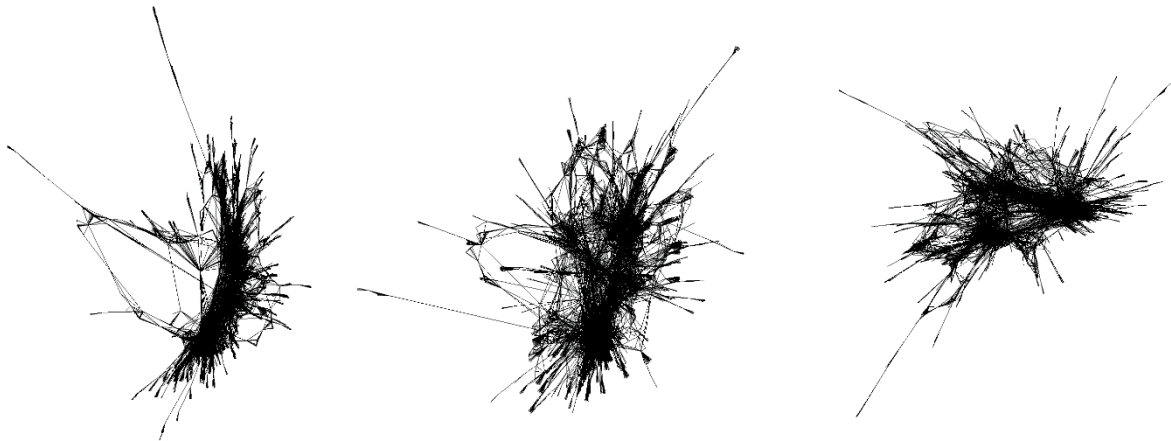


Figure 10.36. Comparison of layouts for sn6121, using leaf coarsening algorithms C (left), D (centre) and standard edge contraction (right).

10.10 Primitive Graph Coarsening

It is expected that graphs with differing structures are likely to be coarsened to different primitive graph types, for example, it was expected that `finan512` would coarsen to a ring due to its ring-like global structure (see **Figure 10.38**). As such the methods used to provide initial layout are recorded and given in **Table 10.8**. The star primitive was the most common method used, with none of the tested graphs being coarsened to a chain. It may be noticed that `finan512` is also coarsened to a star primitive, showing that the global ring structure is not preserved in the coarsest graph.

	Primitive Usage (of 10 repeats)			
	Star	Ring	Chain	<i>FDP</i>
3025	8	2	0	0
data	8	0	0	2
add32	10	0	0	0
4elt	8	0	0	2
dime20	9	1	0	0
finan512	10	0	0	0
mesh100	9	0	0	1
sierpinski10	7	0	0	3

Table 10.8 Which of the Primitive Graphs each of the test graphs are coarsened to, showing that many graphs coarsened down into the star primitive, with few coarsening into Ring or used standard *FDP*. None coarsened into a Chain.

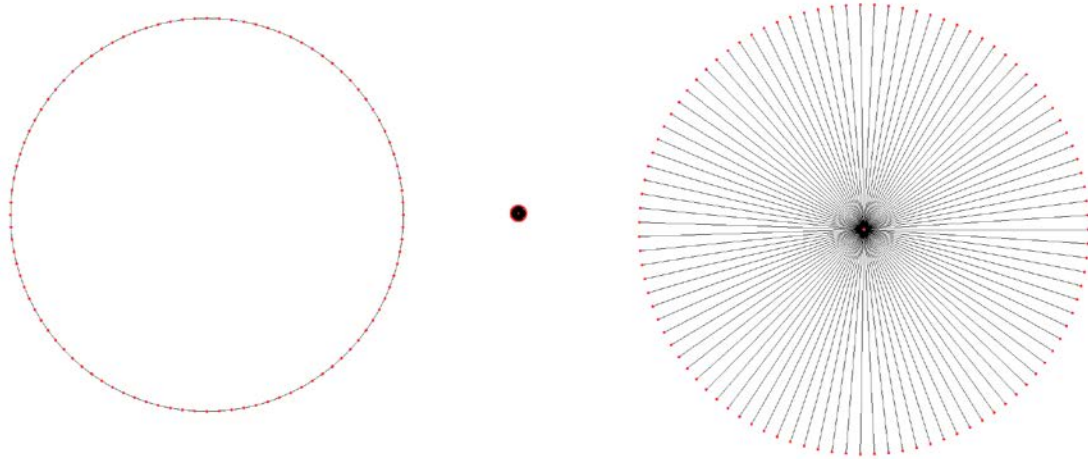
Graph	$ G_L $	$ G_I $
3025	12.8	2.2
data	12	3
add32	15.6	3
4elt	15.2	3
sierpinski10	18.4	3.6
finan512	17	3.4
dime20	19.6	2.6
mesh100	17.4	2.6

Table 10.9 The number of graphs generated in the multilevel scheme and the size of the coarsest graph, showing that even though primitives were used, the coarsest graph is still coarse

The results show little change between standard coarsening and use of primitives, a result of the effectiveness of the standard weighted coarsener. By changing the coarsener or using Multimatching, methods known to increase the number of graphs in the multilevel scheme, the value of primitives can be seen. For example, by removing the function which organises a graph by weight, graphs such as `add32` are coarsened to provide a multilevel scheme with many more levels in it (weighted coarsener

may produce 12-15 graphs, whereas coarsener alone provides between 100 to 120). The result is primitive processing is able to identify larger primitive graphs (for add32, a star of 78 vertices can be identified).

Firstly a look at some examples of Star and Ring placement for a graphs of 100 vertices.



Experimentation uses three example primitive graphs;

- Star – a central vertex with degree $(|V|-1)$ connected to all other vertices in the graph, each with degree of 1, connecting in to the centre
- Ring – a ring of interconnected vertices, each with degree of 2
- Chain – graph is coarsened down to a chain of vertices, each with degree of 2, except the ends which have degree of 1

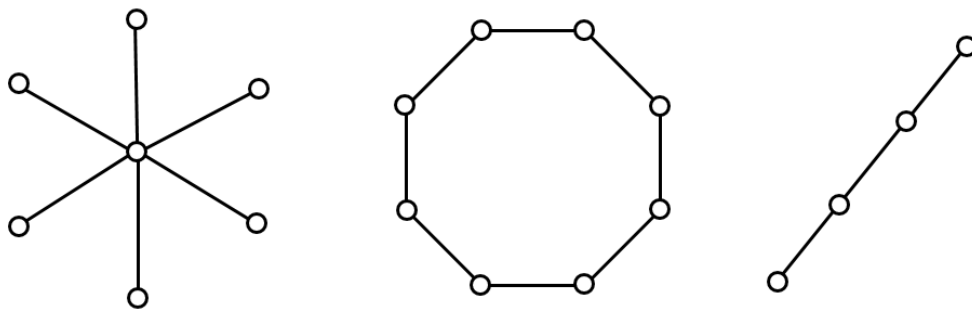


Figure 10.37. Primitive Graphs, featuring a Star (left), a Ring (middle) and a Chain (right). Each exhibiting examples of the calculated layouts provided by the Vertex Placement Scheme.

Use as a stopping mechanism is as simple as identifying if a newly coarsened graph is a primitive graph. After identification, the coarsening scheme is ended and the coarsest “primitive” graph is given a calculated layout using the Vertex Placement Scheme, see Implementation 10.22 .

Results for primitive graphs as a means of ending coarsening schemes are tested on the static test graphs, following the same methodology used in Coarsening Tolerance 0. Each test is repeated 10 times, with layout quality, running time and the primitive used (if any) recorded. The results are compared to the standard Multilevel Global Force Algorithm to determine which of the primitive graphs, if any, improved layout quality, with subjective analysis to identify and compare warping.

10.10.1 Numerical Results

There is little change between primitives due to the size of the primitive graphs identified (as a result of the weighted coarsener).

Table 10.10 shows that running time is largely unchanged on average, with changes being so small they can be regarded as fluctuations observed as a result of different matchings being identified. As such, running time of the coarsening process is included, showing that the coarsener takes longer time for smaller graphs, whereas larger graphs appear to have a reduction in running time. These changes come from use of identifying a primitive and providing them layout.

Graph	Running Time (ms)			
	Standard		Primitive	
	All	Coarsener	All	Coarsener
3025	846.2	43.4	885.0	58.8
data	1028.6	70.6	1005.8	73.8
add32	1968.8	64.6	2007.4	69.0
4elt	7507.8	207.8	8344.8	235.2
sierpinski10	67074.2	1145.0	68011.4	1443.4
finan512	69532.6	2015.6	71037.4	2015.8
dime20	218313.4	3220.2		
mesh100	94864.2	2009.0	84189.0	1762.2

Table 10.10. Comparison of running time for layout generation and coarsening processes using standard edge contraction (Standard) and primitive graph processing (Primitive)

Layout Quality in regard to edge crossings shows little change as well, shown in Table 10.11. The range of edge lengths show some increase on average, attributed to the structure of the *MGF* scheme (due to the repulsive approximation calculated in more than two directions, reducing the warping – as shown in Multimatching).

Graph	Standard		Primitive	
	Edge Crossings	Edge Length Range	Edge Crossings	Edge Length Range
3025	0.0	0.4696	0.0	0.4402
data	42255.2	0.7789	35868.2	0.8598
add32	12564.8	2.9163	12077.6	3.2129
4elt	20273.2	0.6604	24905.6	0.6832
sierpinski10	18717.6	0.2611	19664.2	0.2897
finan512	4719437.8	0.7856	4736235.6	0.7608
dime20	74200.2	0.4448		
mesh100	948812.6	0.3940	960406.4	0.3985

Table 10.11. Comparison of edge crossings and range of edge length for layouts generated using standard edge contraction (Standard) and primitive graph processing (Primitive)

From a numerical standing, there appears to be little overall benefit to primitive graph identification and layout.

10.10.2 Subjective Analysis

In application, the graph *finan512* is often coarsened to a ring or a star, as such, layouts can be provided as shown in **Figure 10.38**. Note due to the small size of the primitive graph, the benefit of using the primitive is only achieved in the structure of the *MGF*. The layout generated features less warping that using standard edge crossings, however, the weighting of vertices still cause some deformation (note there is some compression).

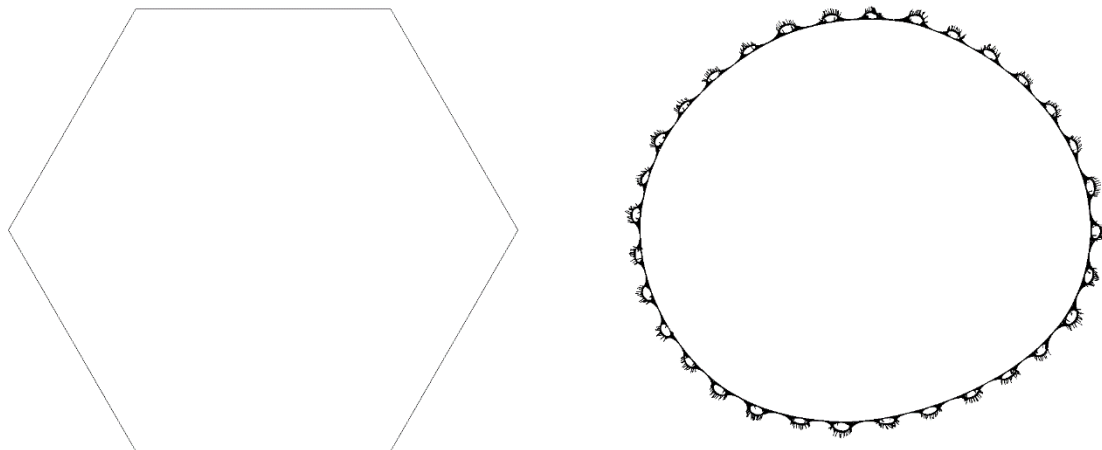


Figure 10.38. *Finan512* showing the primitive ring graph identified during the coarsening process having been given a layout generated by the VPS (left), and the resulting layout for the graph (right)

Layouts for *dime20* (**Figure 10.39**) and *add32* (**Figure 10.40**) show little change between usage of primitive (right) or standard edge contraction (left). Subtle changes may be noticed, for example, the increase in

edge length range can be seen in the primitive layout for dime20, whereby the compression on the outer skirts of the layout is minimised in some parts.

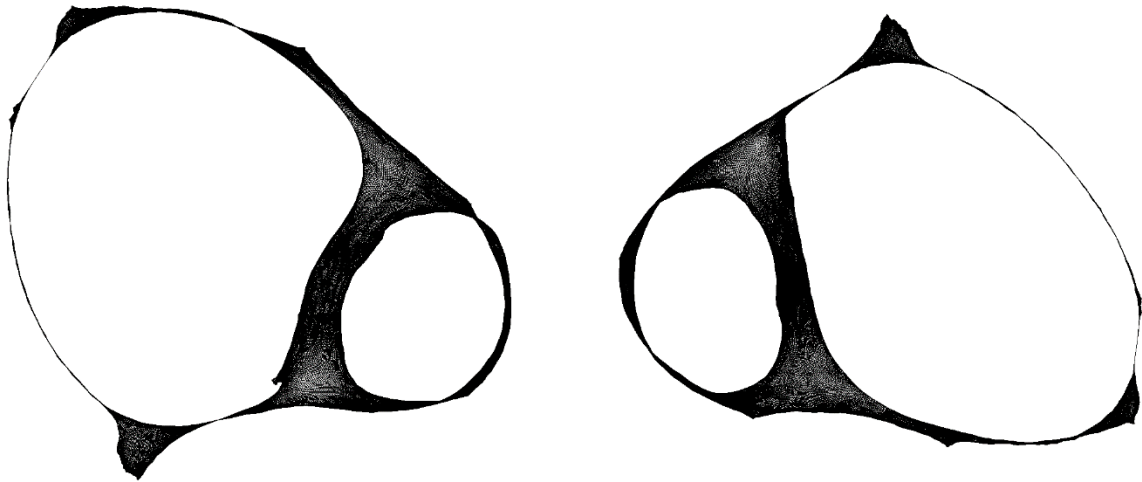


Figure 10.39. Comparison of layouts for graph dime20 drawn using the primitive graph coarsener (left) and standard edge contraction (right).

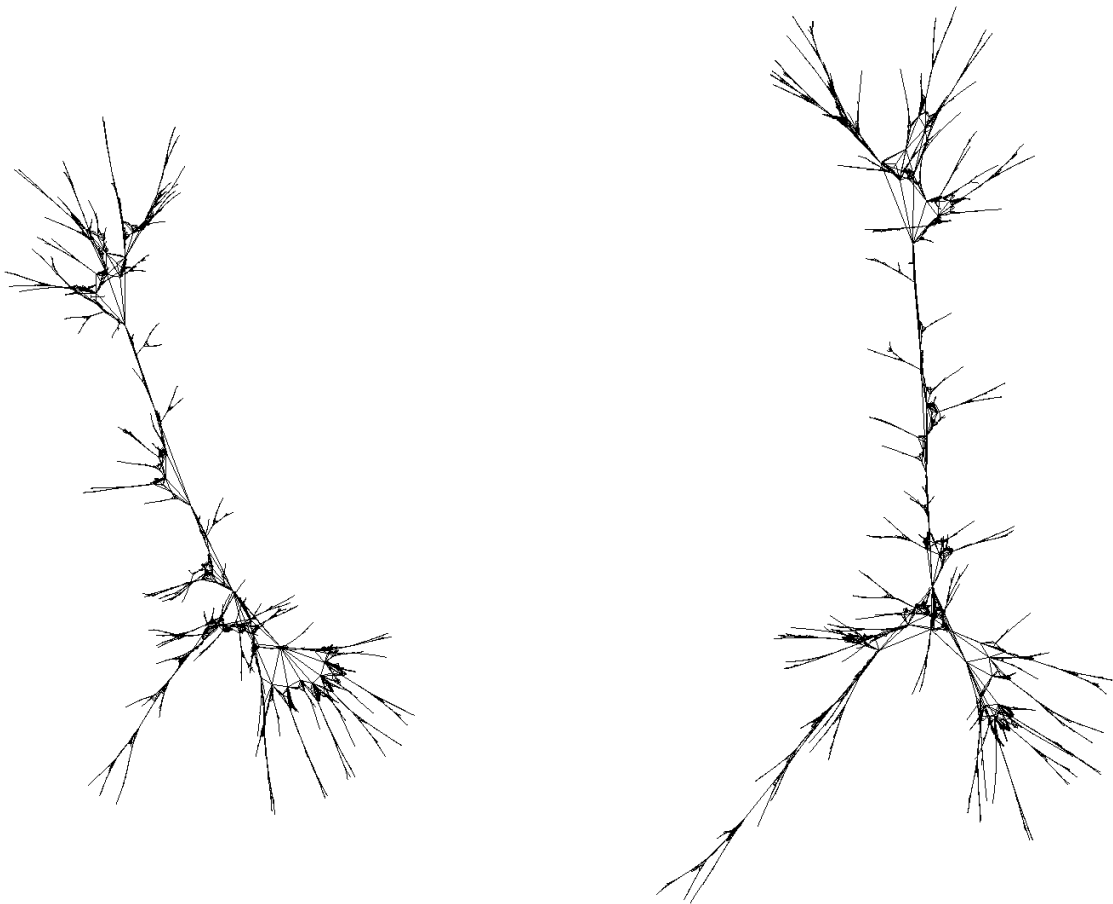


Figure 10.40. Comparison of layouts for graph add32 drawn using the primitive graph coarsening process (left) and standard edge contraction (right)

10.11 Configuring Dynamic Spring Embedder

For the collection of results, the abbreviations “EC” and “MMT” are used to refer to the number of edge crossings for a frame, and the average movement between a frame and a previous frame.

Figure 10.41 and Figure 10.42 provide an illustration (chart) showing changes to the measured layout quality over time for regular-small. The measurements show that a value of 0.5 for tr generates a layout with zero edge crossings (on average) within 180 frames. In contrast, no other values are shown to provide such layouts, with 0.3 being closest on average. In addition to providing high quality layouts for the graph, the change in movement appears to decrease as well, suggesting movement slows as a layout is identified. However, a value of 0.3 provides reduced movement with 0.1 providing the least movement, requiring subjective analysis to confirm which of the two measurements (edge crossings or movement) provides smoothest visualisation and understanding of the development of a layout.

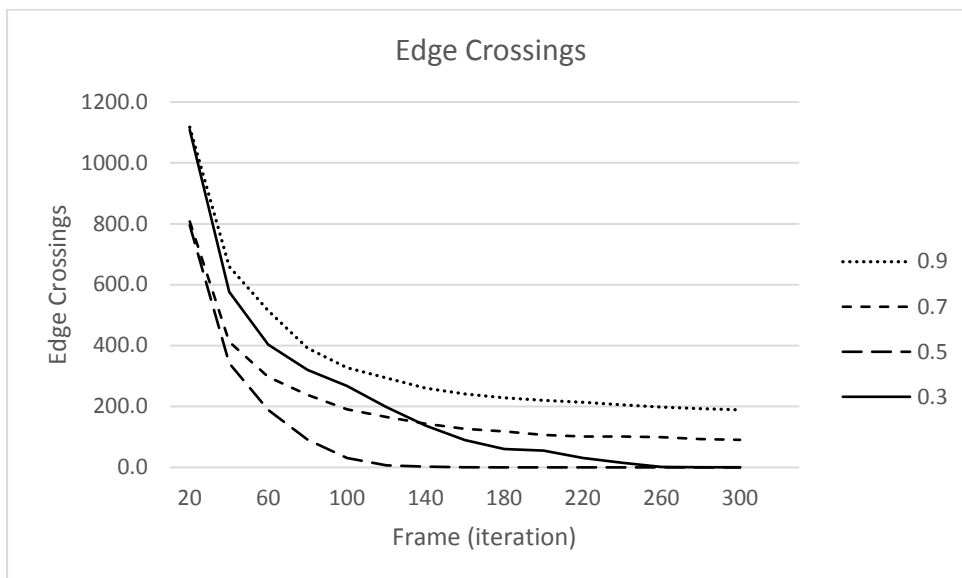


Figure 10.41 Change in edge crossings 300 iterations of force directed placement of graph regular-small, showing the difference of the value tr (repulsive force displacement)

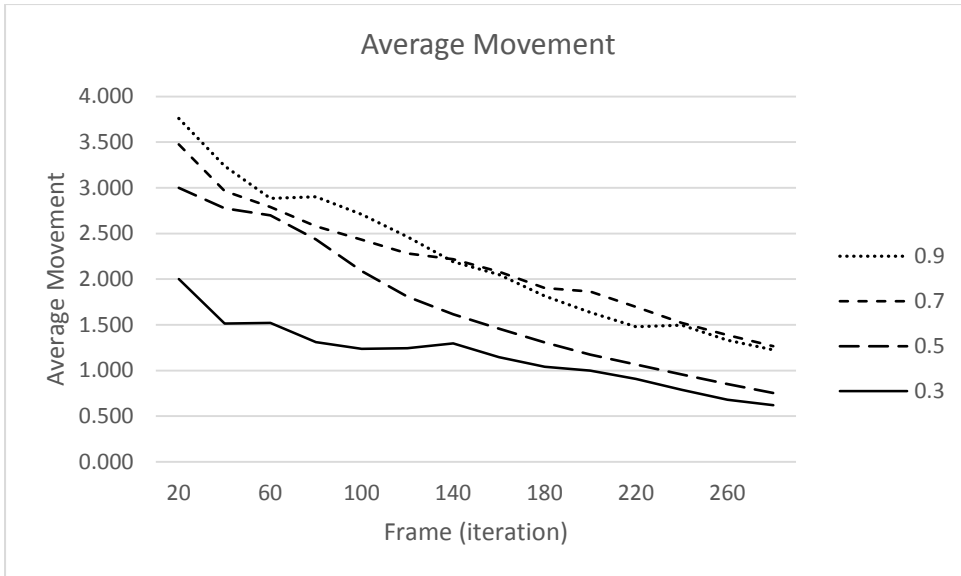


Figure 10.42 Change in average vertex movement 300 iterations of force directed placement of graph regular-small, showing the difference of the value tr (repulsive force displacement)

Investigation of the value of tr on the metrics for graph stability of the graph sparse-small is provided in Figure 10.43 and Figure 10.44. The results suggest that a large value of tr is better for representing the sparse graph, with a value of 0.7 providing layout with least edge crossings, however, a value of 0.5 follows closely behind in regard to rate of convergence. For movement, a value of 0.5 shows a gradual reduction as layout is found, in contrast to values of 0.1 which shows increasing movement as layout is found and 0.9 which shows a very quick drop to a low amount of movement. As above, subjective analysis is required to identify which of the values is easier to read, with a value of 0.7 determined as the ideal.

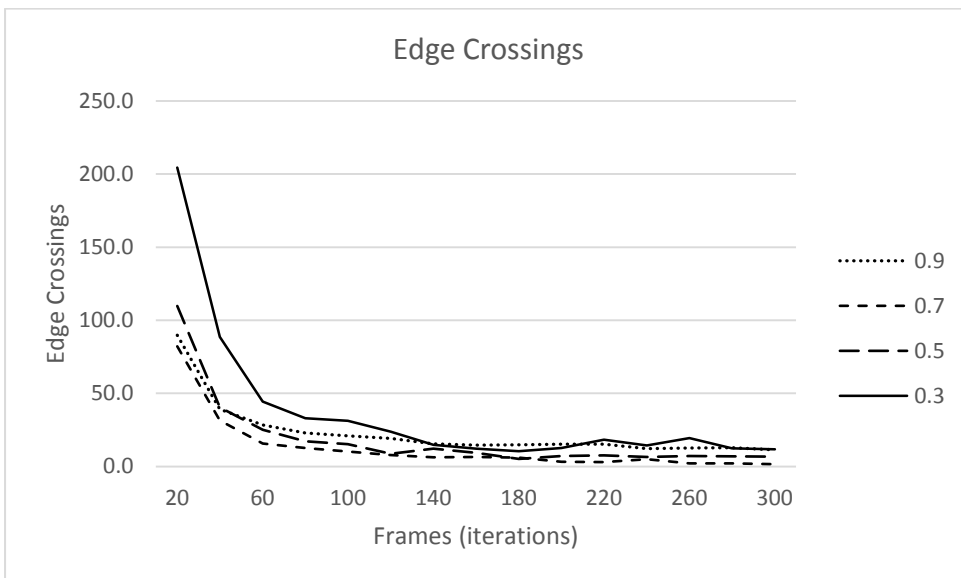


Figure 10.43 Change in edge crossings 300 iterations of force directed placement of graph sparse-small, showing the difference of the value tr (repulsive force displacement)

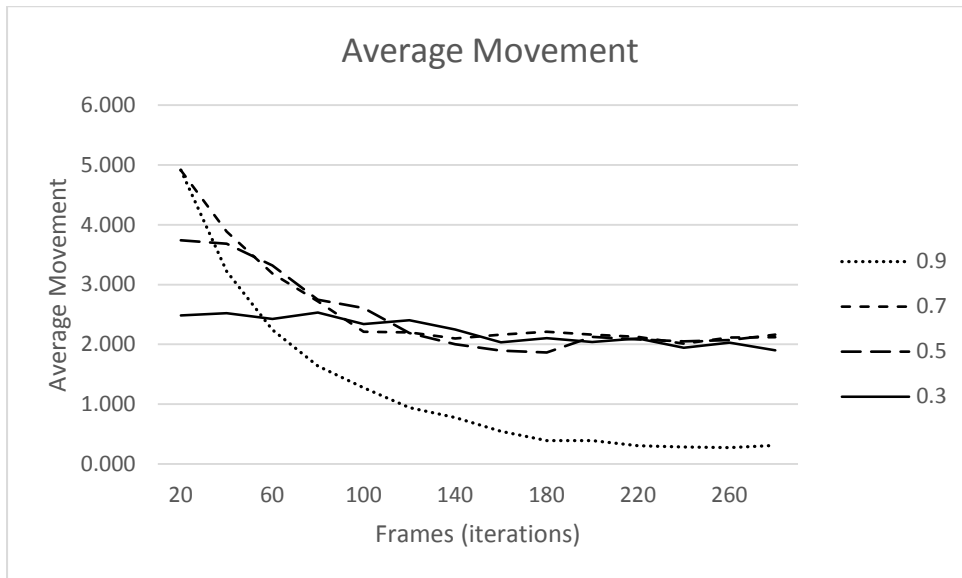


Figure 10.44 Change in average vertex movement 300 iterations of force directed placement of graph sparse-small, showing the difference of the value tr (repulsive force displacement)

Figure 10.45 and Figure 10.46 shows the measurements of the metrics for graph stability for the graph *dense-small*. The results suggest that a value of 0.9 for tr , provides layouts with the greatest rate of convergence and associated vertex movement. Unlike the examples above, the structure of dense graph provides layouts with high density of edge crossings and therefore the focus moves from minimal edge crossings to least movement (see Subjective Analysis).

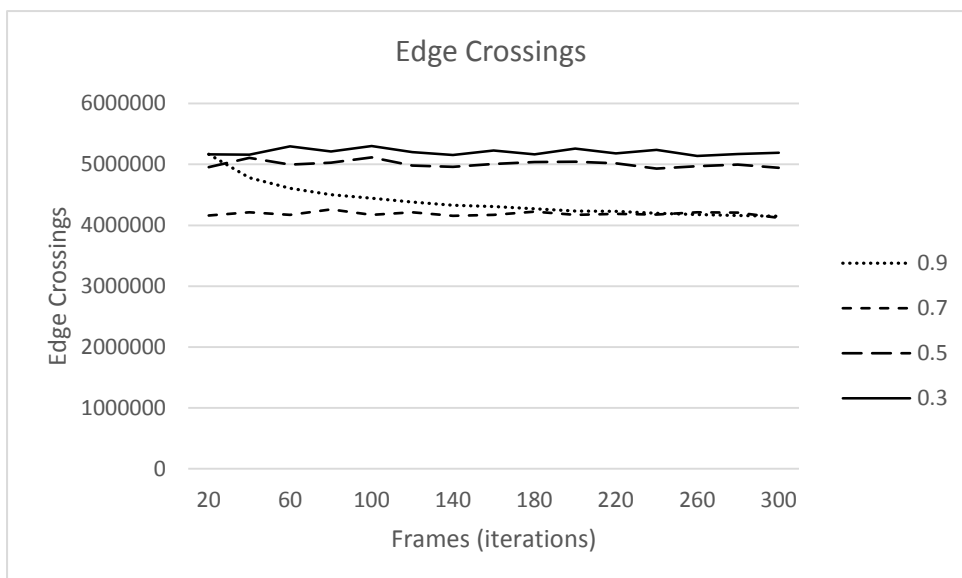


Figure 10.45 Change in edge crossings 300 iterations of force directed placement of graph dense-small, showing the difference of the value tr (repulsive force displacement)

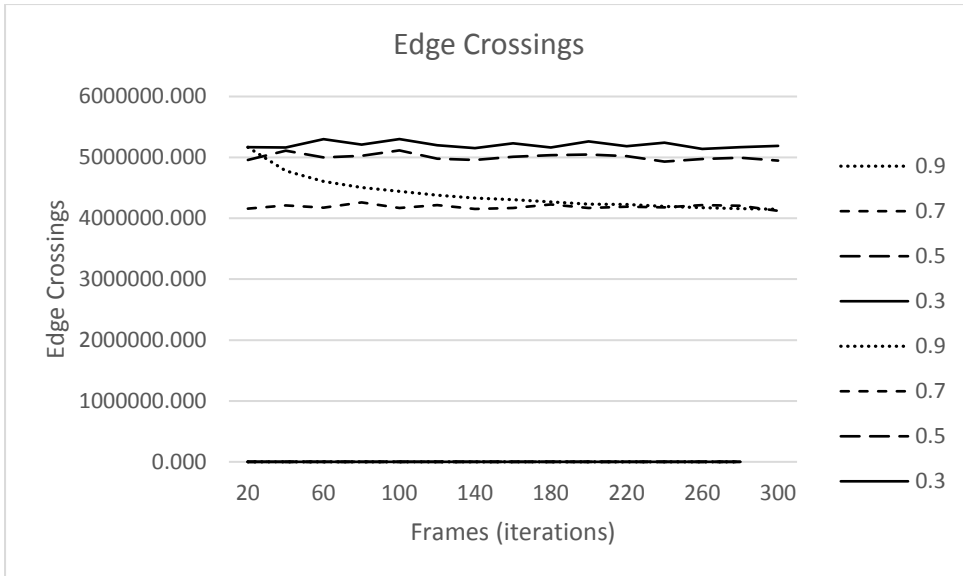
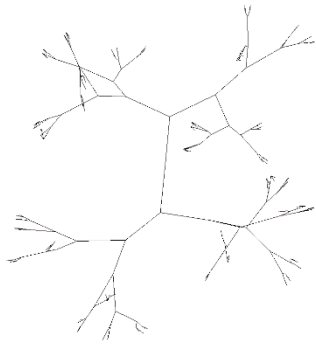
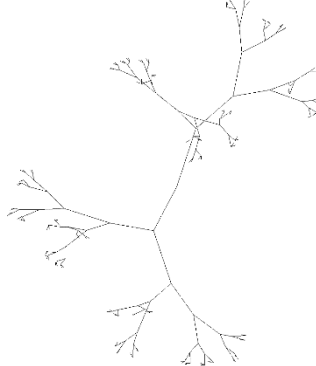
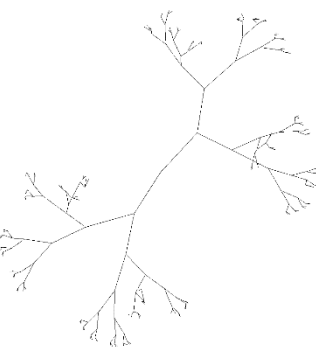
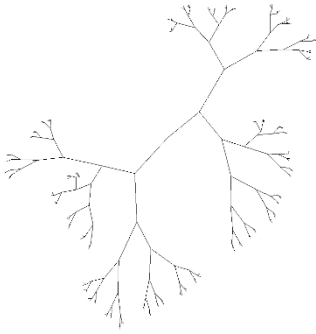
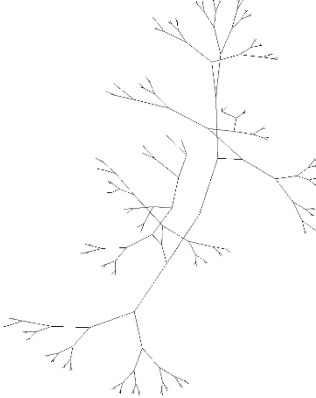
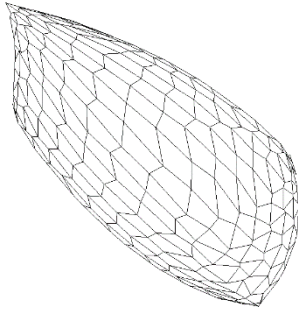
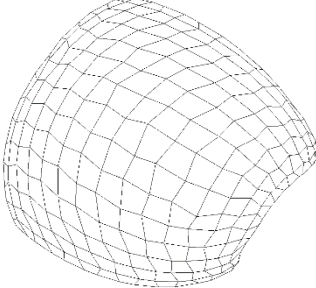
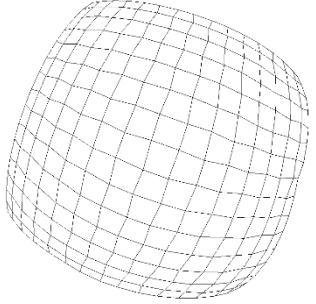
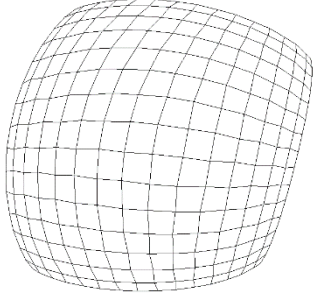
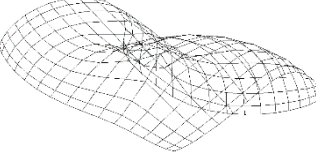


Figure 10.46 Change in average vertex movement 300 iterations of force directed placement of graph dense-small, showing the difference of the value tr (repulsive force displacement)

Layouts for the above charts are included below showing the layouts provided and the quality difference between them. Layouts for sparse-small show that the force largely changes the clustering of leaf like vertices, with many of the graphs showing readable layout. Layouts for regular-small show compression and warping as forces are unable to escape each other, with a value of 0.5 providing highest quality layouts. Dense small shows to have very erratic movement for lower values of tr , with a n equilibrium preferred for generating symmetrical/equal layouts (lower values for tr show high movement as vertices swap between far and close from the centre of mass).

0.1	0.3	0.5	0.7	0.9
				
				

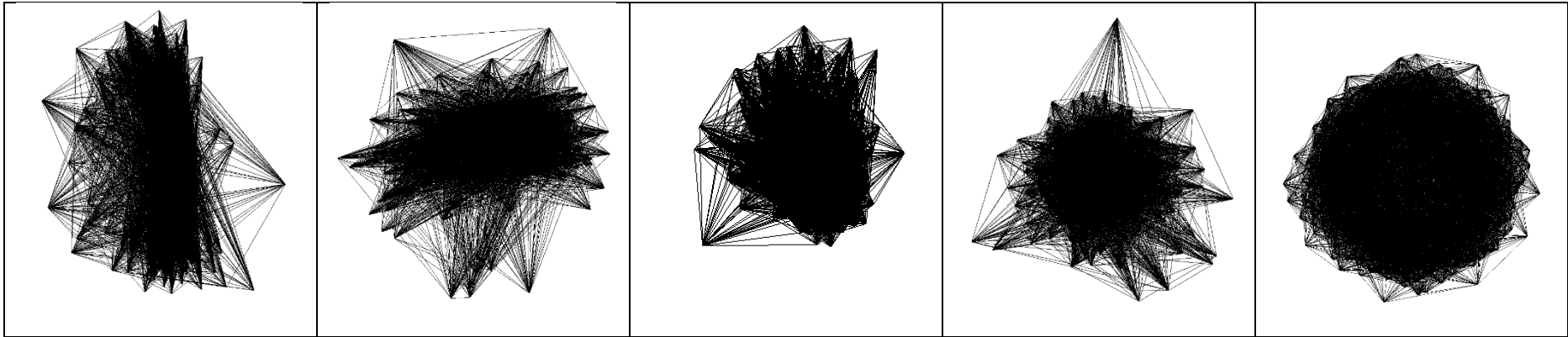
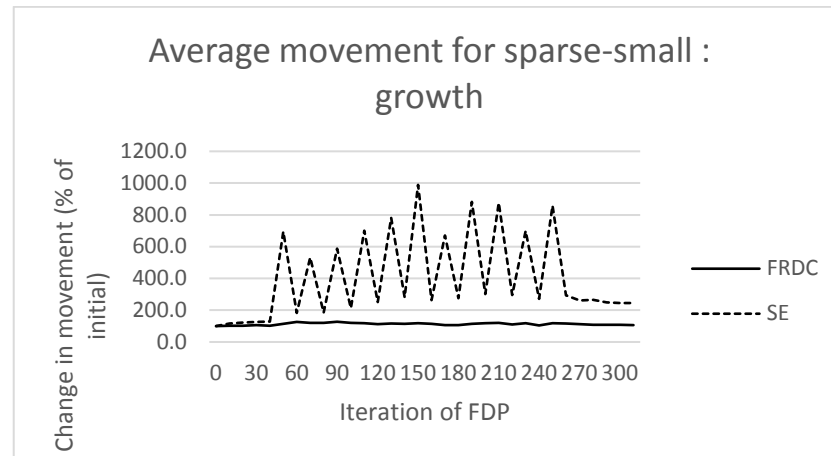
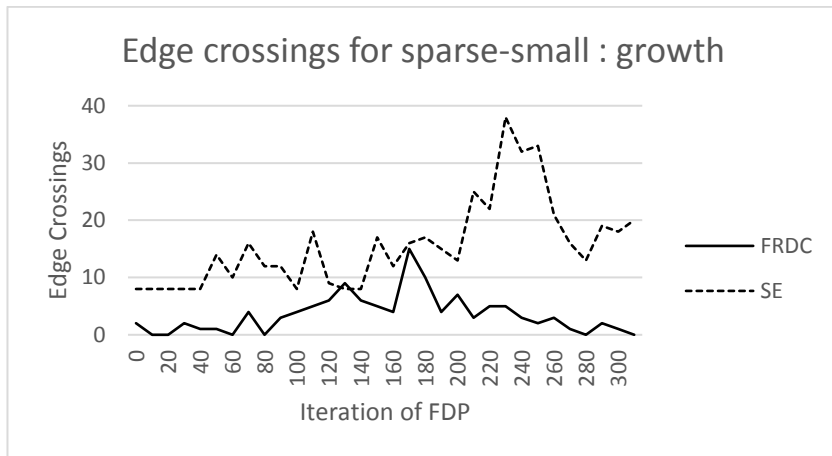
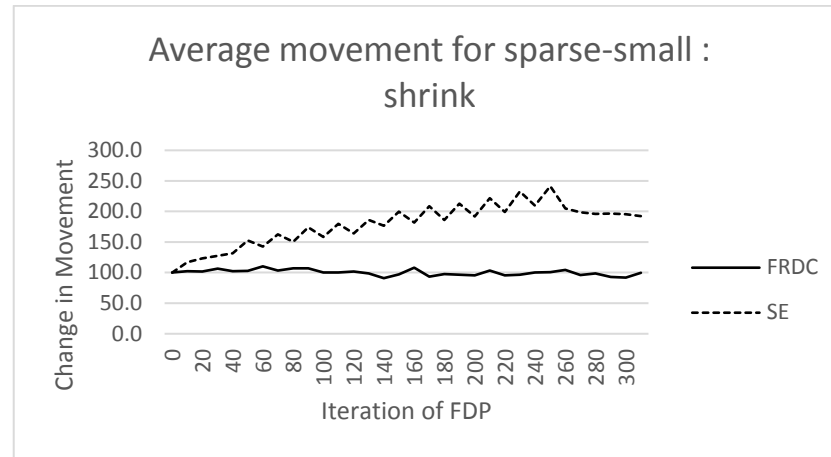
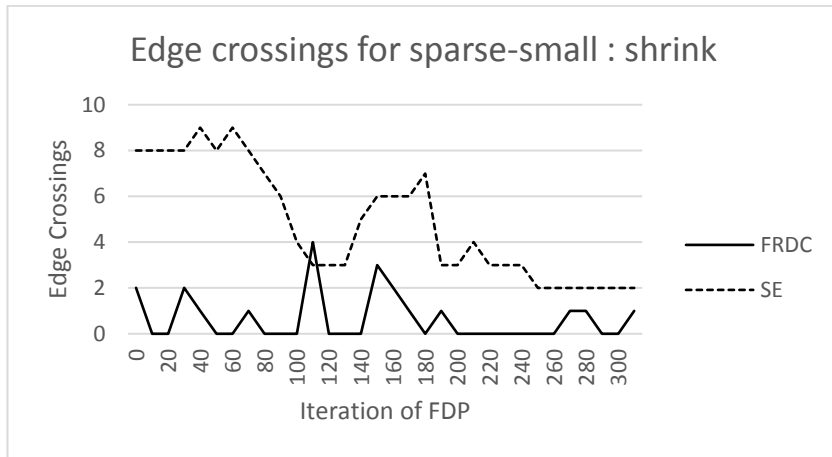
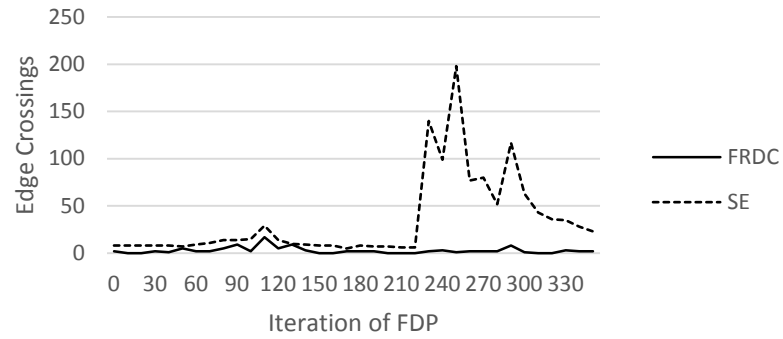


Table 10.12 Layouts given to the graphs regular-small, sparse-small and dense-small using tr values of '0.1, 0.3, 0.5, 0.7 and 0.9', showing the impact of changing the equilibrium of forces used in force directed placement. Layouts for sparse-small show that the force largely changes the clustering of leaf like vertices, with many of the graphs showing readable layout. Layouts for regular-small show compression and warping as forces are unable to escape each other, with a value of 0.5 providing highest quality layouts. Dense small shows to have very erratic movement for lower values of tr , with a n equilibrium preferred for generating symmetrical/equal layouts (lower values for tr show high movement as vertices swap between far and close from the centre of mass)

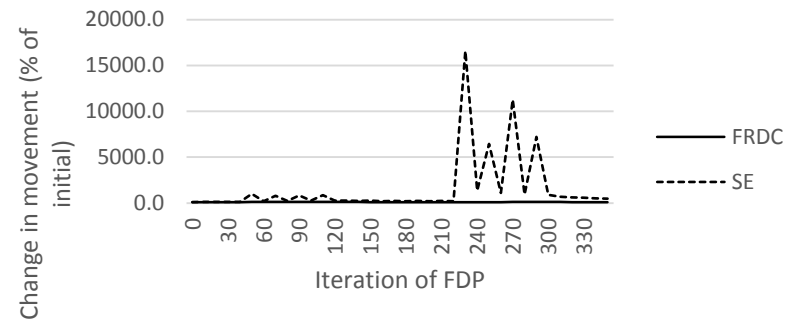
10.12 Comparison of Layout Adjustment for Dynamic Spring Embedder and the Eades Spring Embedder Implementation



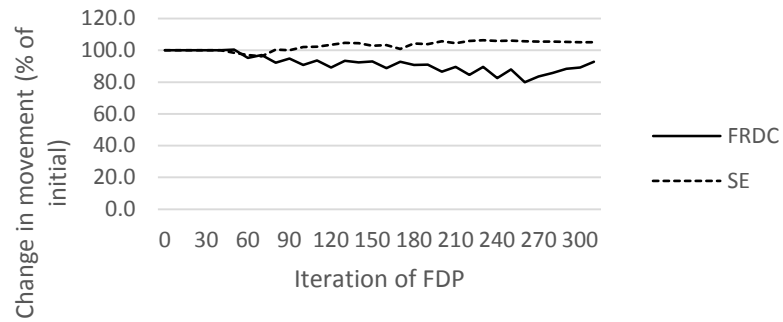
Edge crossings for sparse-small : maintain



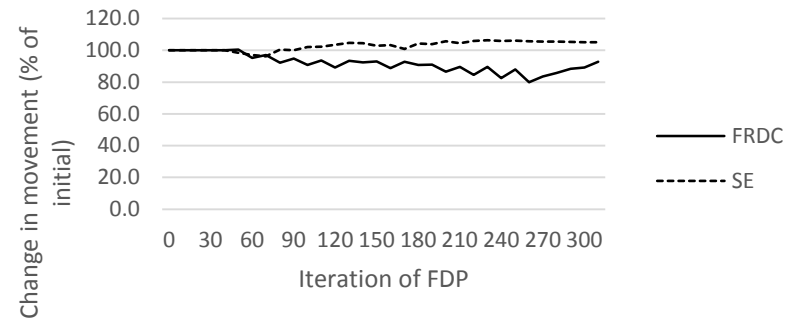
Average movement for sparse-small : maintain



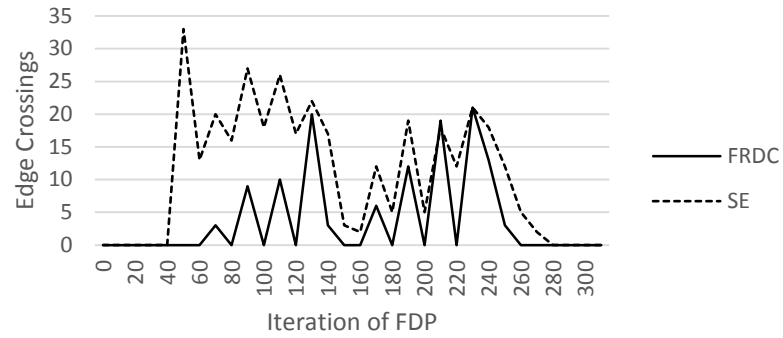
Average movement for regular-small : shrink



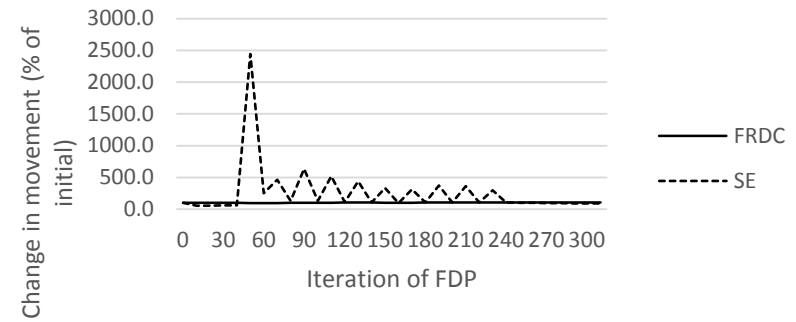
Average movement for regular-small : shrink



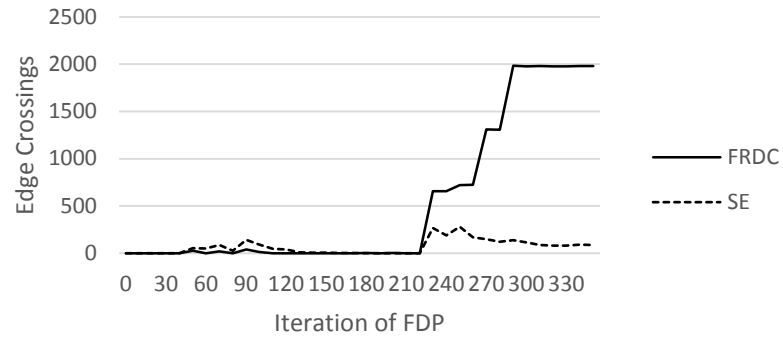
Edge crossings for regular-small : growth



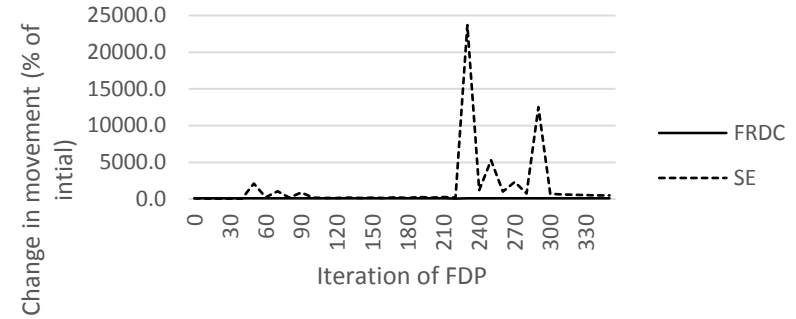
Average movement for regular-small : growth



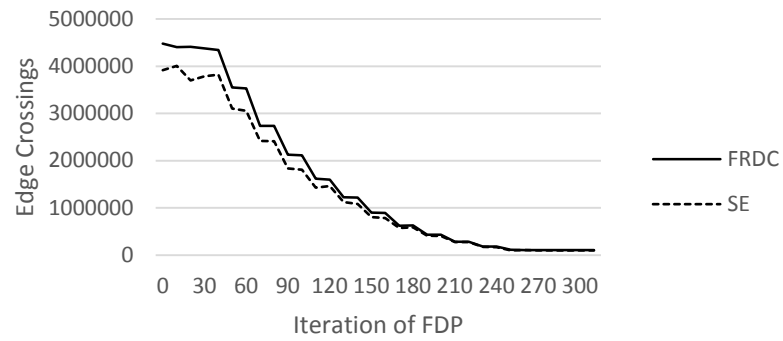
Edge crossings for regular-small : maintain



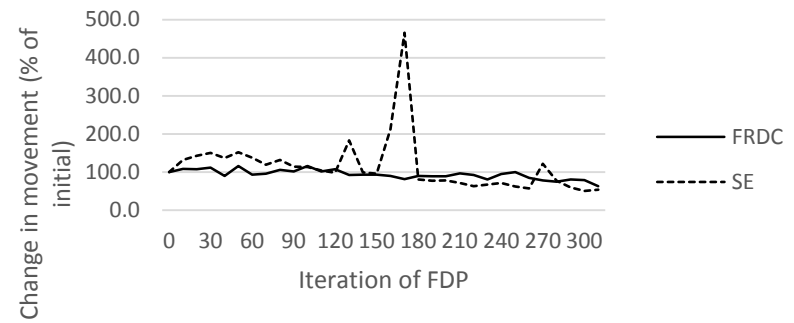
Average movement for regular-small : maintain



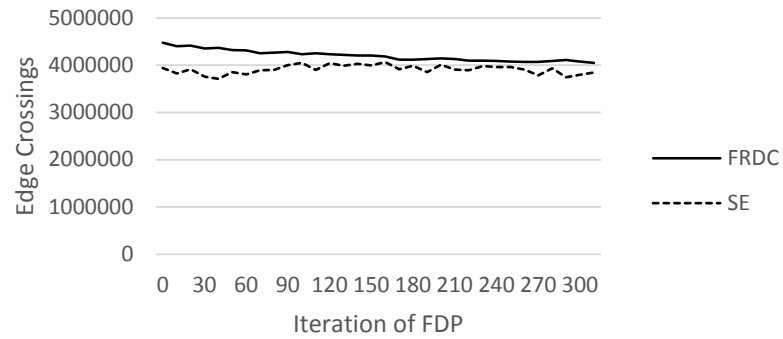
Edge crossings for dense-small : shrink



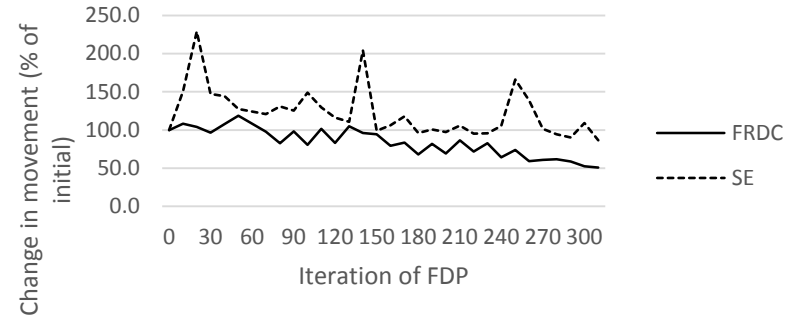
Average movement for dense-small : shrink



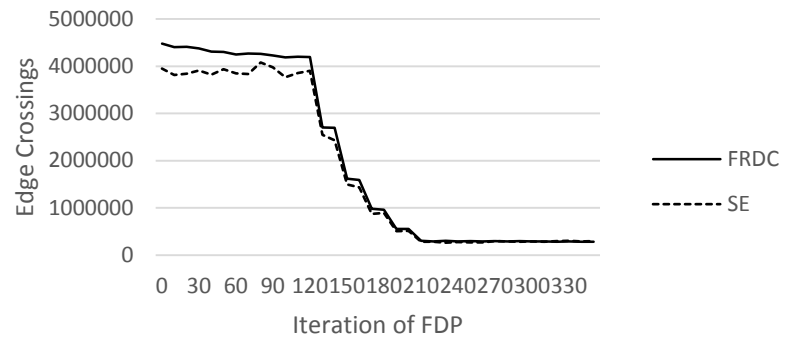
Edge crossings for dense-small : growth



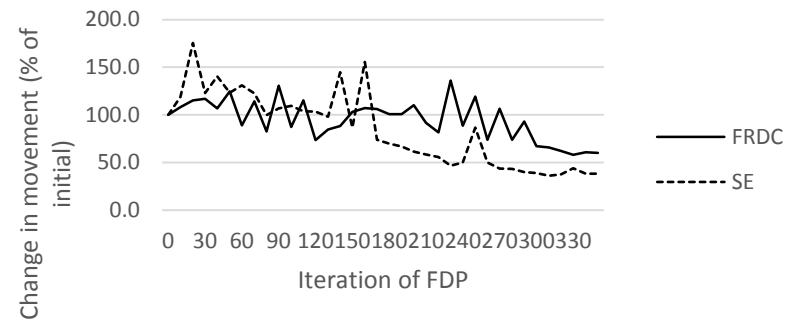
Average movement for dense-small : growth



Edge crossings for dense-small : maintain



Average movement for dense-small : maintain

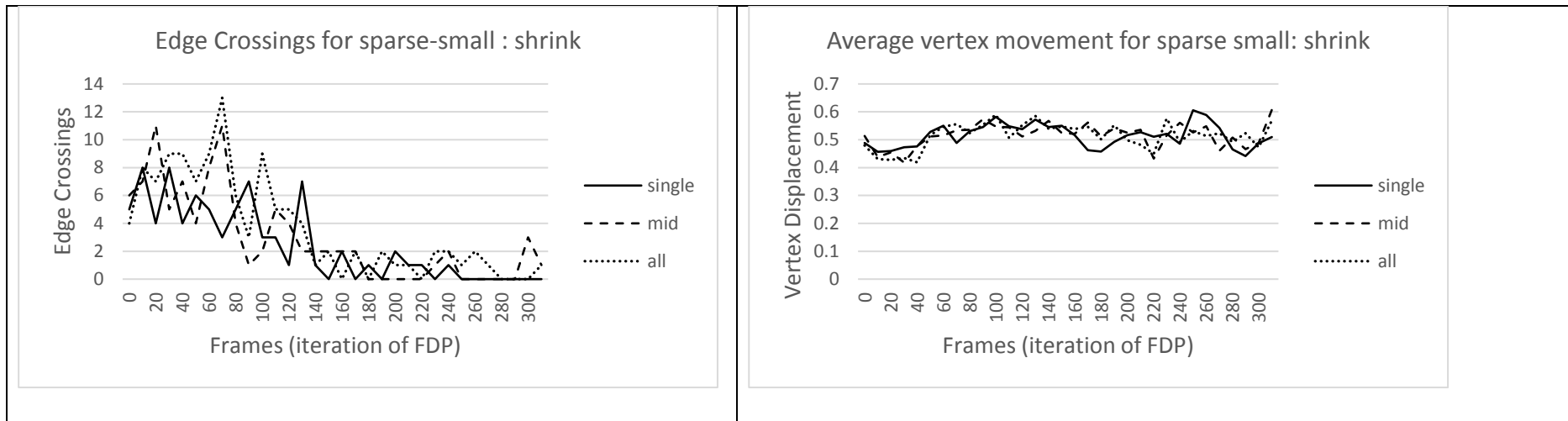


10.13 Dynamic Matching: Multilevel Updates

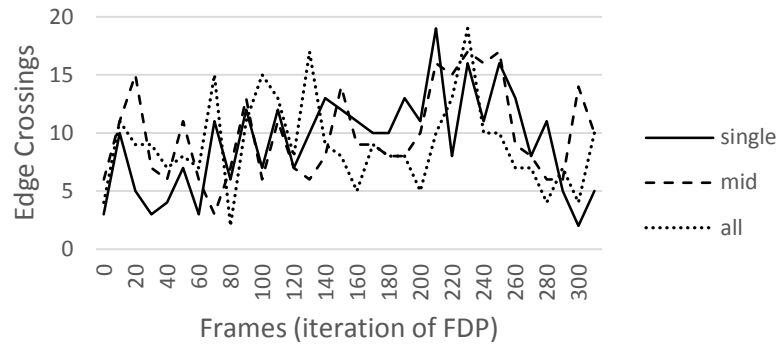
10.13.1 Small Graphs

10.13.1.1 Single, Middle and Full Update Comparison

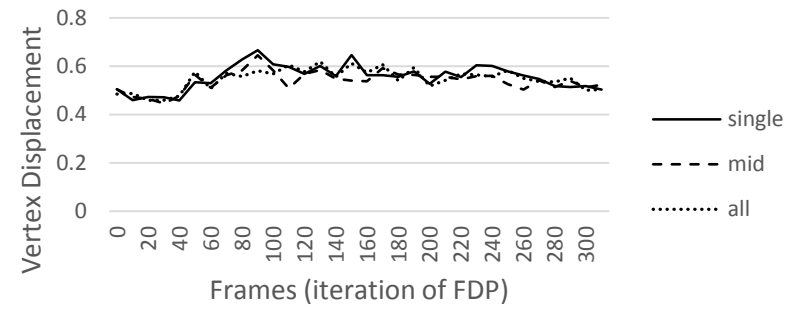
Sparse-small : shrink, grow and maintain operations for single, middle and full update methods



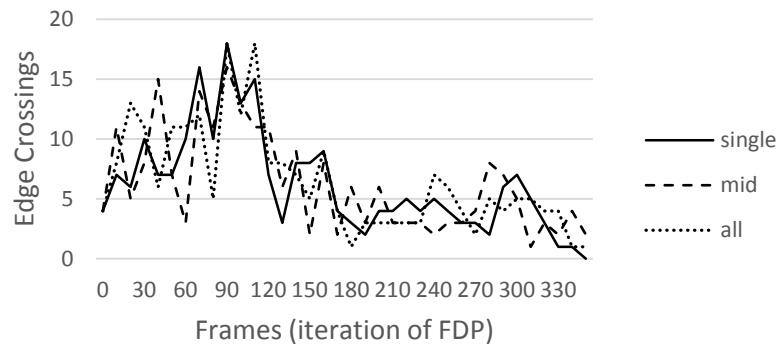
Edge Crossings for sparse-small : growth



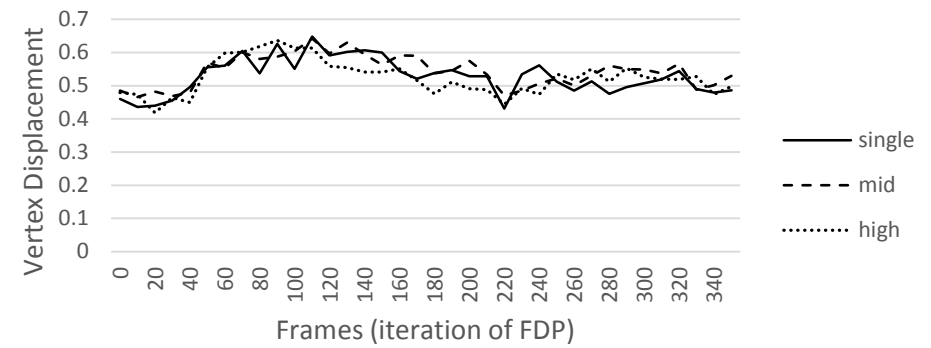
Average vertex movement for sparse small: growth



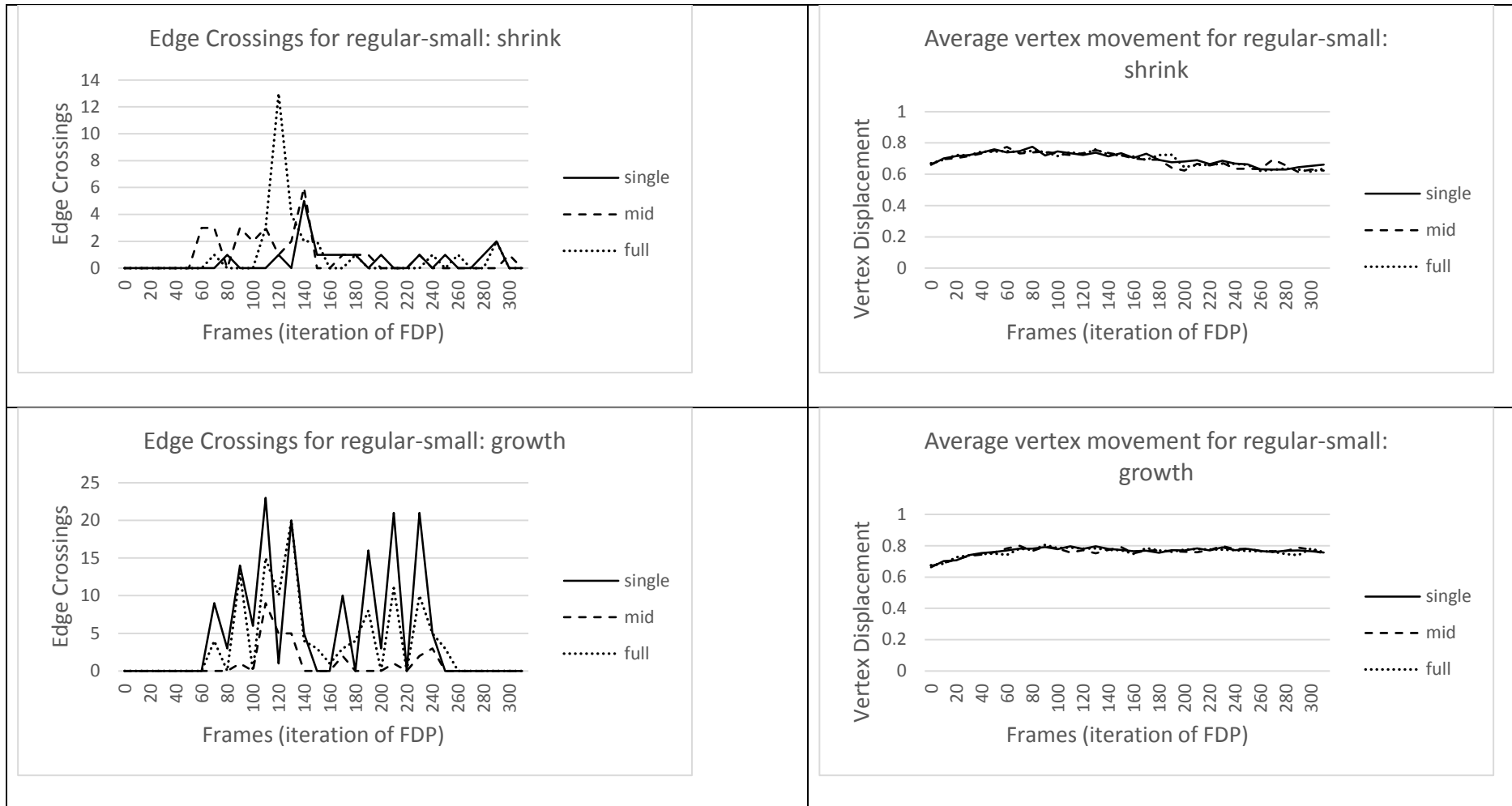
Edge Crossings for sparse-small : maintain

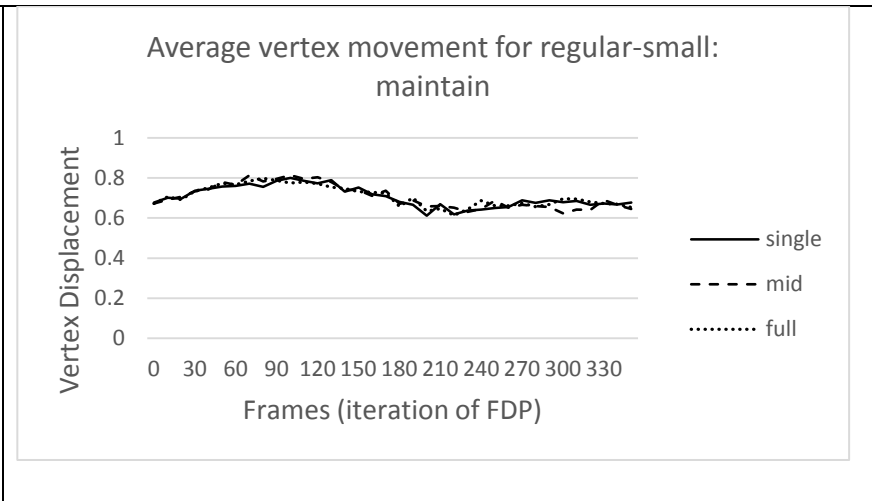
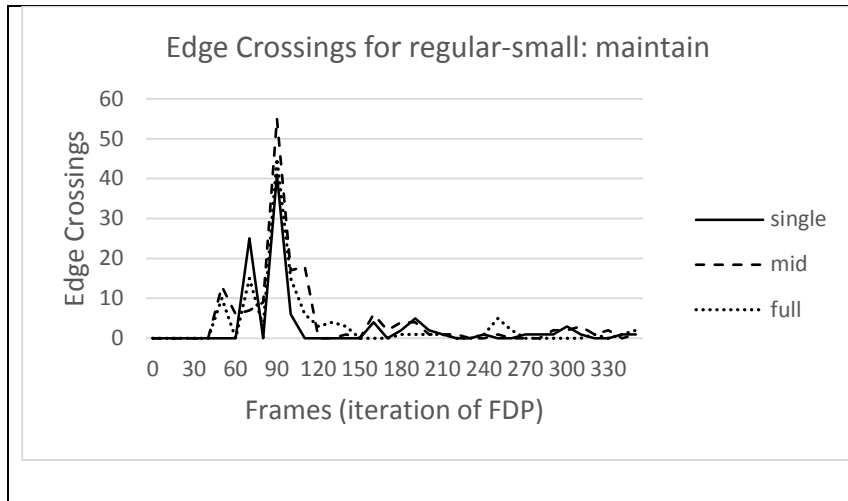


Average vertex movement for sparse small: maintain

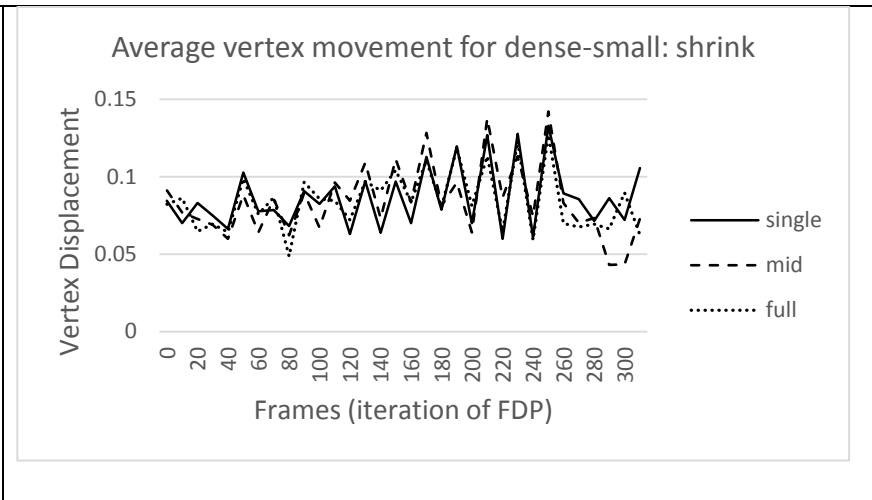
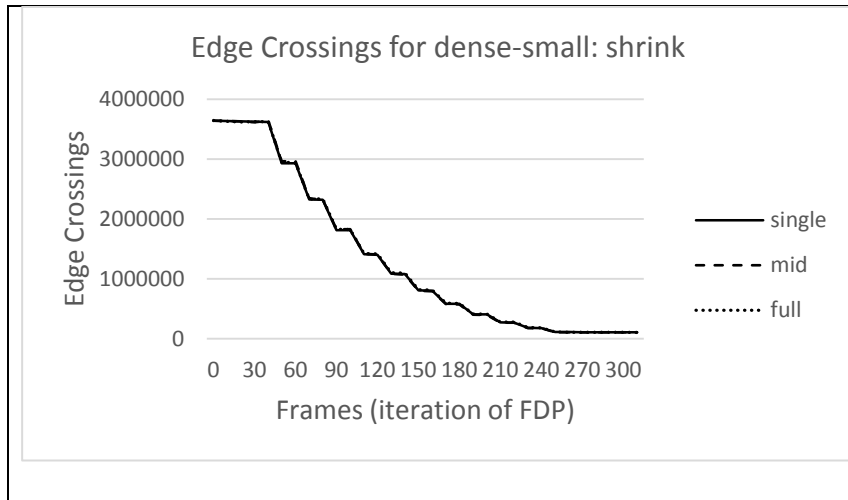


regular-small : shrink, grow and maintain operations for single, middle and full update methods

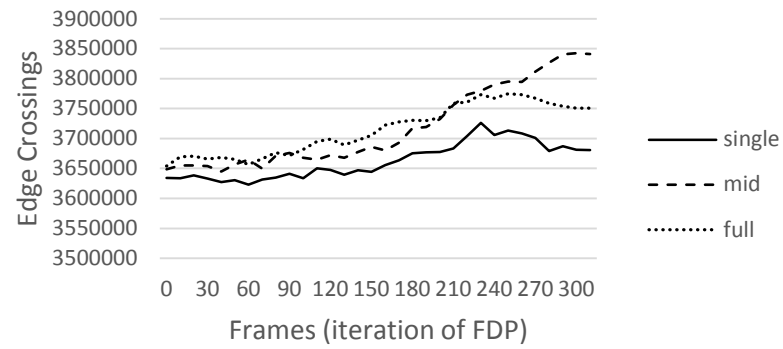




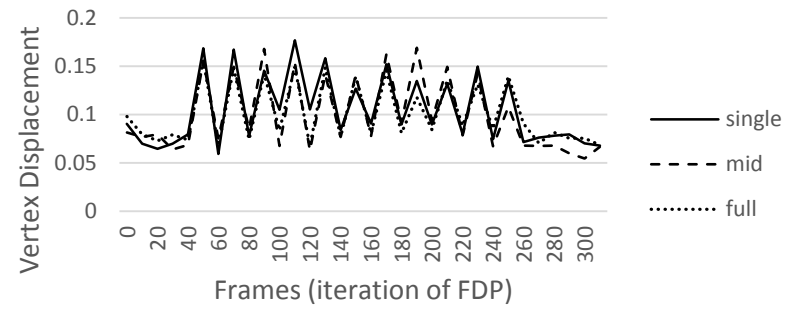
dense-small : shrink, grow and maintain operations for single, middle and full update methods



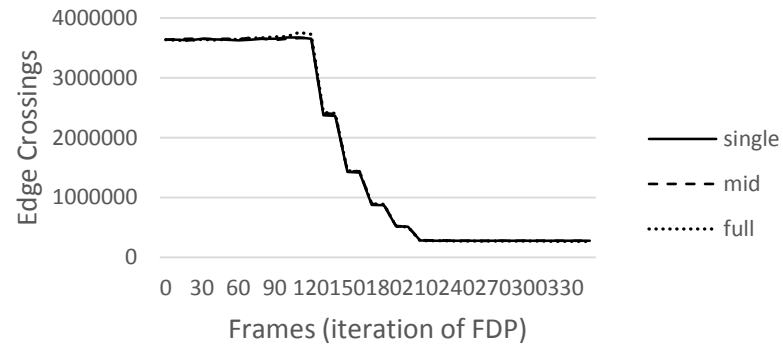
Edge Crossings for dense-small: growth



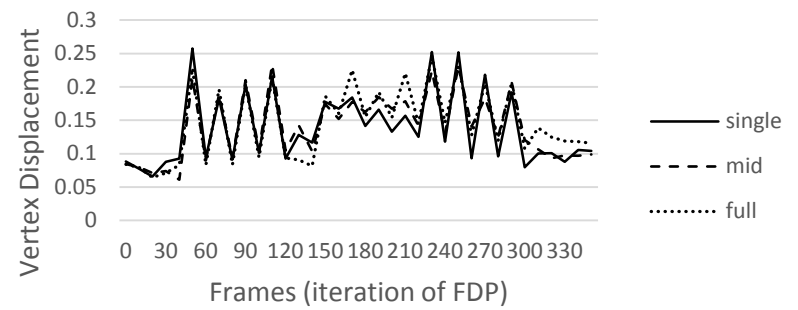
Average vertex movement for dense-small: growth



Edge Crossings for dense-small: maintain

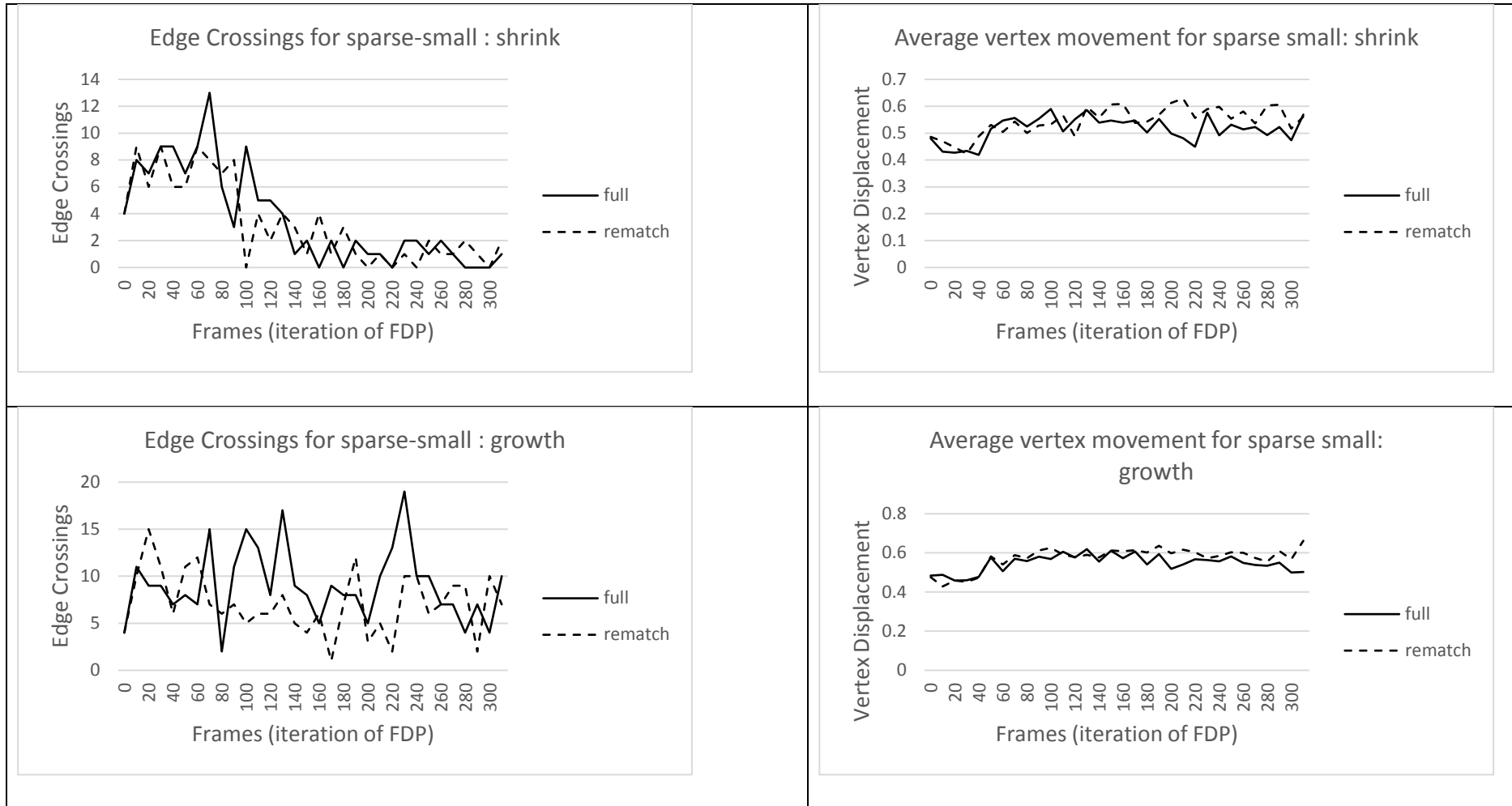


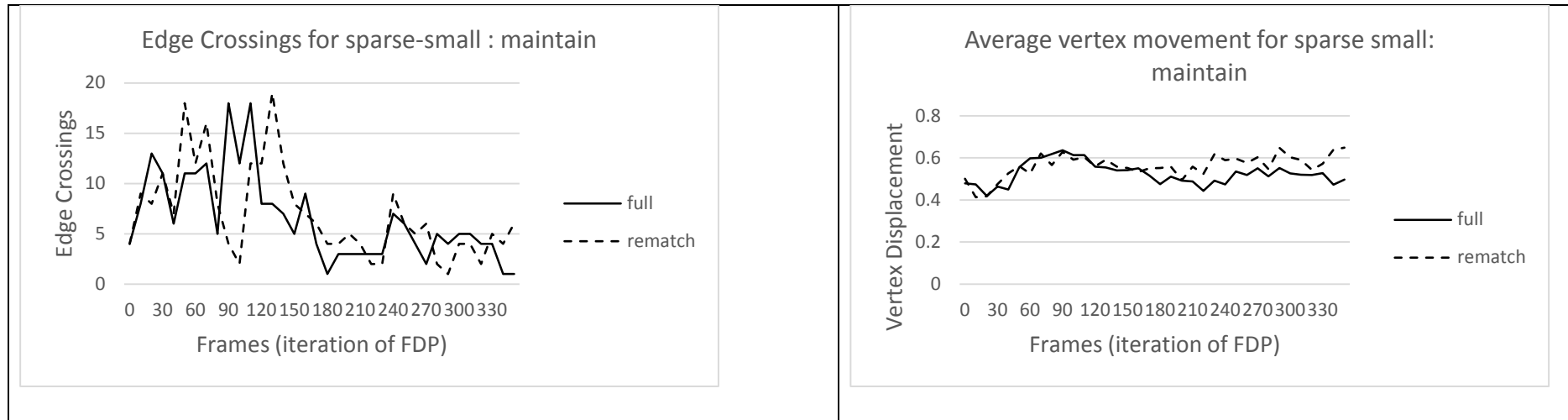
Average vertex movement for dense-small: maintain



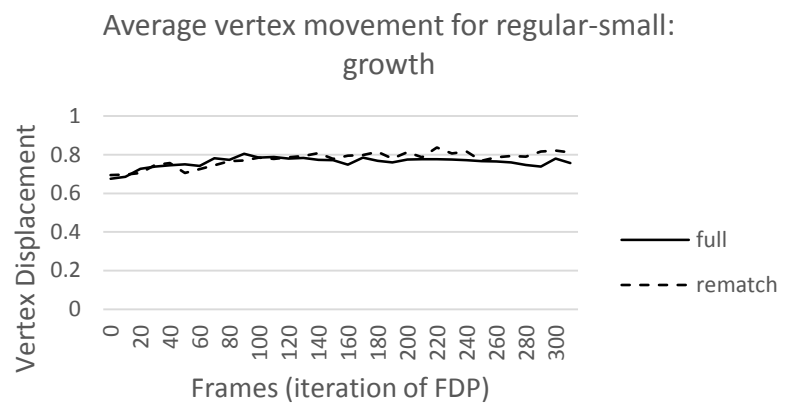
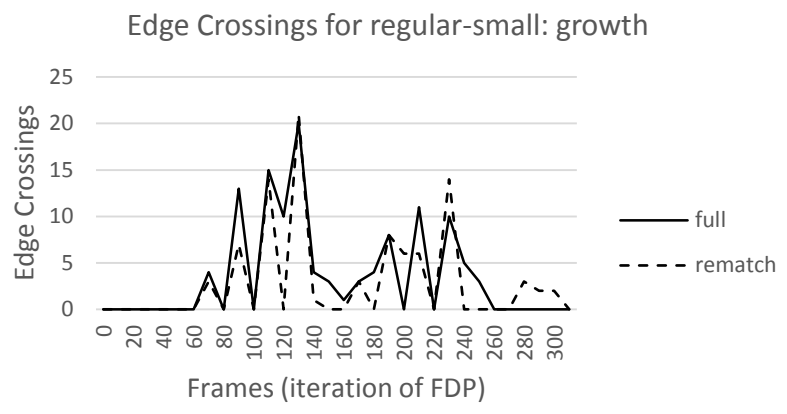
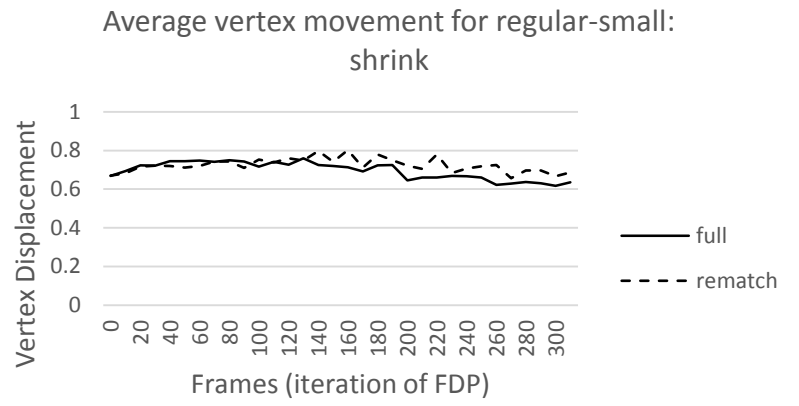
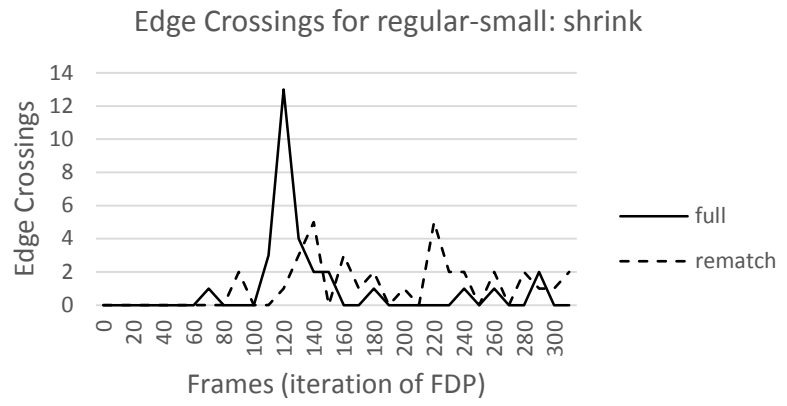
10.13.1.2 Full and Rematch Update Comparison

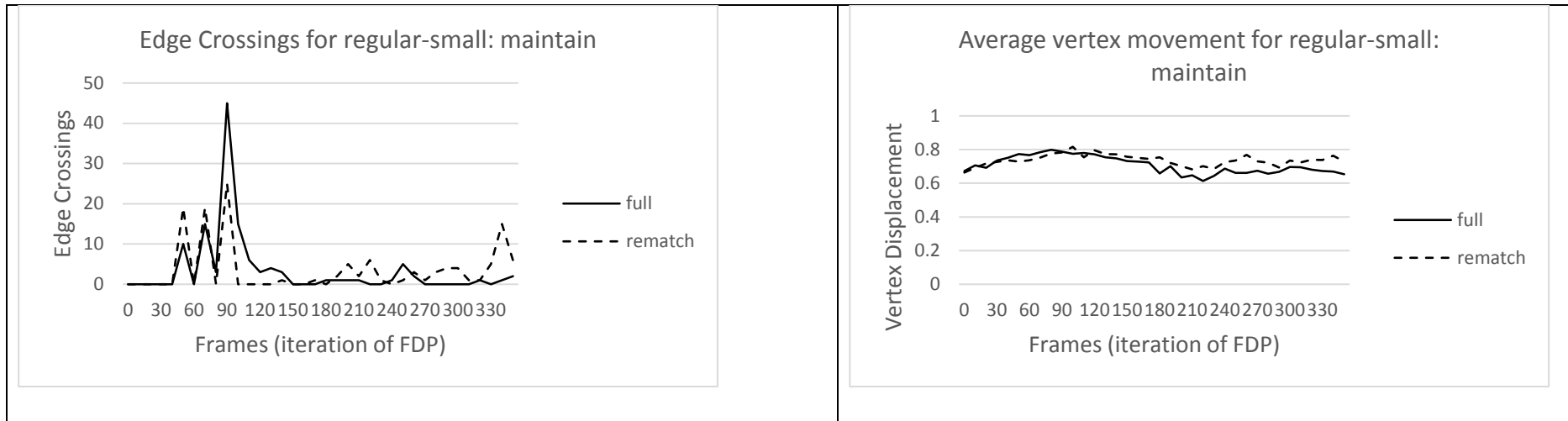
sparse-small : shrink, grow and maintain operations for full and rematch update methods



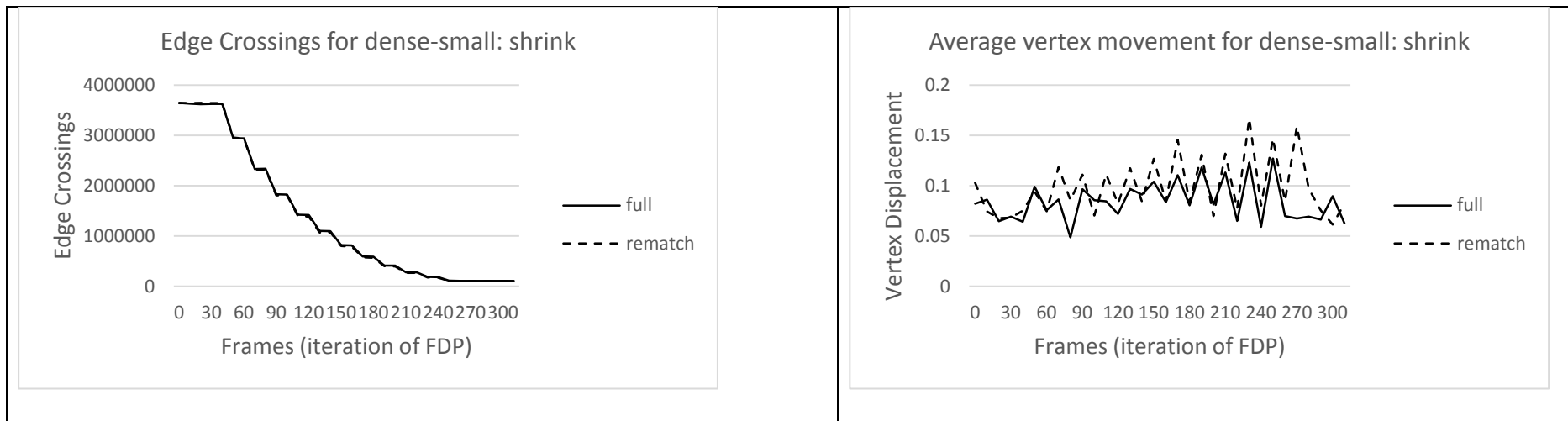


regular-small : shrink, grow and maintain operations for full and rematch update methods

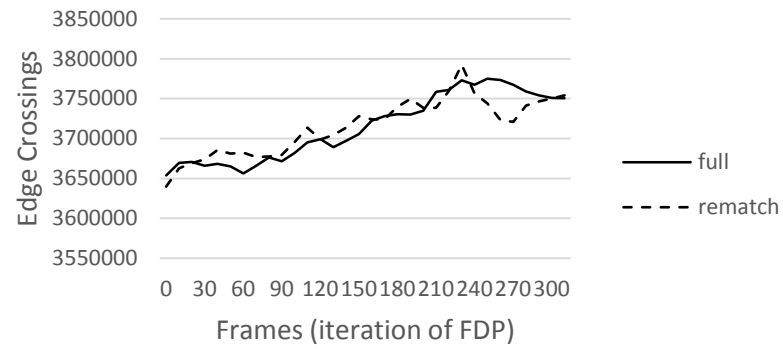




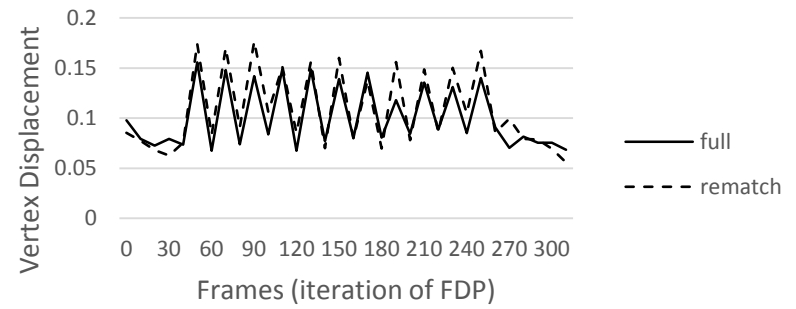
dense-small : shrink, grow and maintain operations for full and rematch update methods



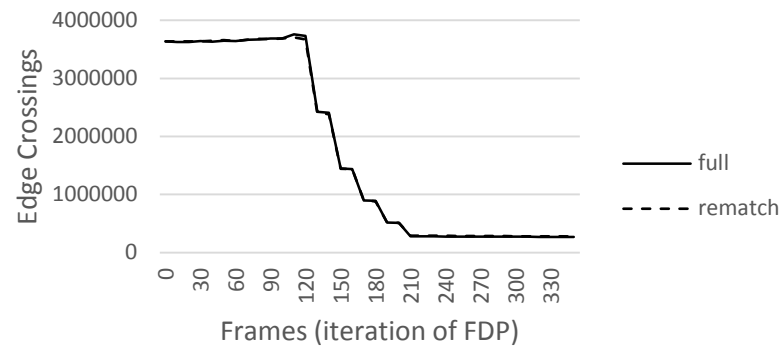
Edge Crossings for dense-small: growth



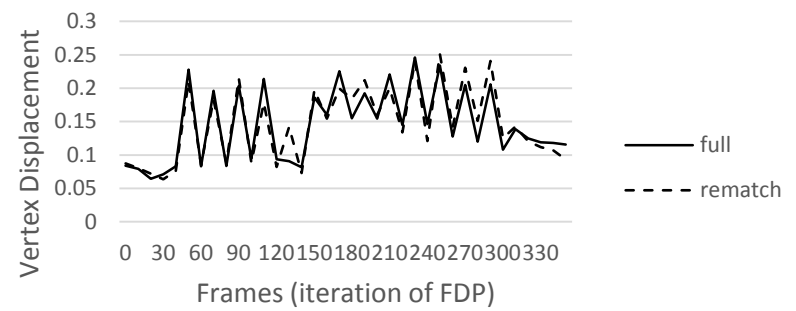
Average vertex movement for dense-small: growth



Edge Crossings for dense-small: maintain



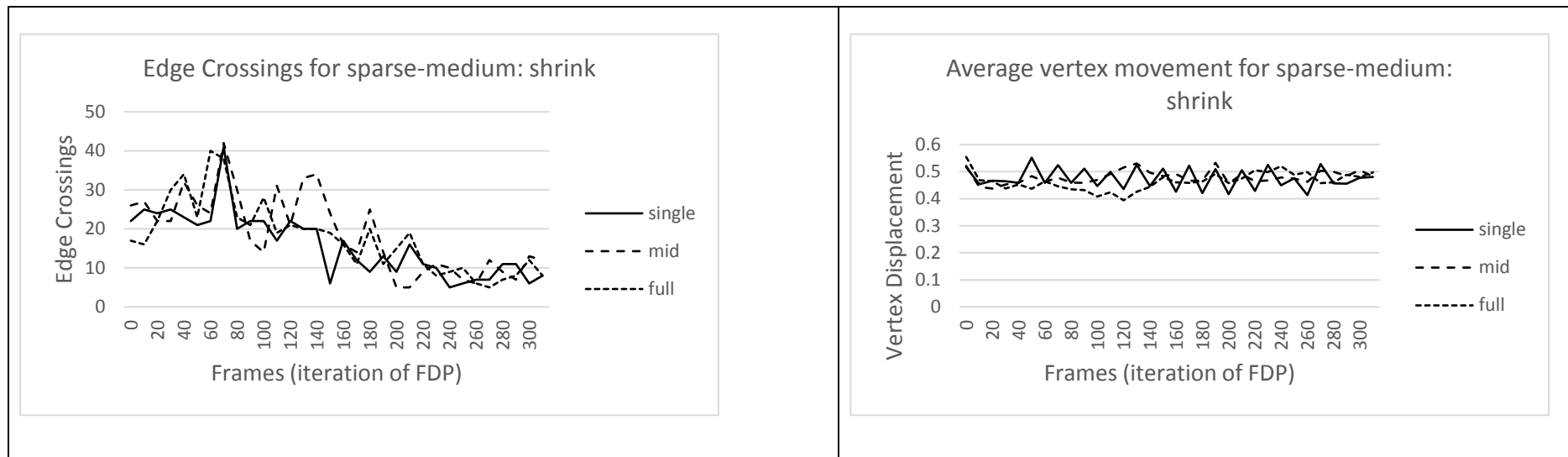
Average vertex movement for dense-small: maintain

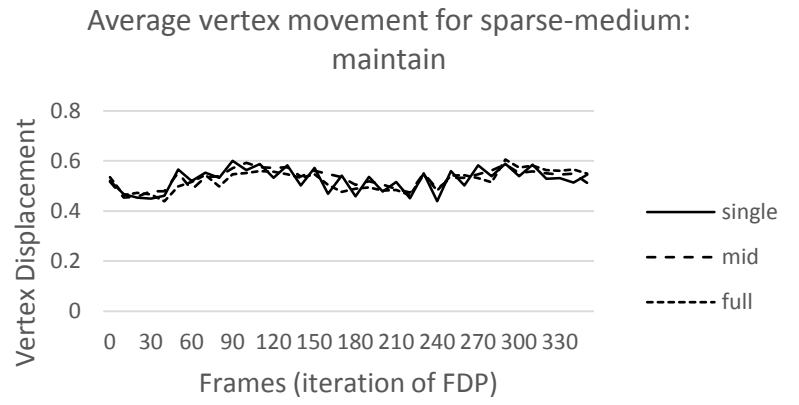
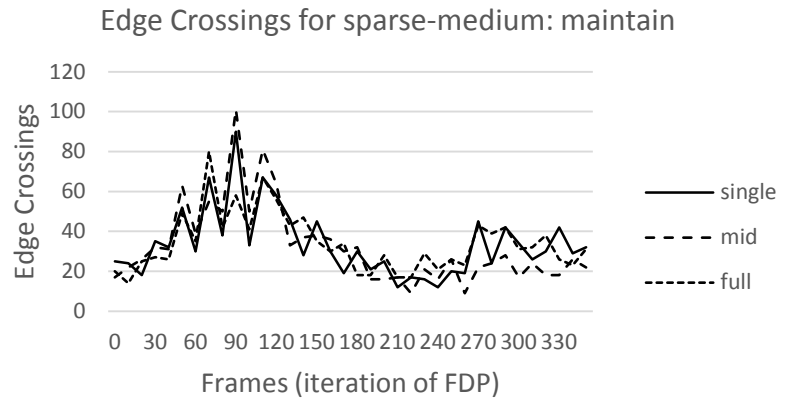
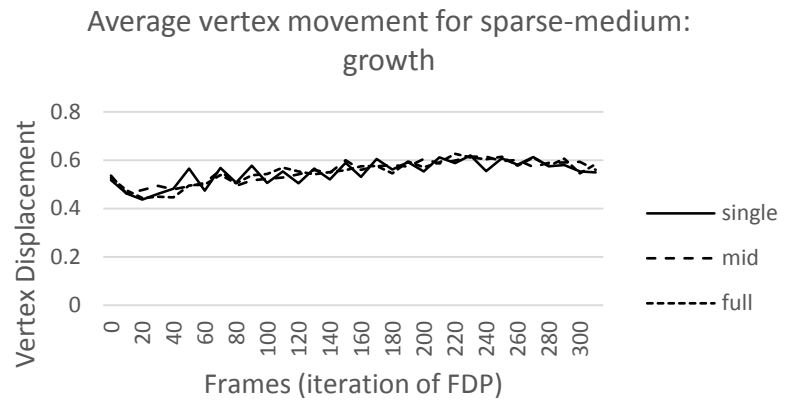
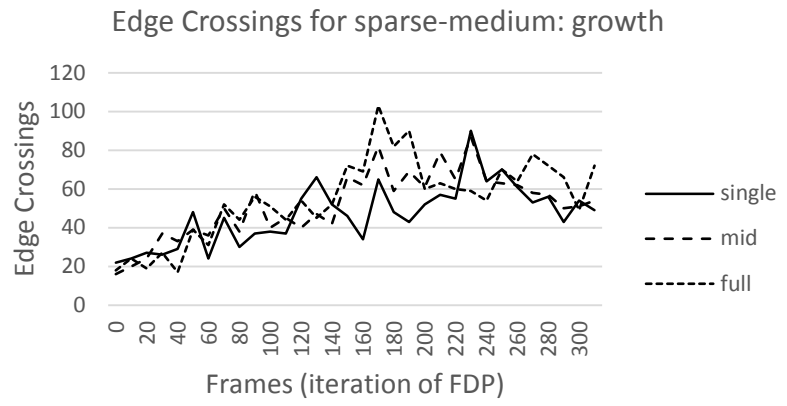


10.13.2 Medium Graphs

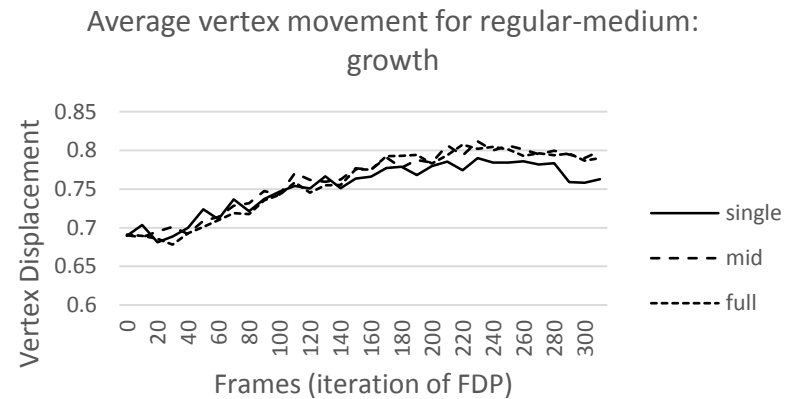
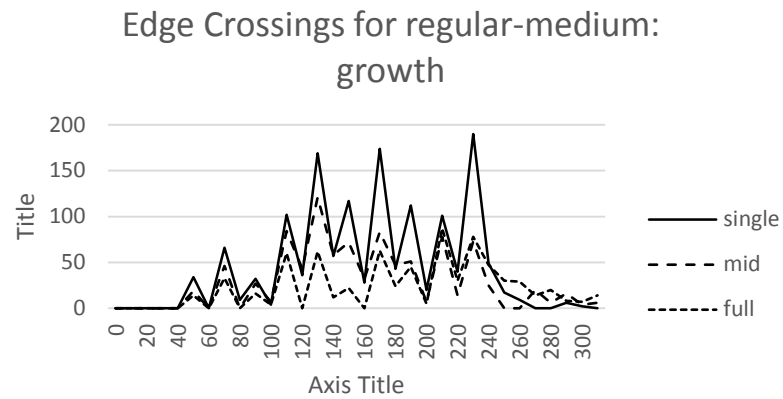
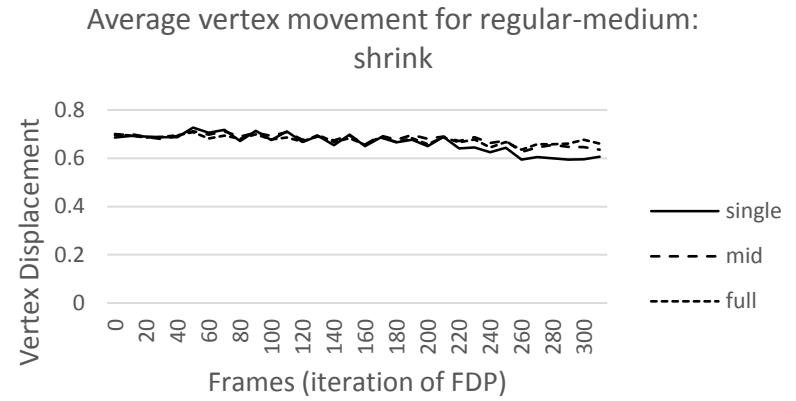
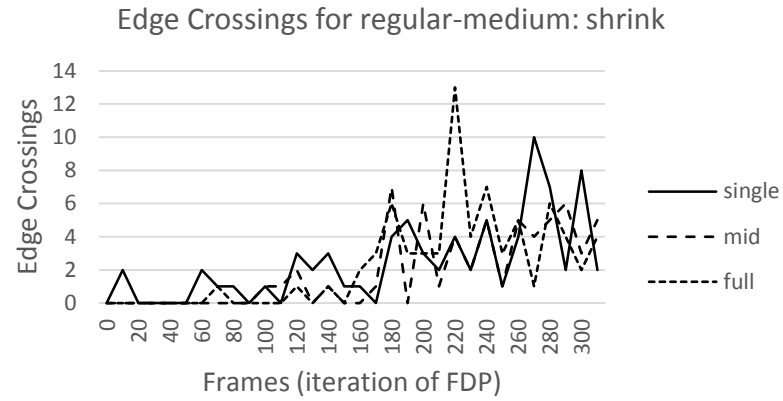
10.13.2.1 Single, Middle and Full Update Comparison

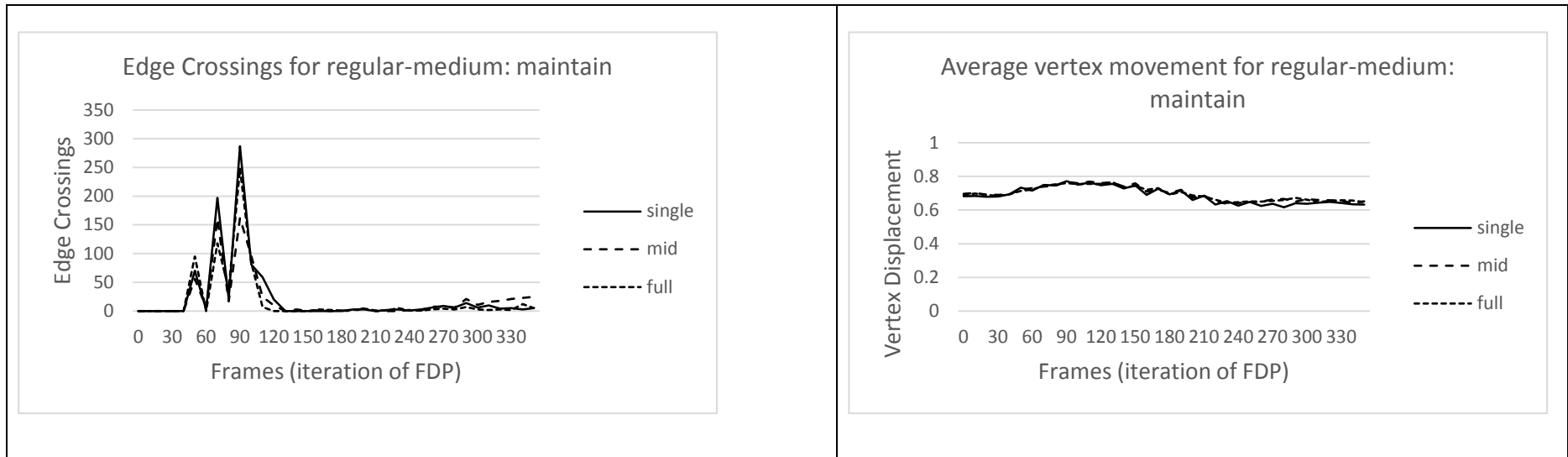
Sparse-medium





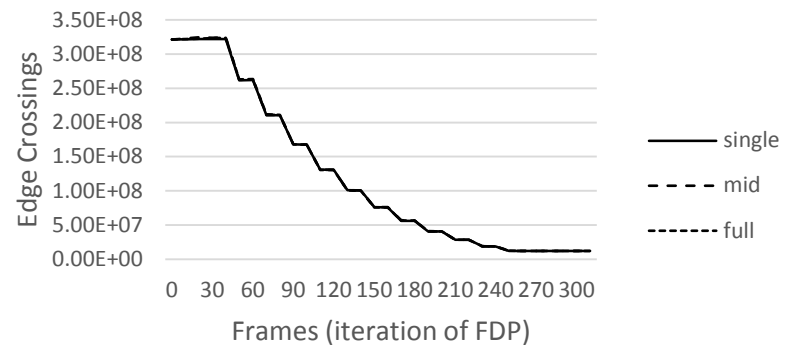
Regular-Medium



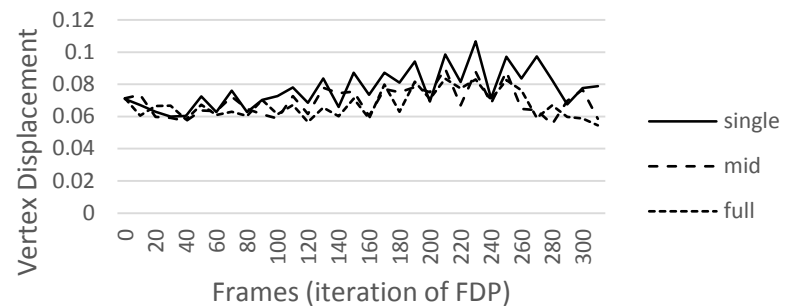


Dense-medium

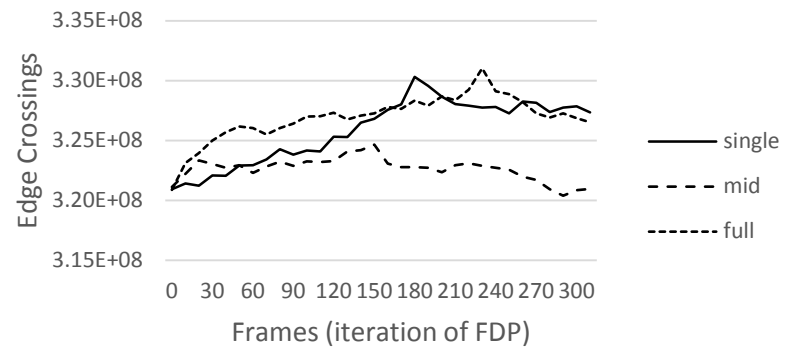
Edge Crossings for dense-medium: shrink



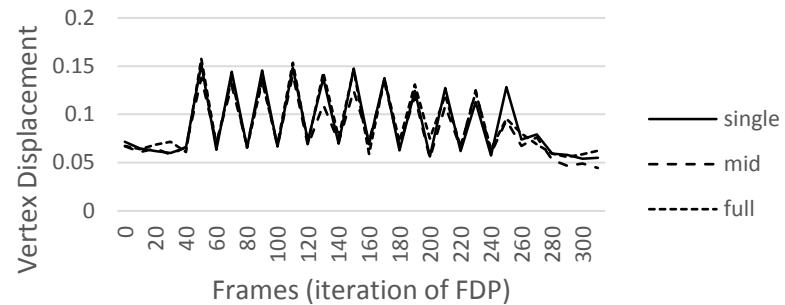
Average vertex movement for dense-medium: shrink

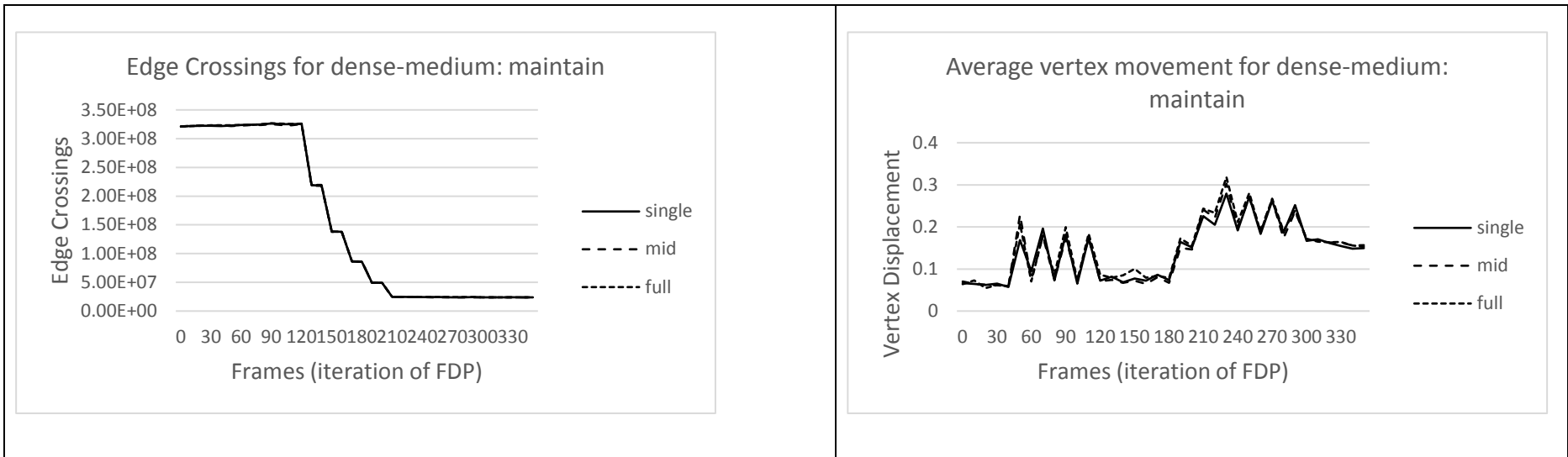


Edge Crossings for dense-medium: growth

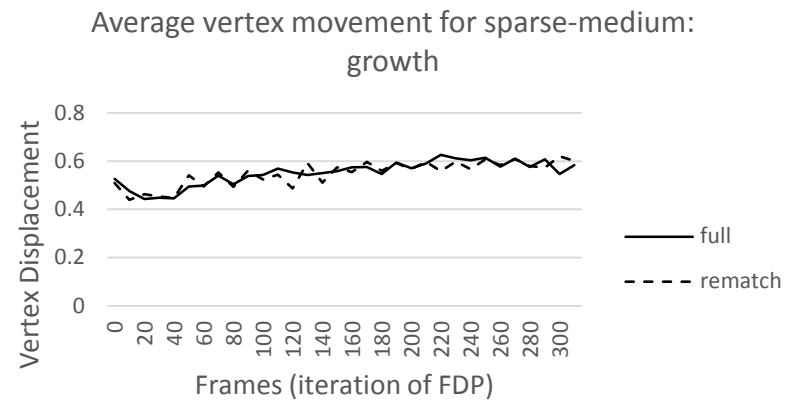
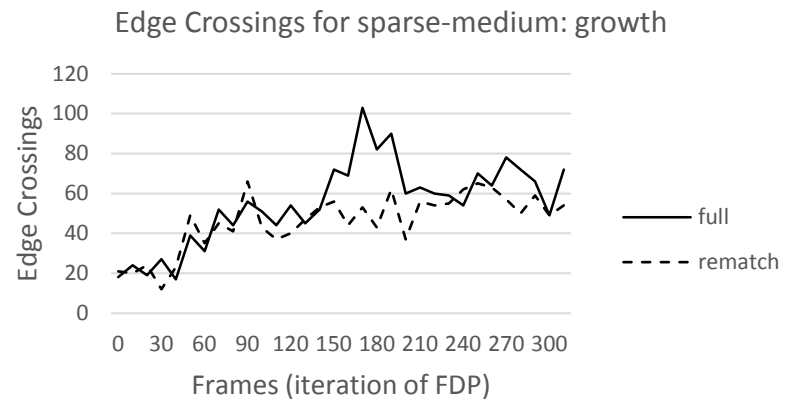
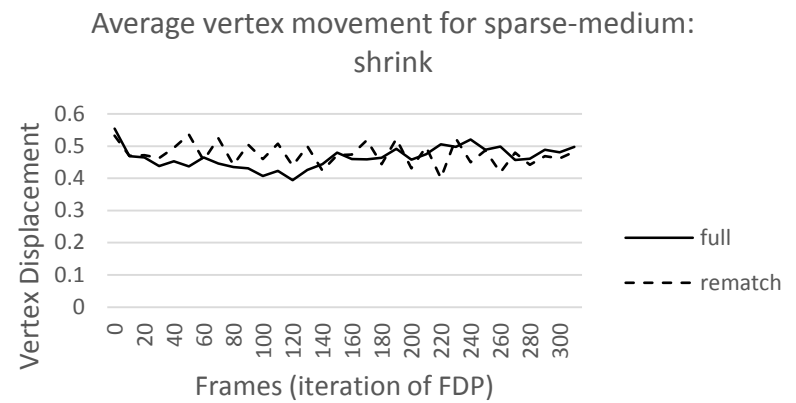
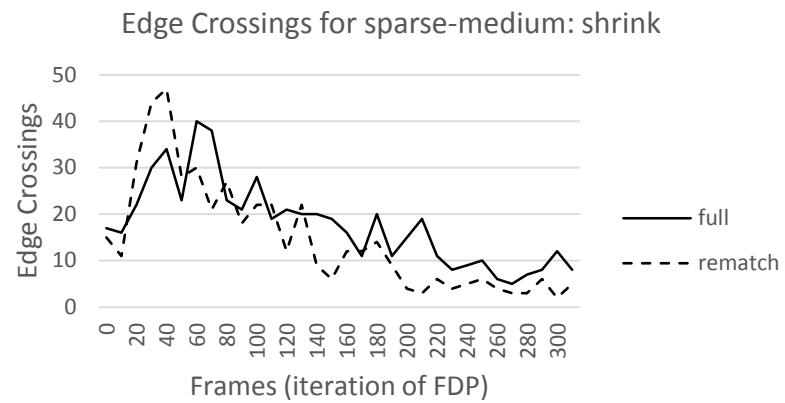


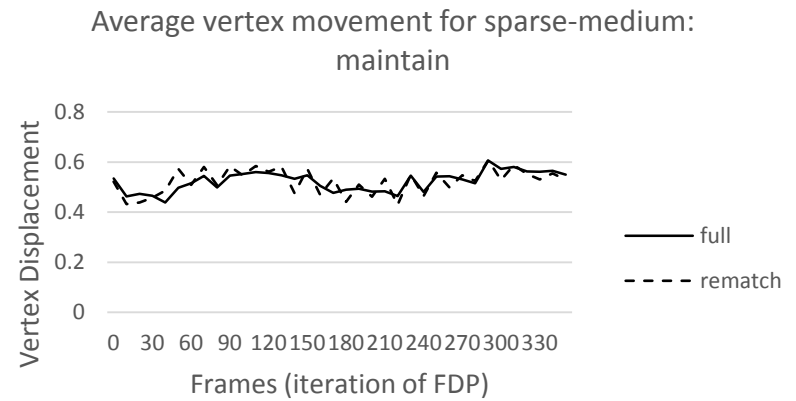
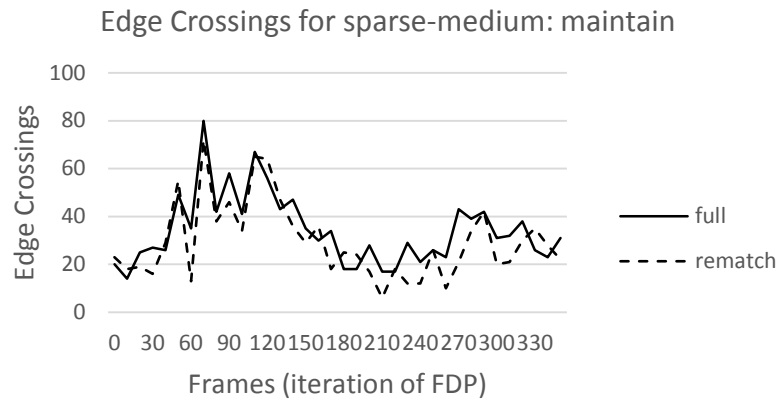
Average vertex movement for dense-medium: growth



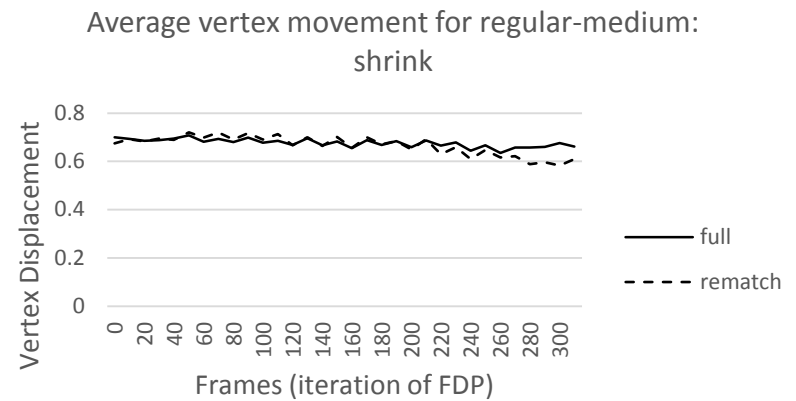
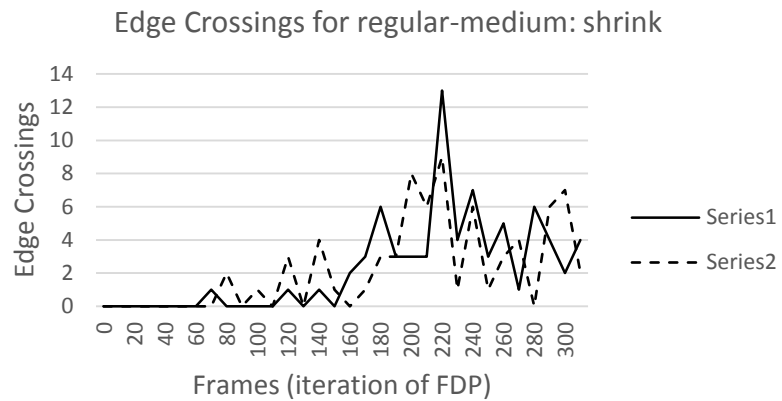


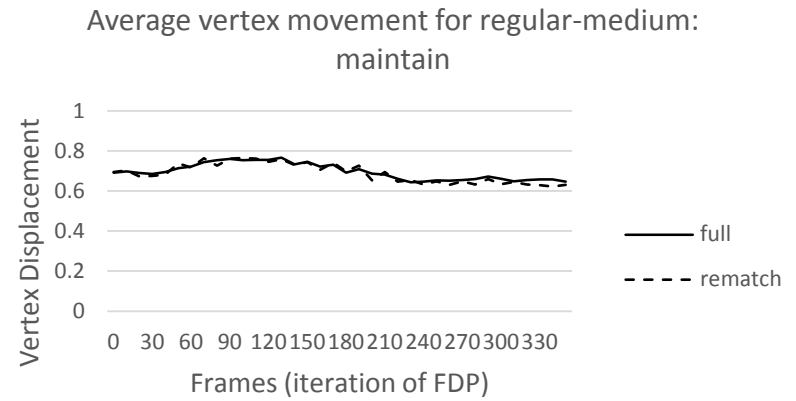
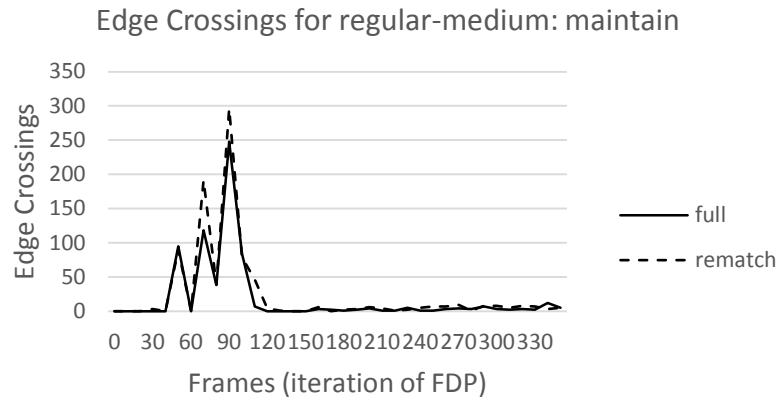
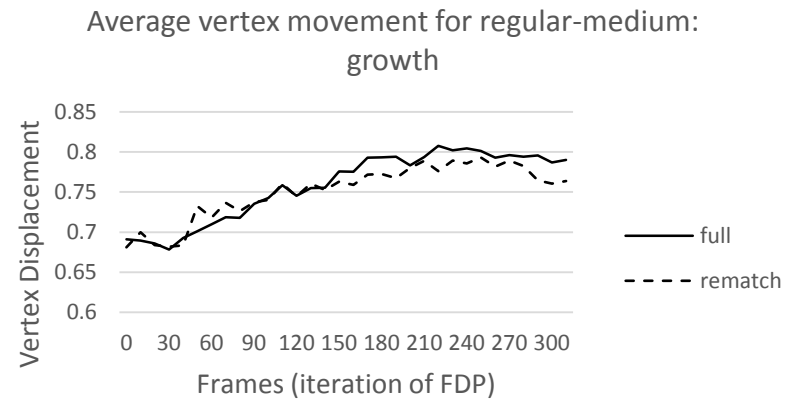
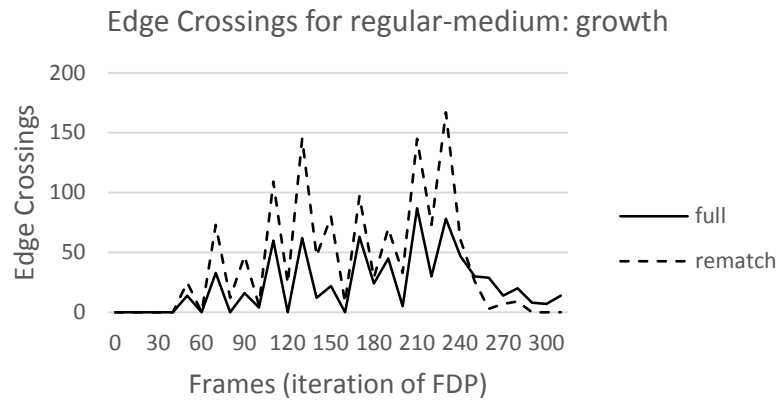
10.13.2.2 Full and Rematch Update Comparison
 Sparse-medium





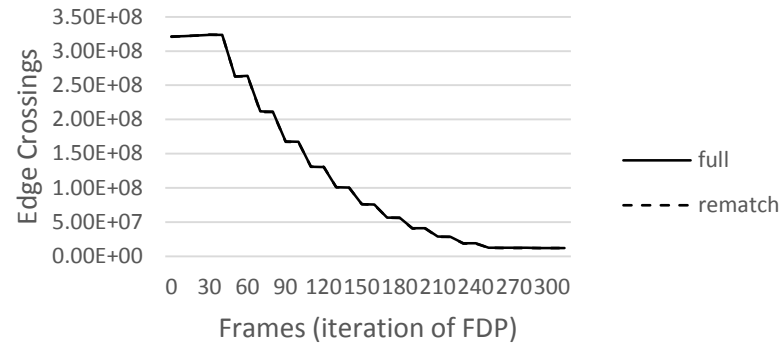
Regular-medium



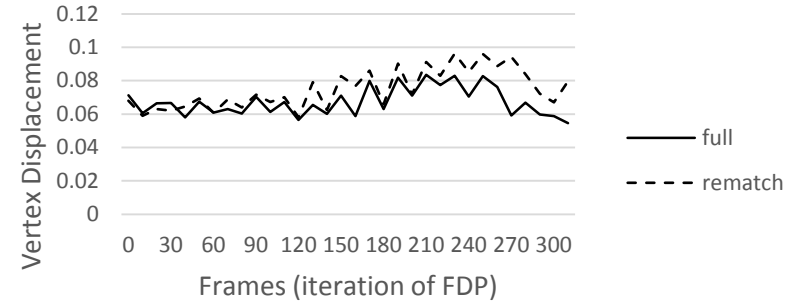


Dense-medium

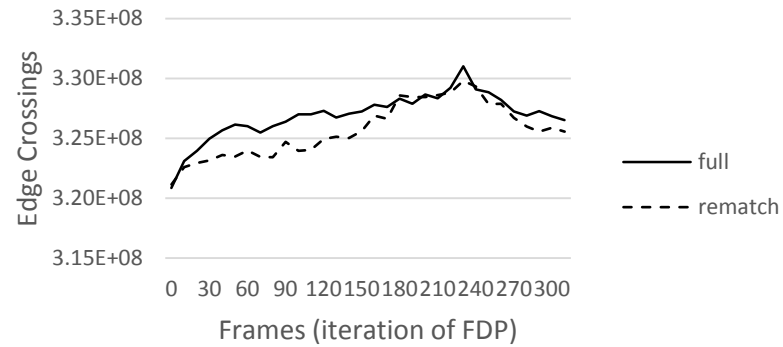
Edge Crossings for dense-medium: shrink



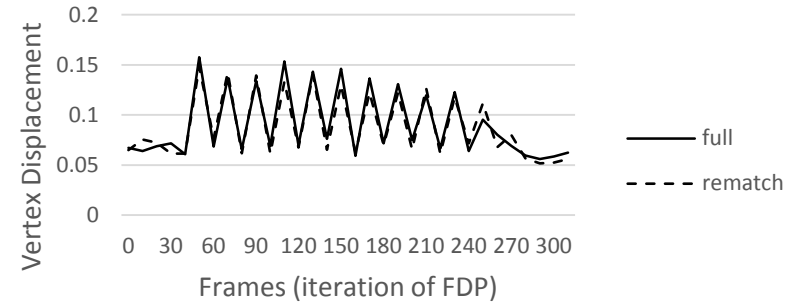
Average vertex movement for dense-medium: shrink

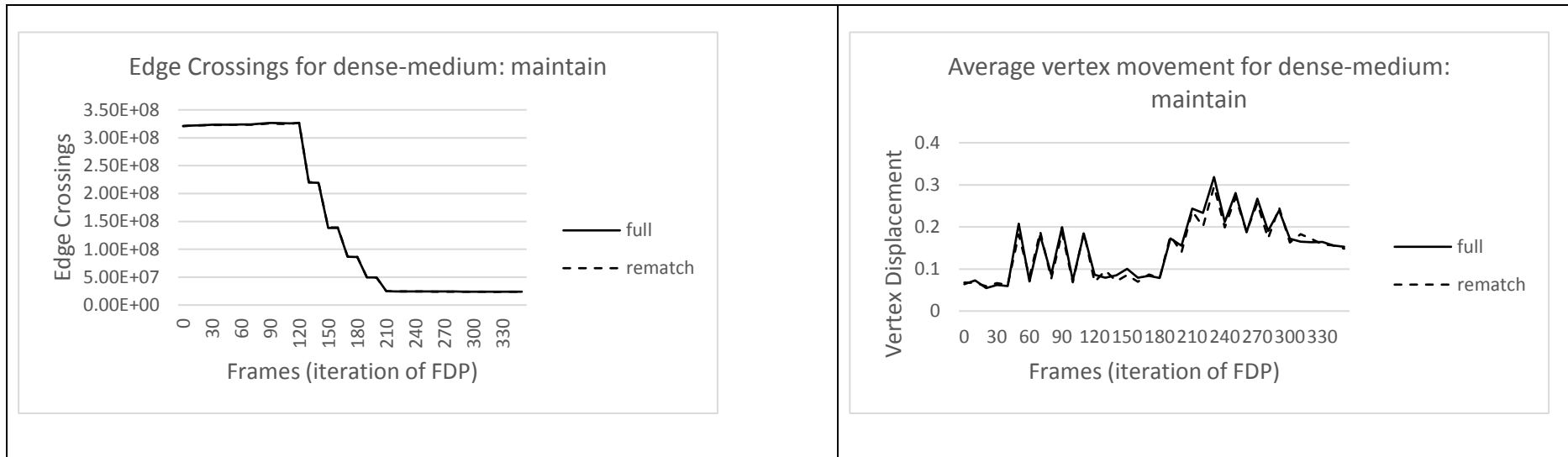


Edge Crossings for dense-medium: growth



Average vertex movement for dense-medium: growth





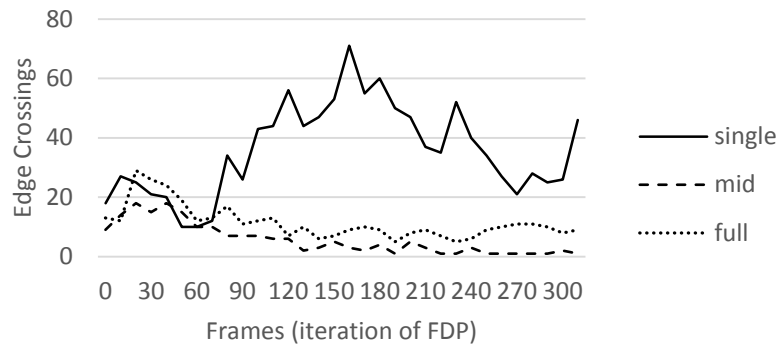
10.14 Dynamic Matching: Multilevel Global Force Updates

10.14.1 Small Graphs

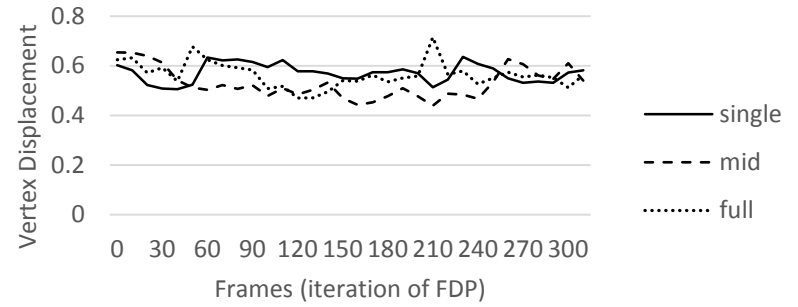
10.14.1.1 Single, Middle and Full Update Comparison

Sparse

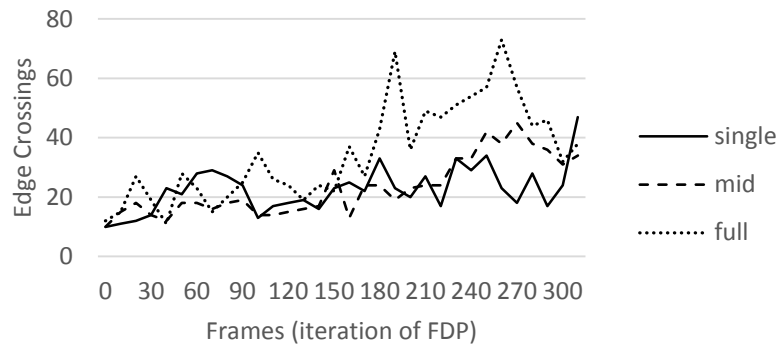
Edge Crossings for sparse-small: shrink



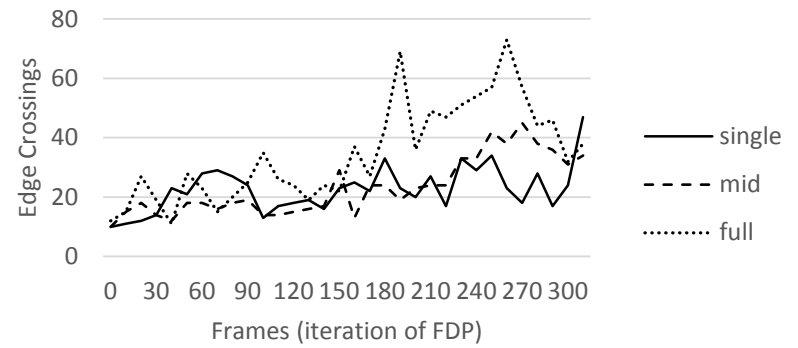
Average vertex movement for sparse-small: shrink

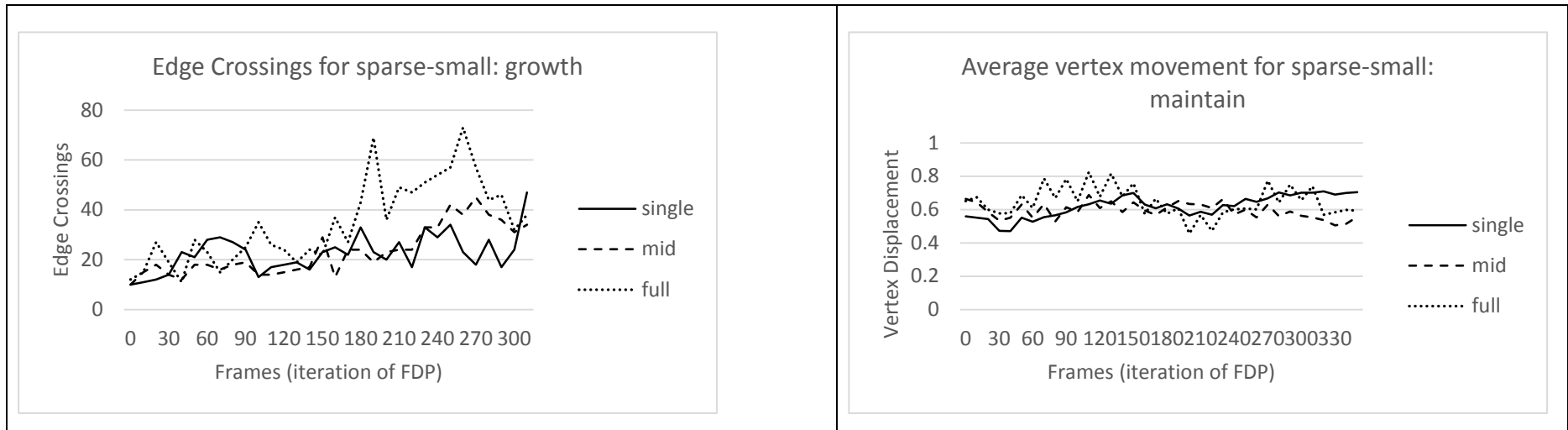


Edge Crossings for sparse-small: growth

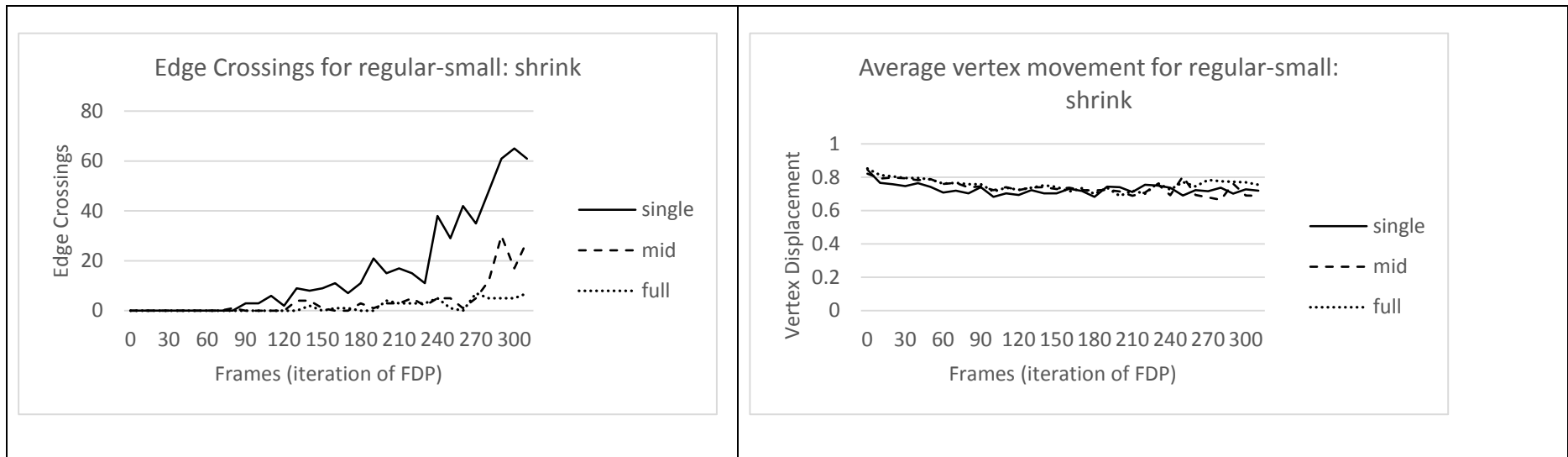


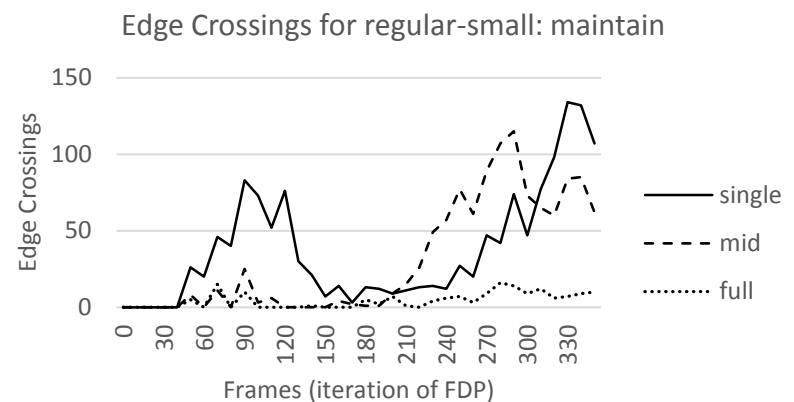
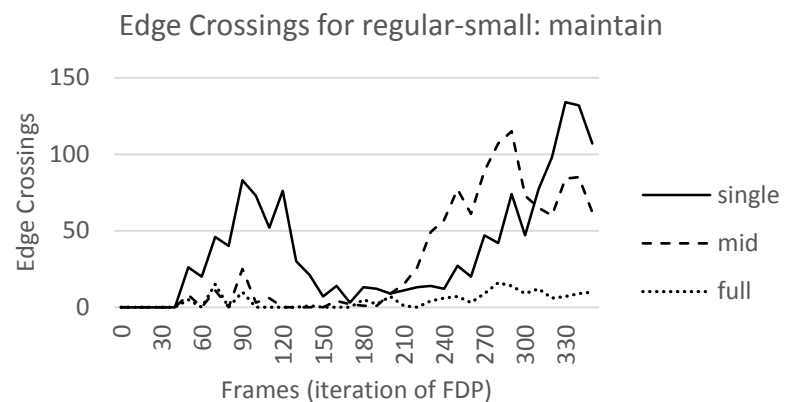
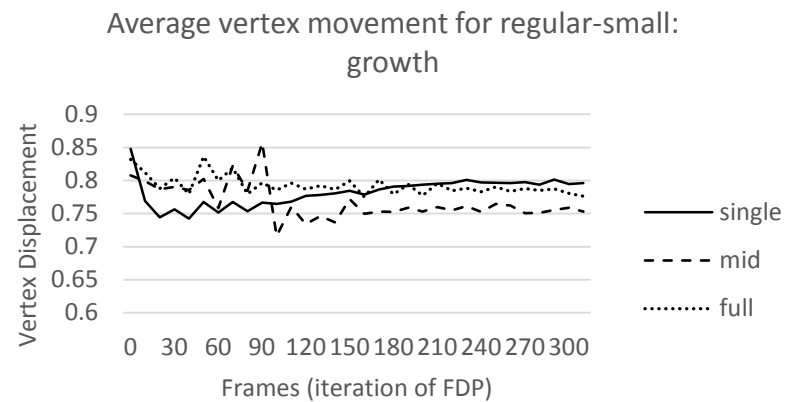
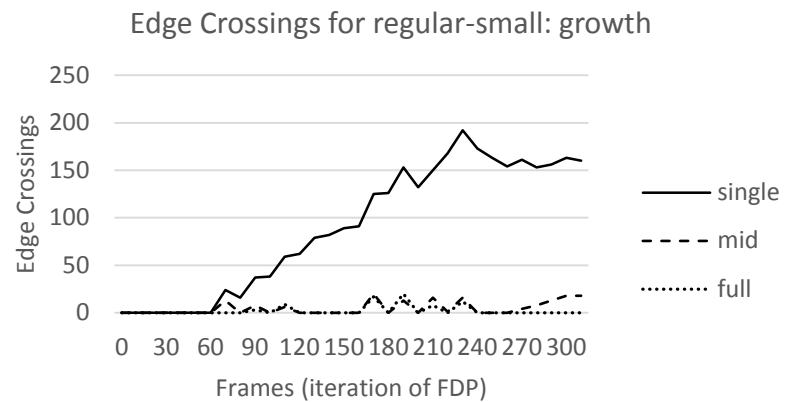
Edge Crossings for sparse-small: growth



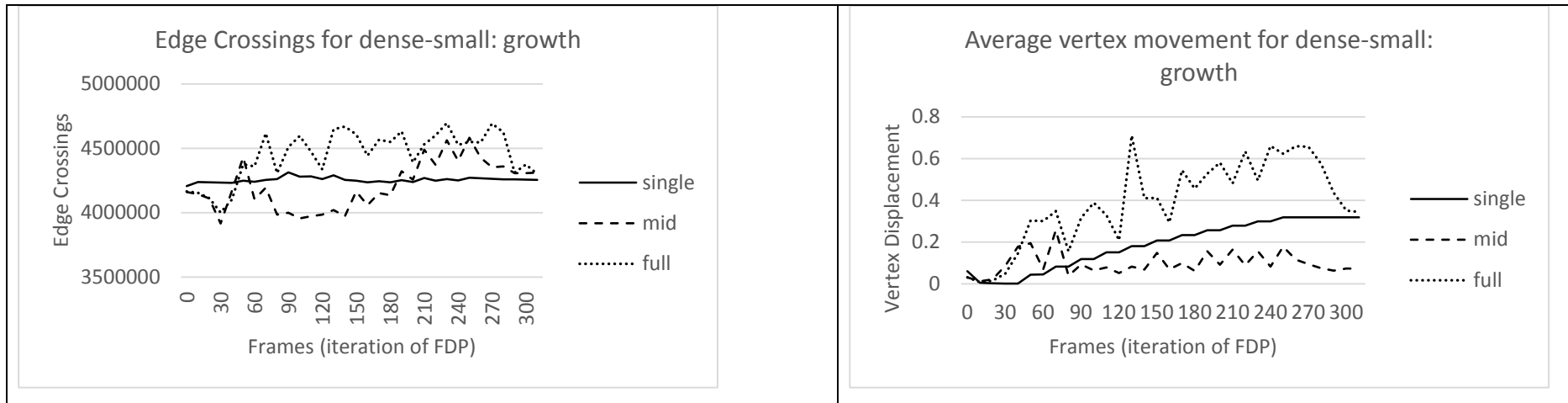
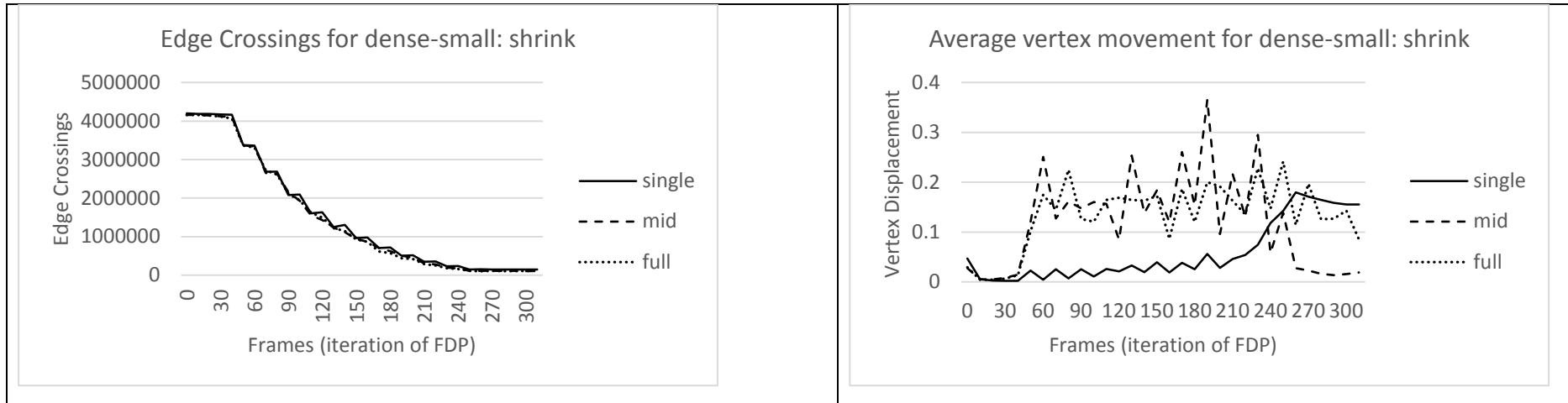


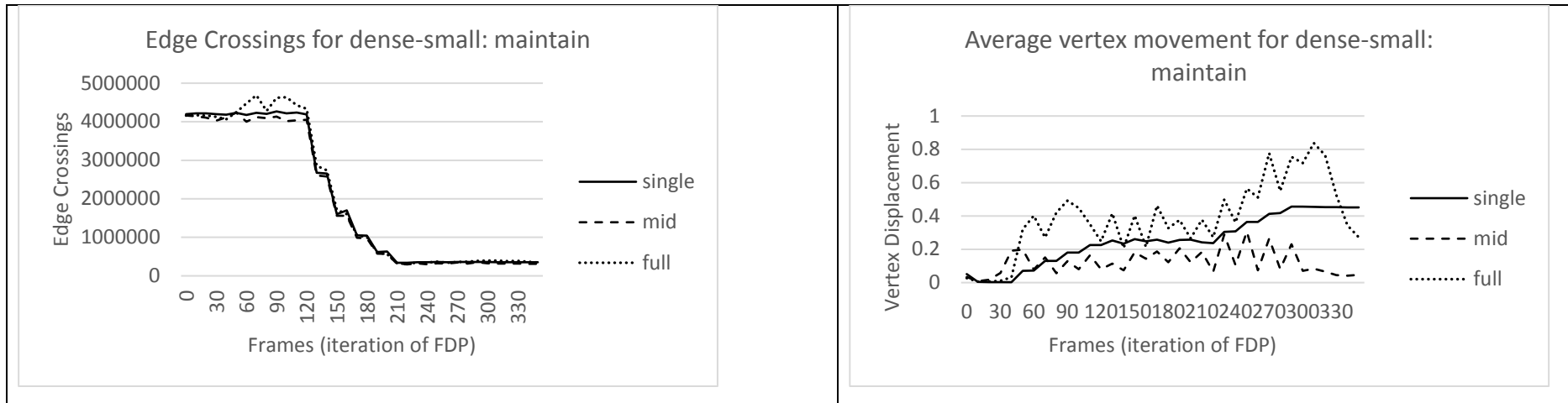
Regular





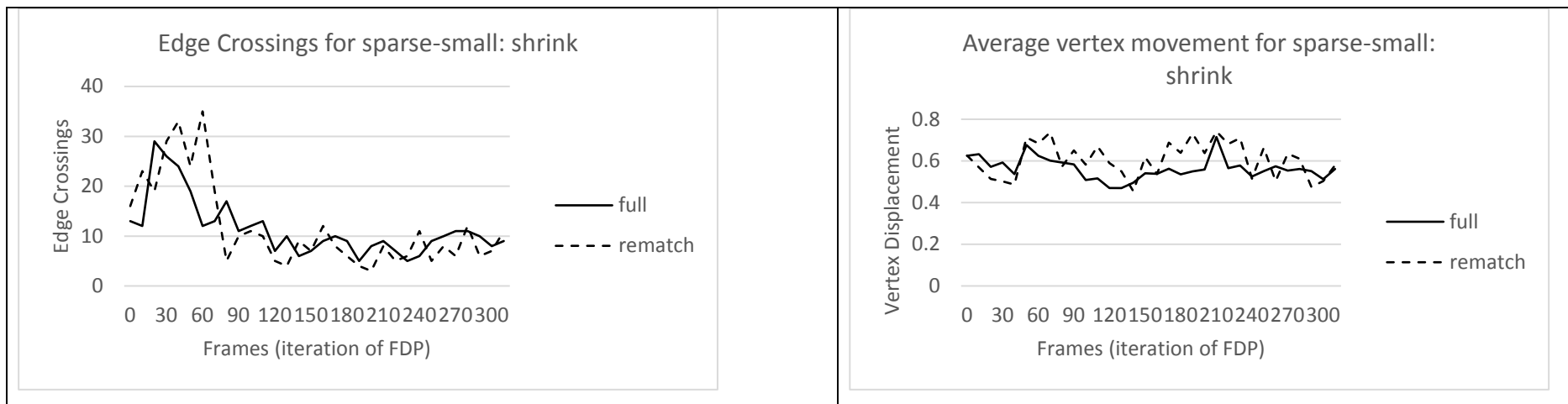
Dense



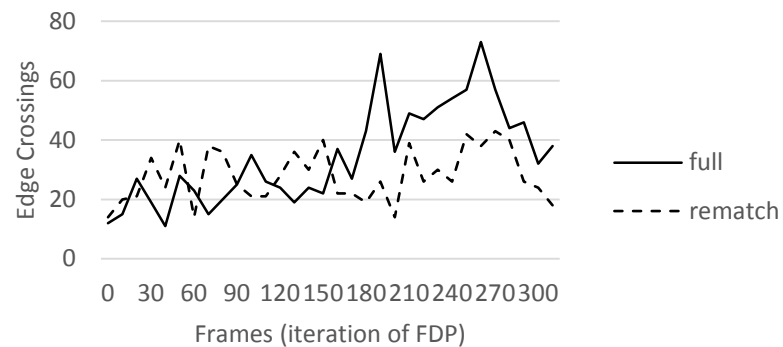


10.14.1.2 Full and Rematch Update Comparison

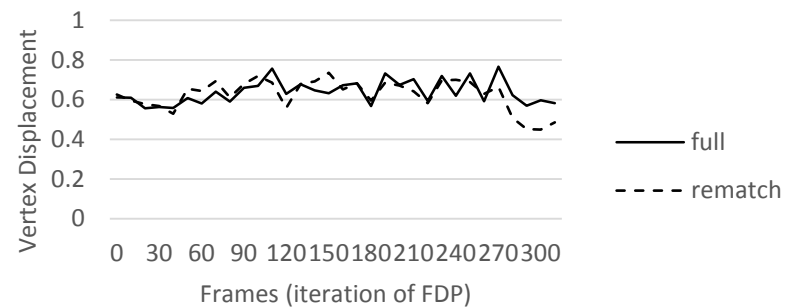
Sparse



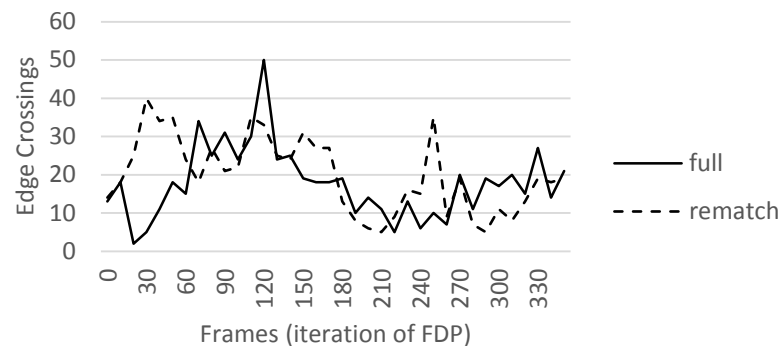
Edge Crossings for sparse-small: growth



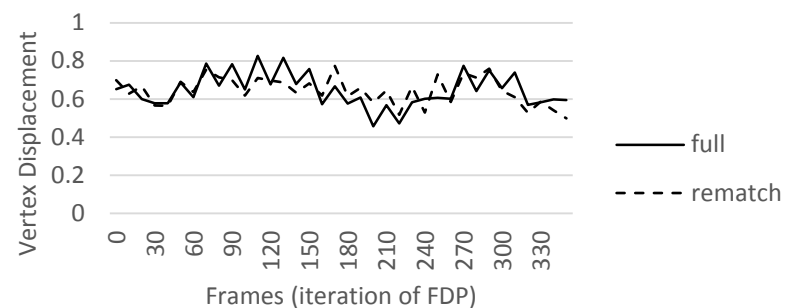
Average vertex movement for sparse-small: growth



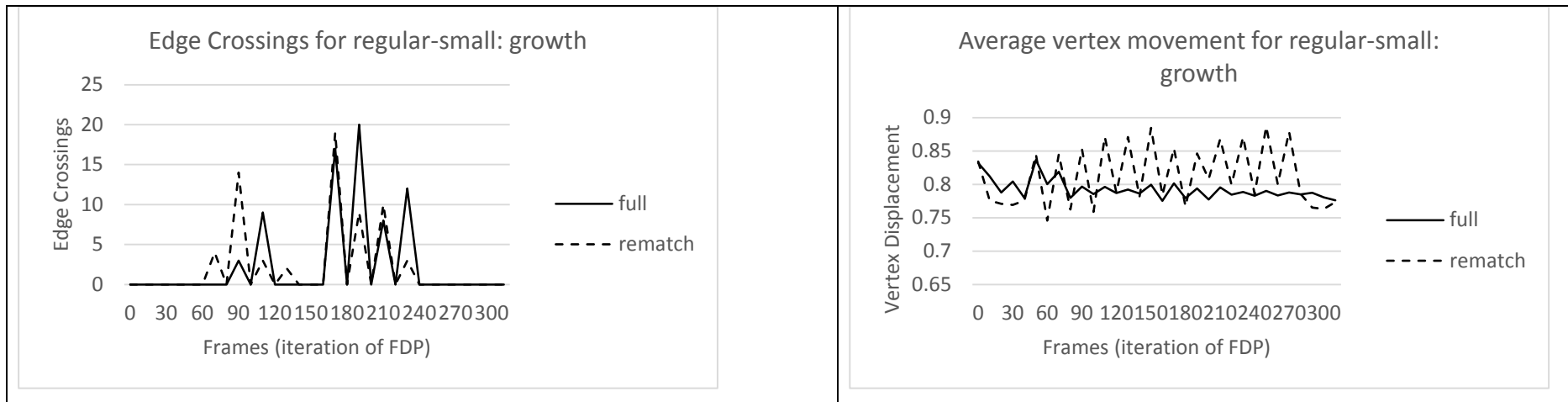
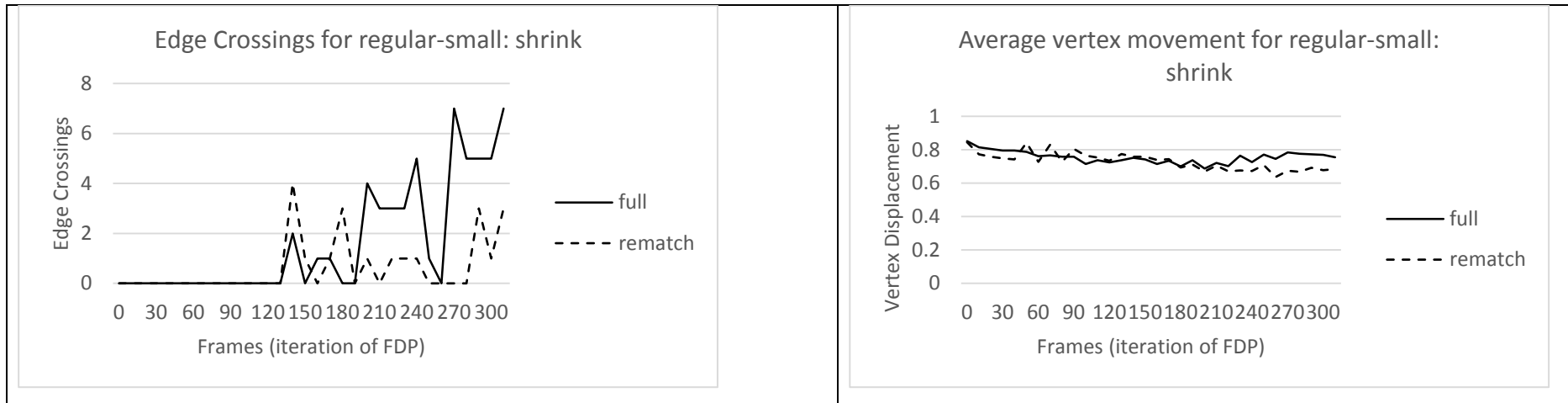
Edge Crossings for sparse-small: maintain

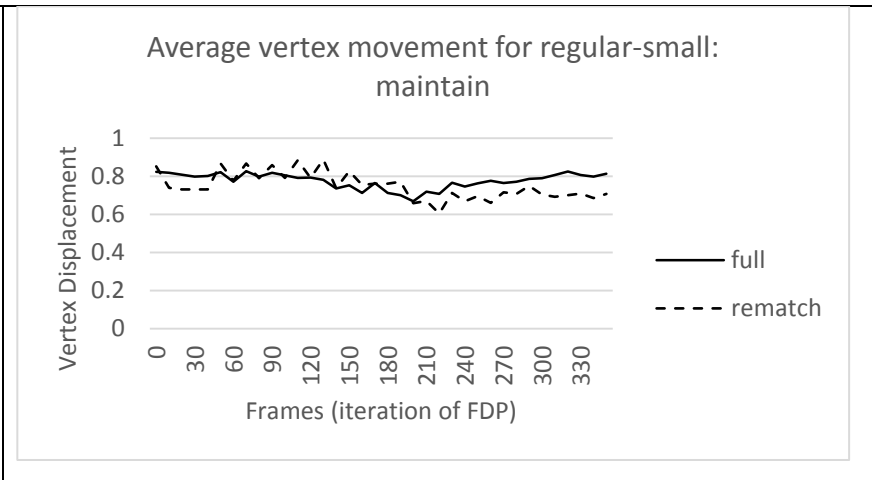
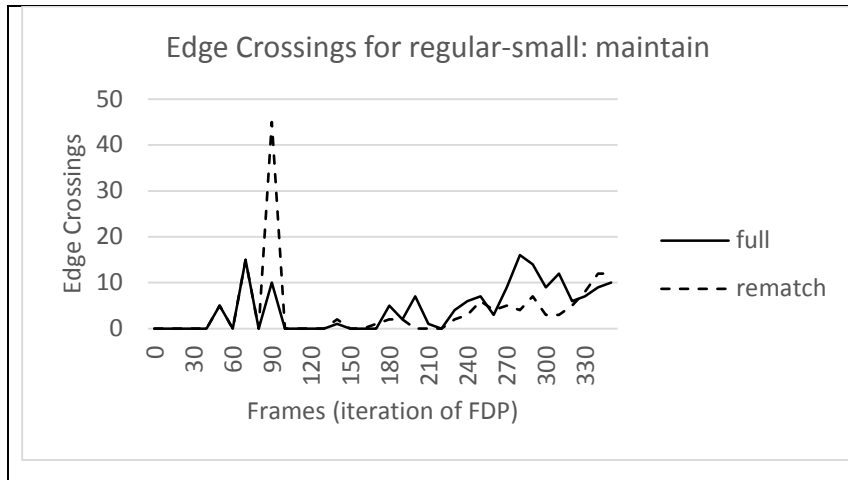


Average vertex movement for sparse-small: maintain

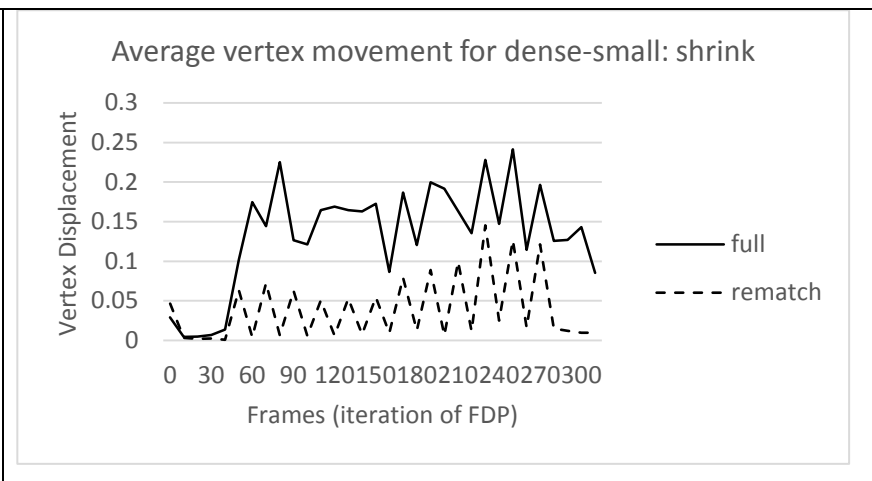
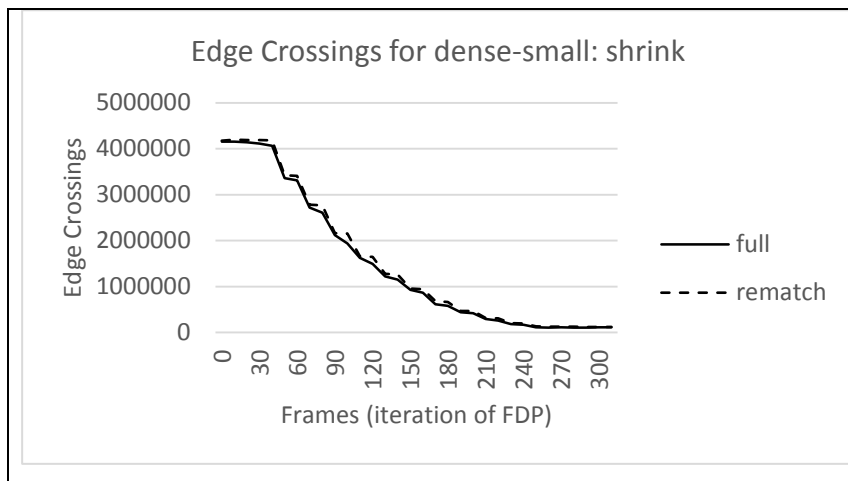


Regular

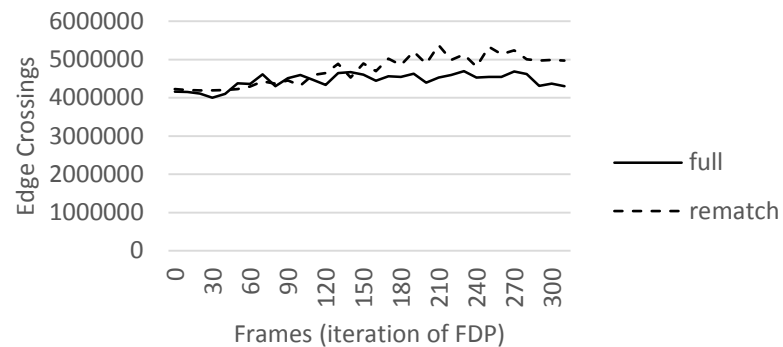




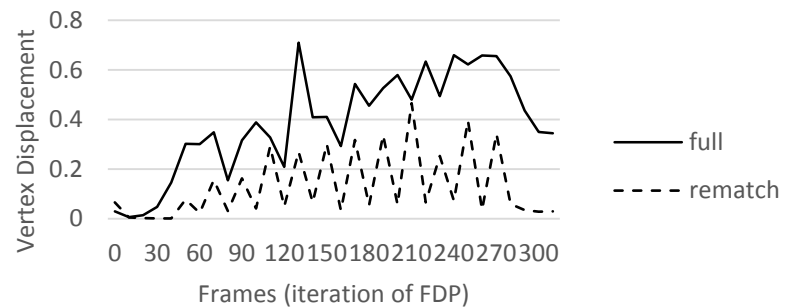
Dense



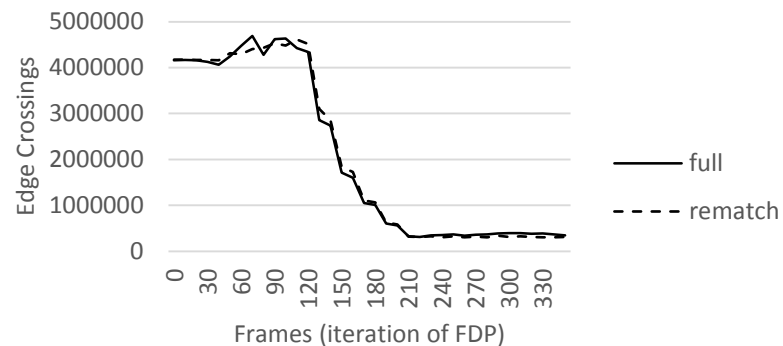
Edge Crossings for dense-small: growth



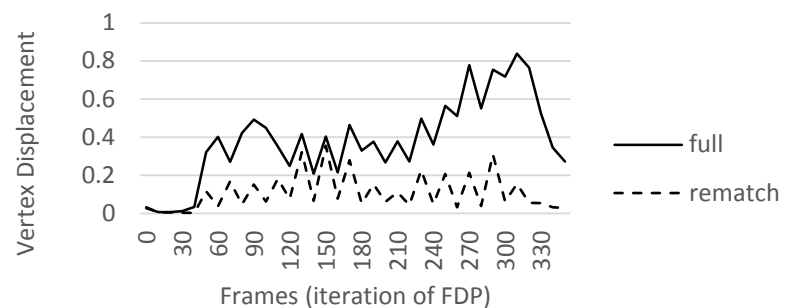
Average vertex movement for dense-small: growth



Edge Crossings for dense-small: maintain



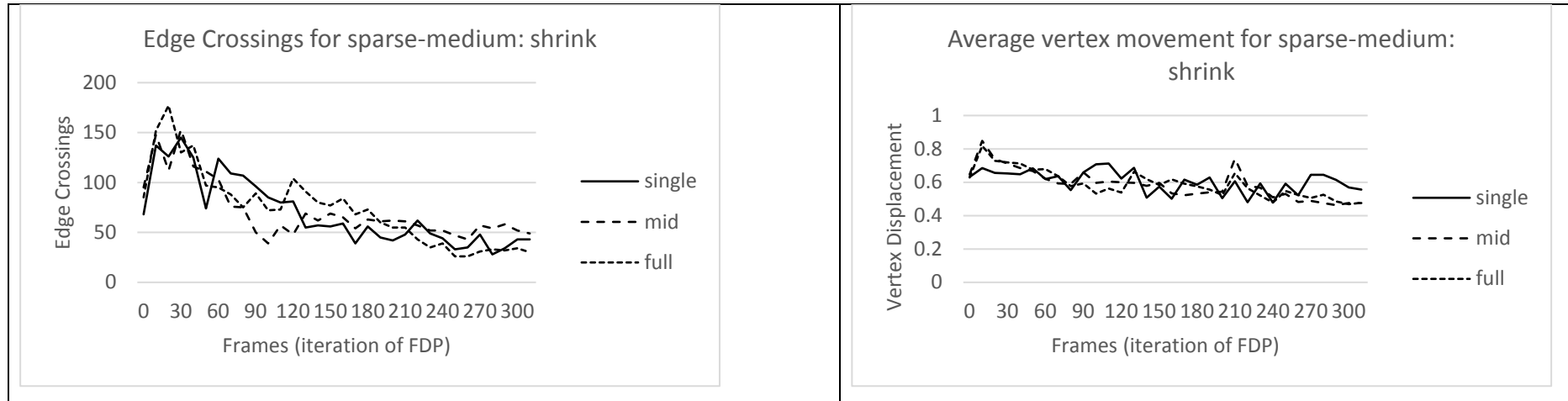
Average vertex movement for dense-small: maintain

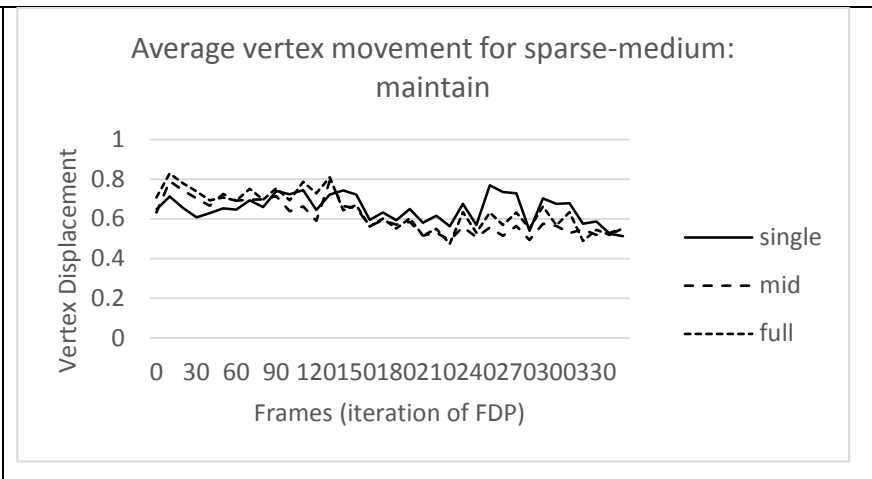
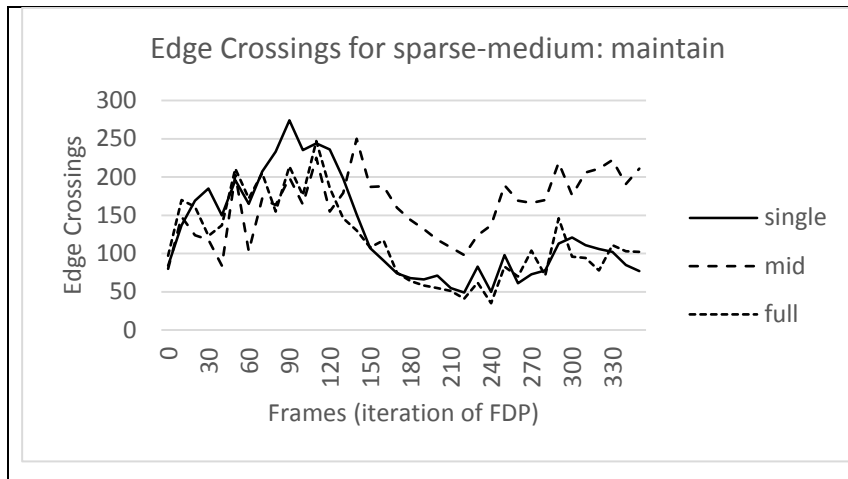
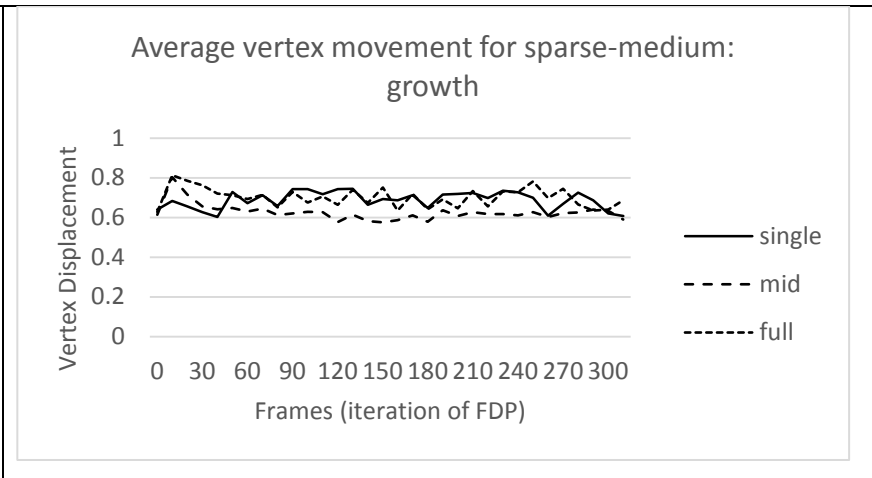
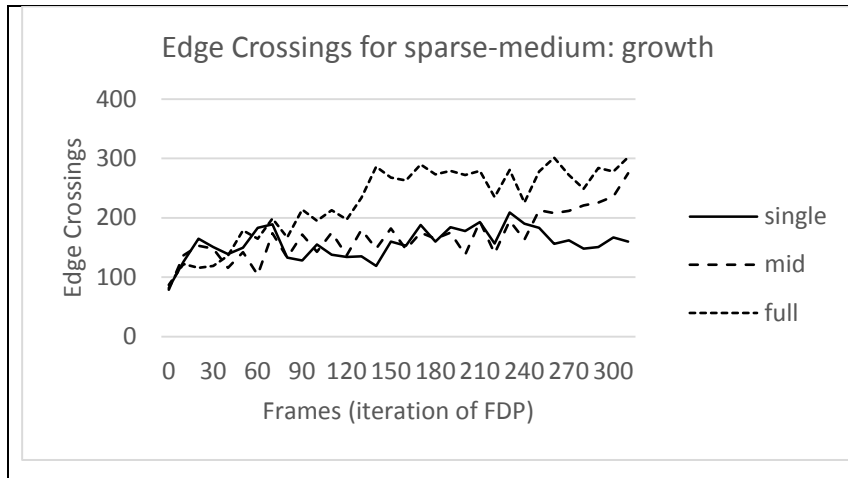


10.14.2 Medium Graphs

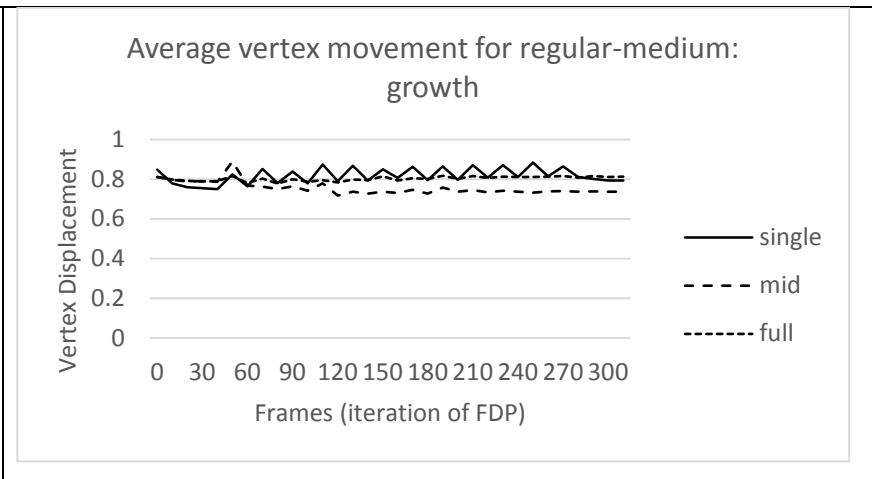
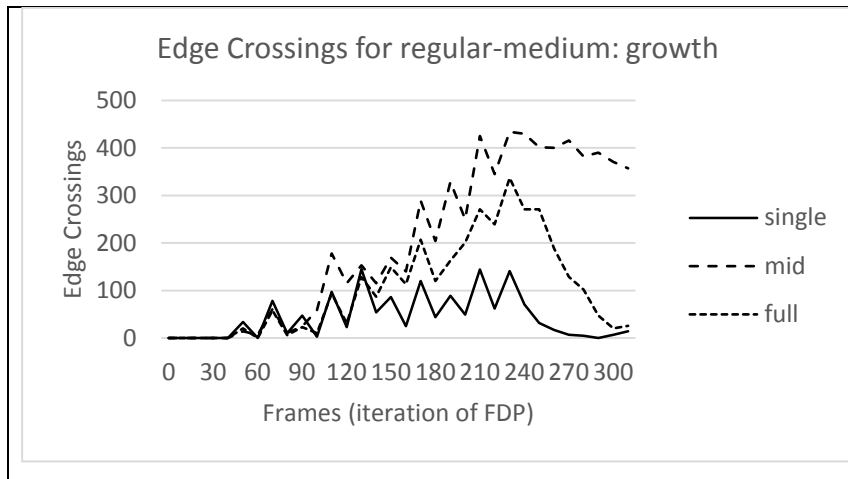
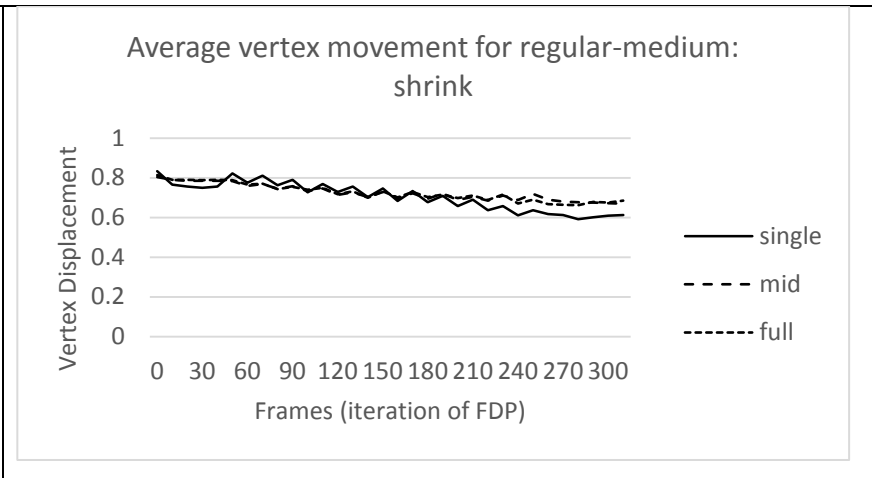
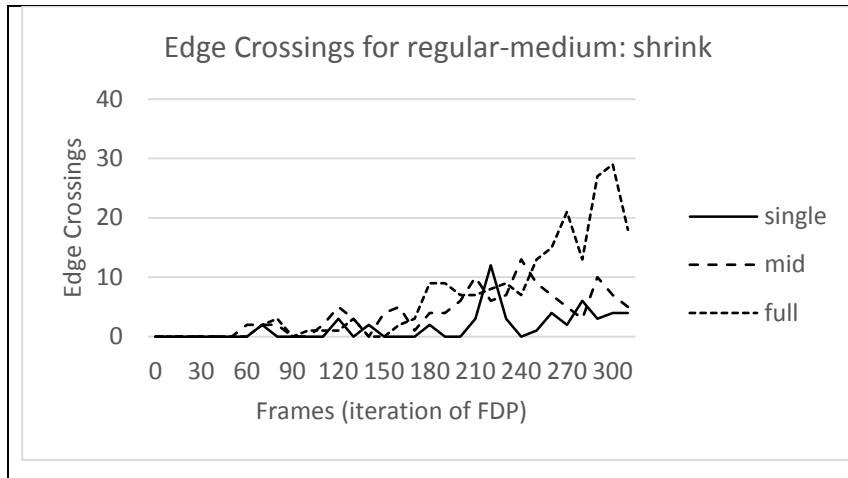
10.14.2.1 Single, Middle and Full Update Comparison

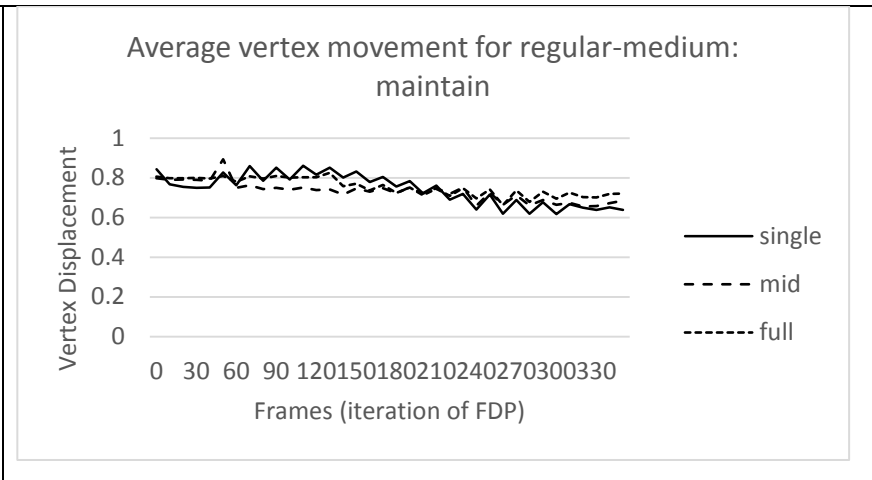
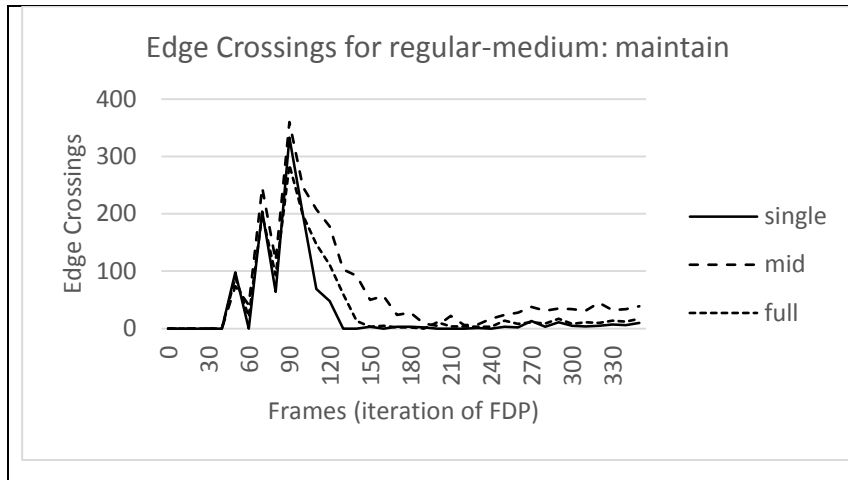
Sparse



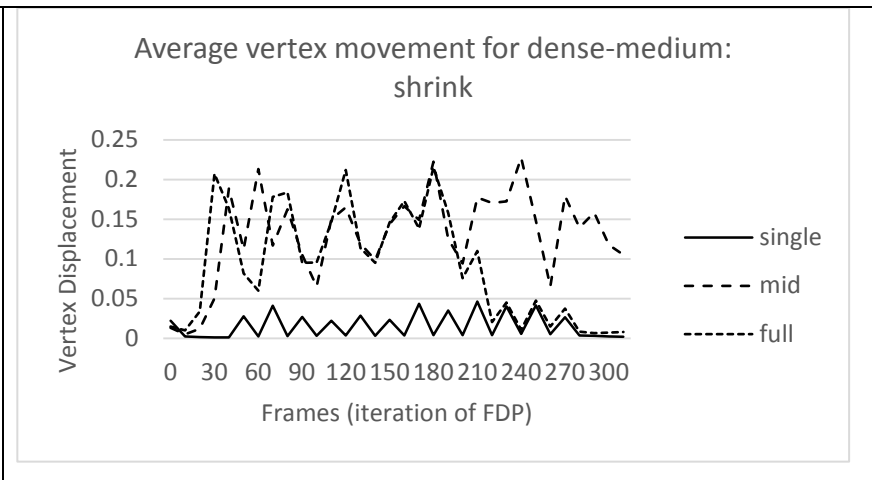
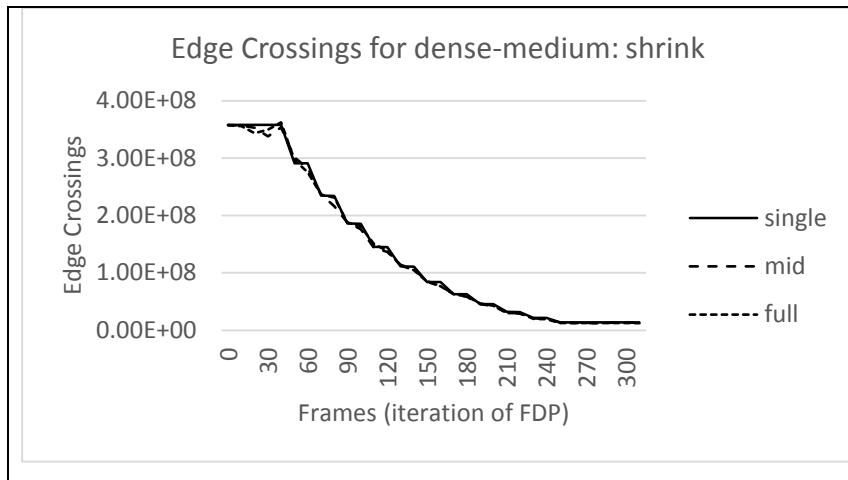


Regular

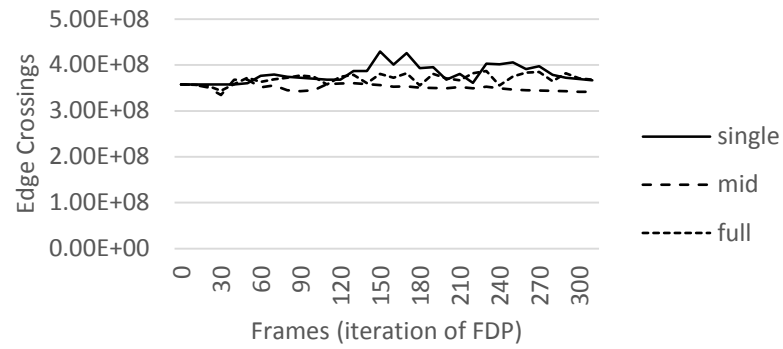




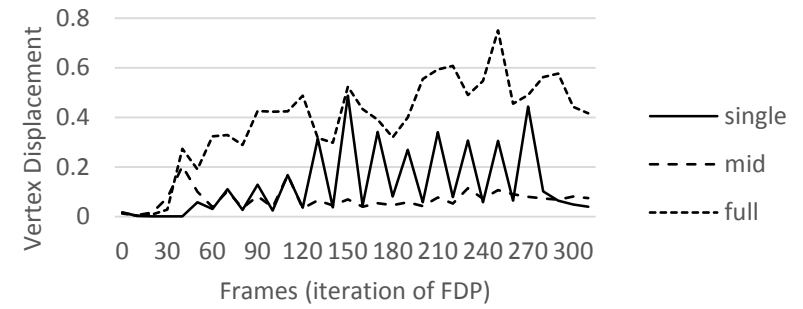
Dense



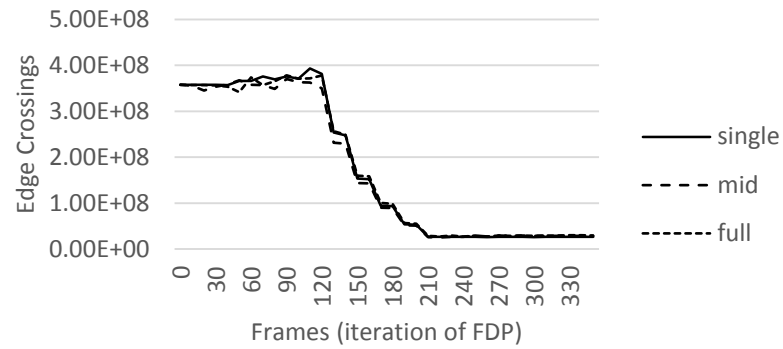
Edge Crossings for dense-medium: growth



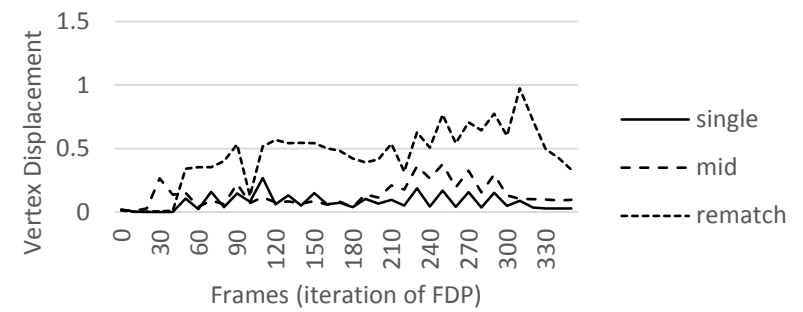
Average vertex movement for dense-medium: growth



Edge Crossings for dense-medium: maintain

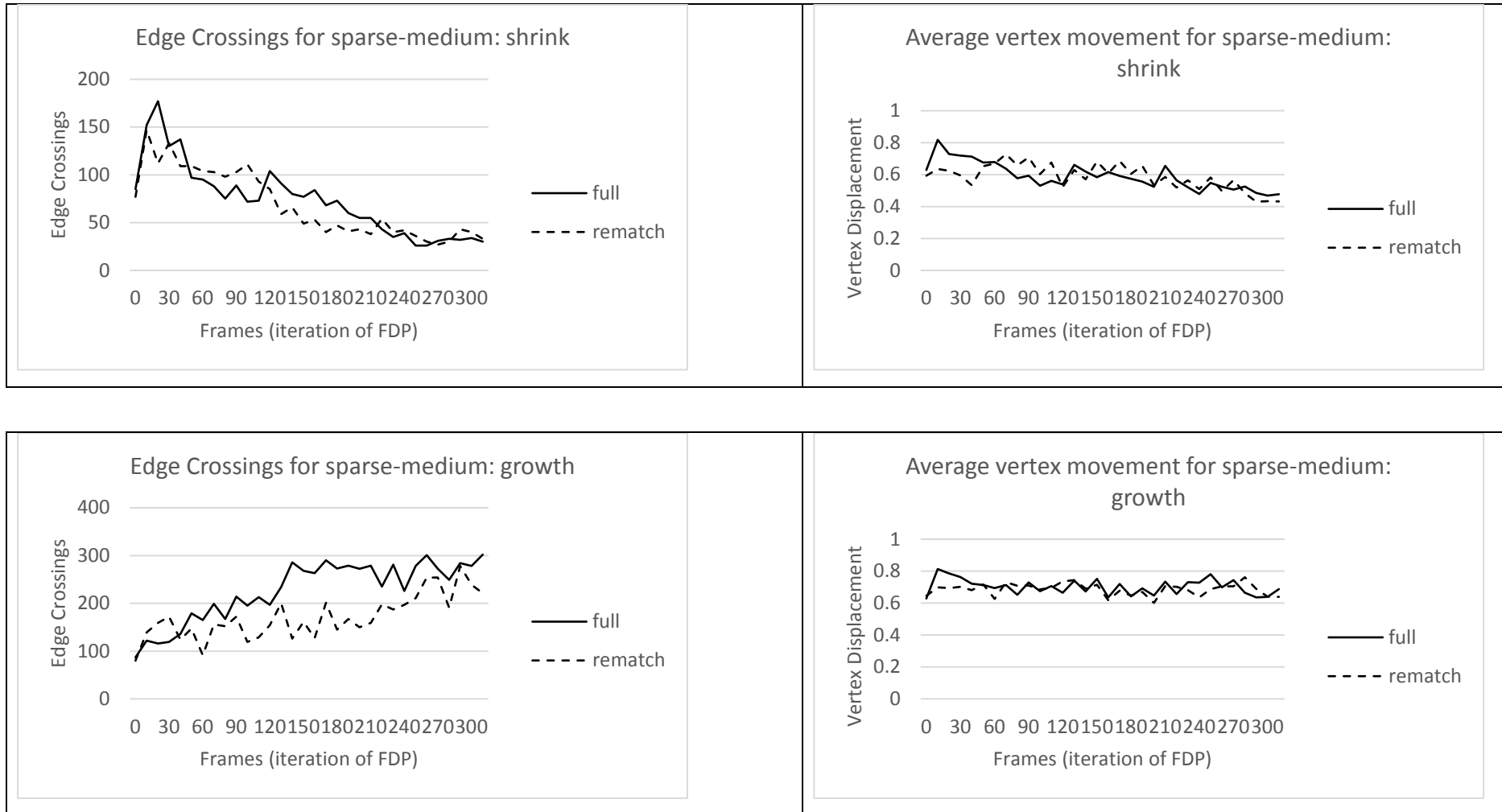


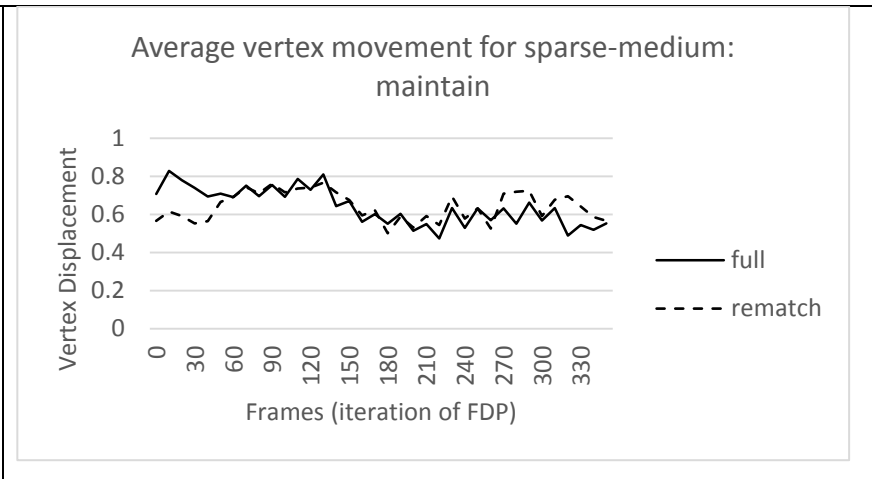
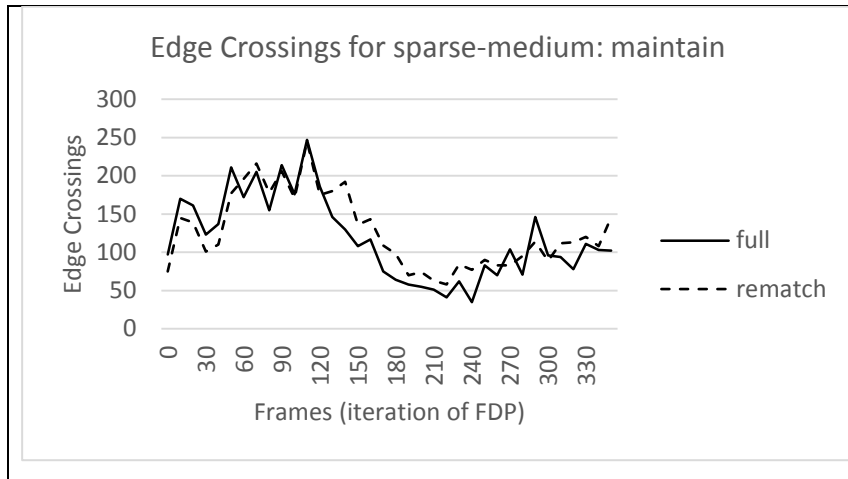
Average vertex movement for dense-medium: maintain



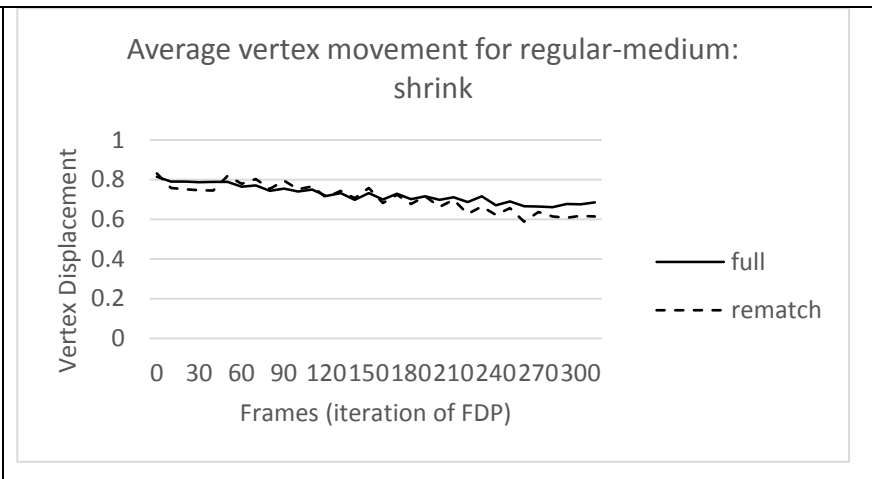
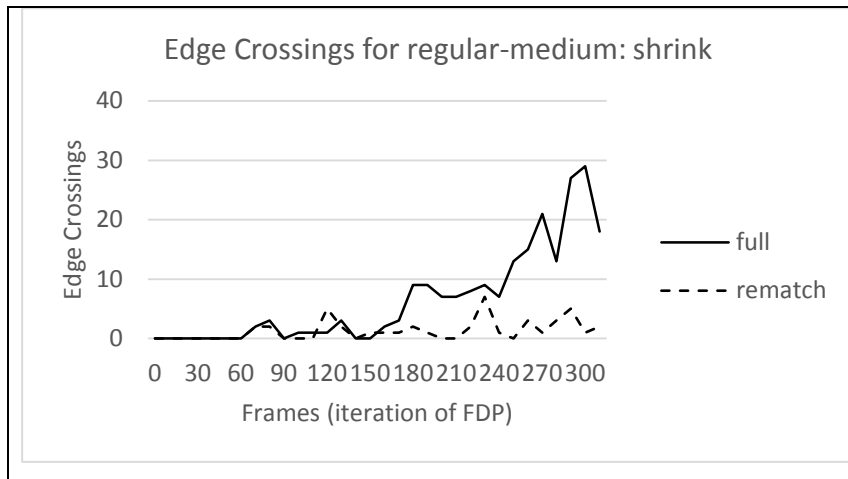
10.14.2.2 Full and Rematch Update Comparison

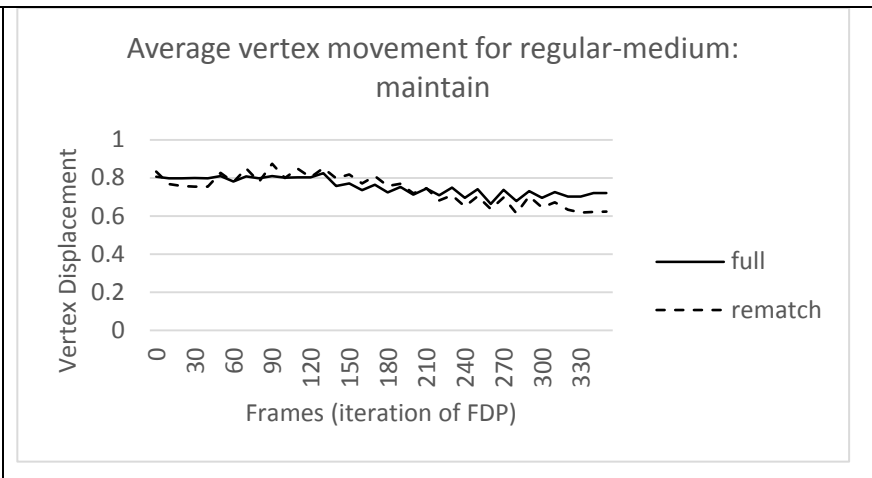
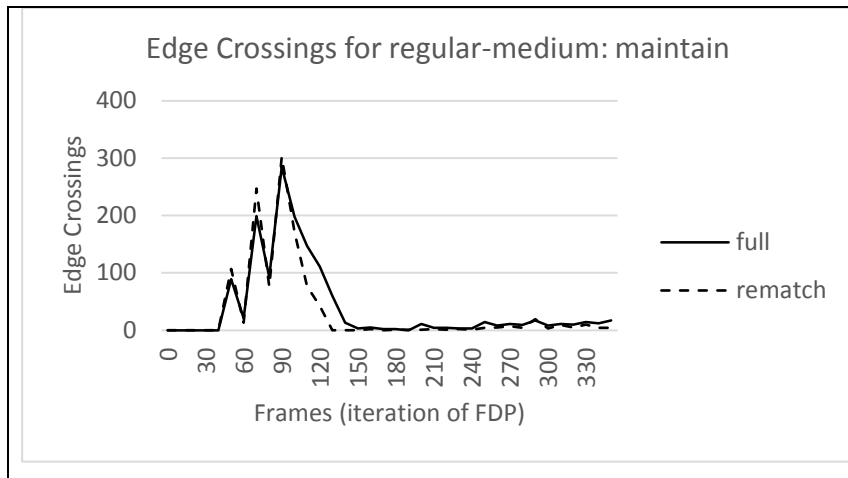
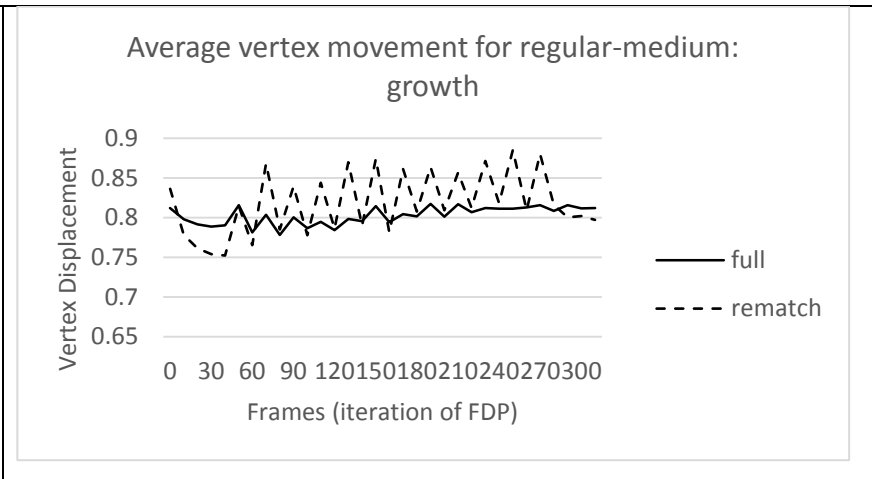
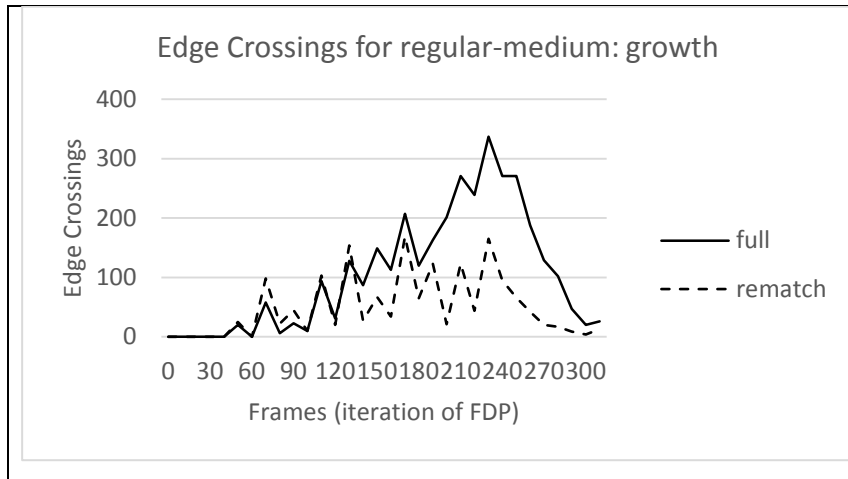
Sparse





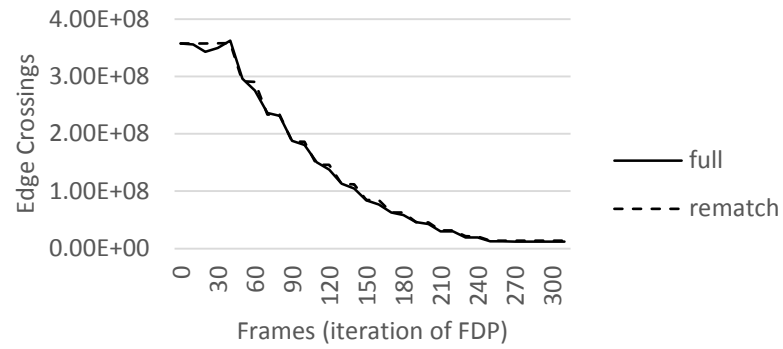
Regular



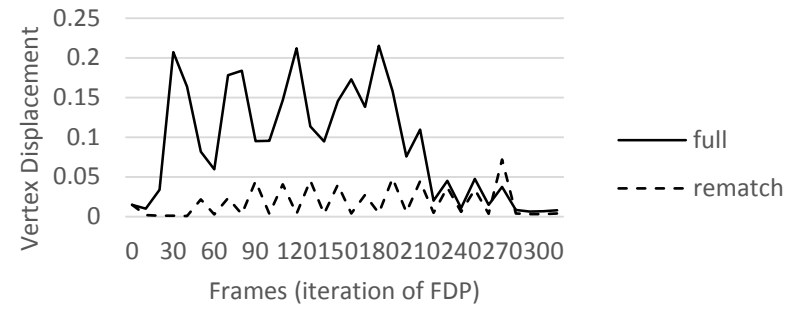


Dense

Edge Crossings for dense-medium: shrink



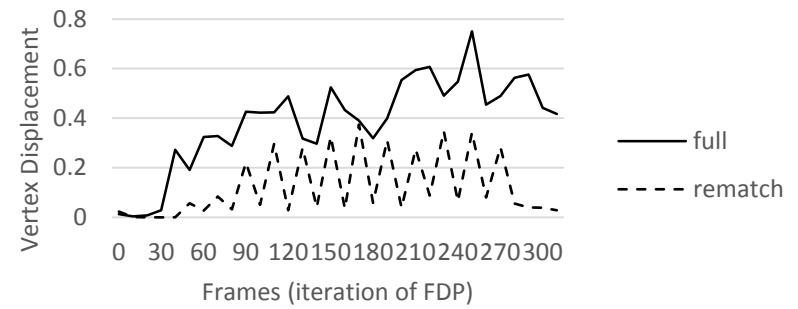
Average vertex movement for dense-medium: shrink

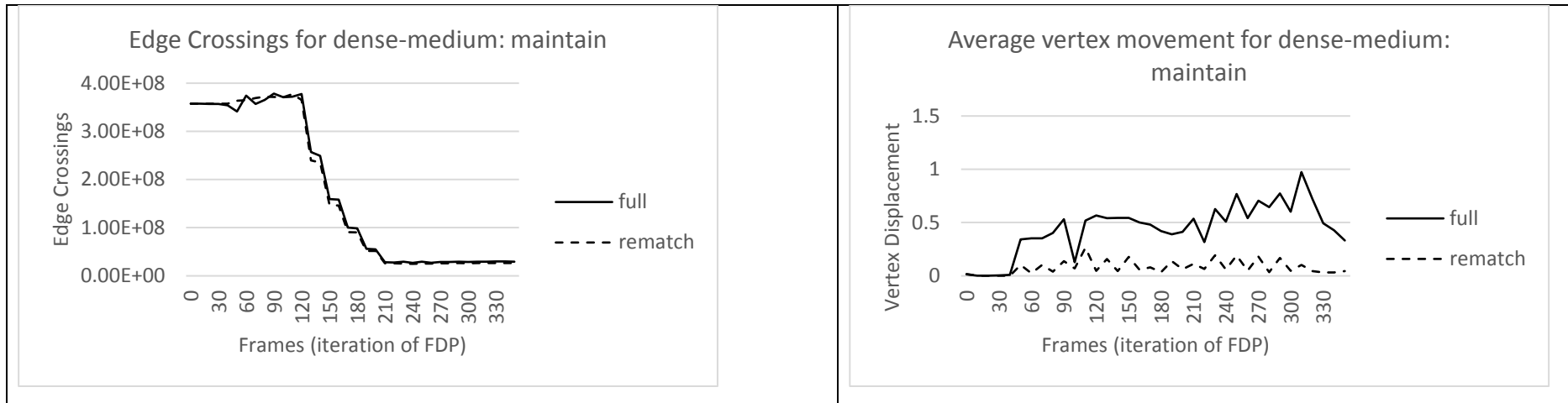


Edge Crossings for dense-medium: growth



Average vertex movement for dense-medium: growth

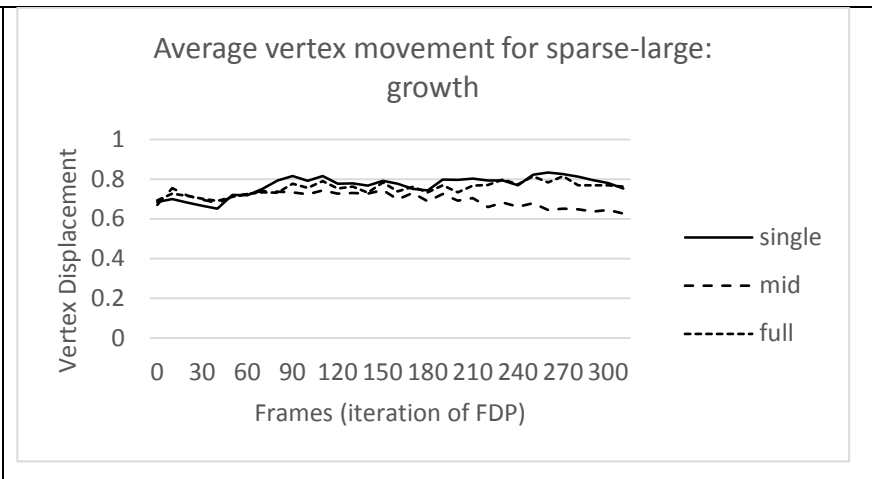
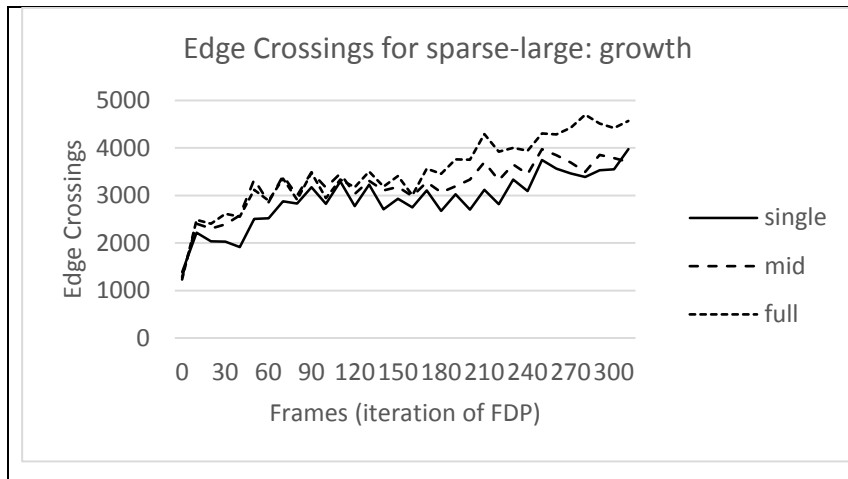
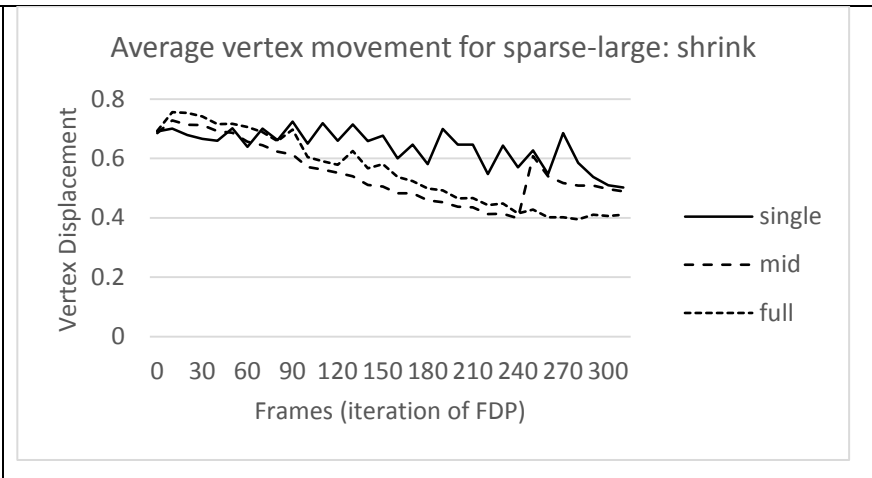
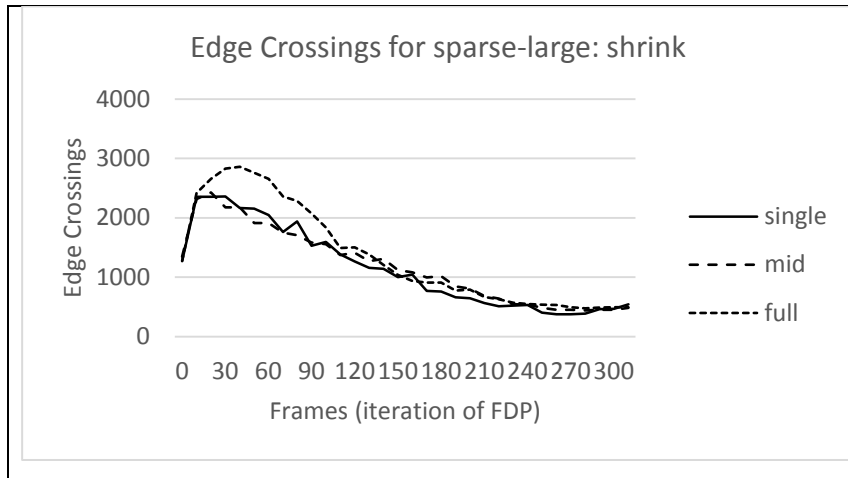


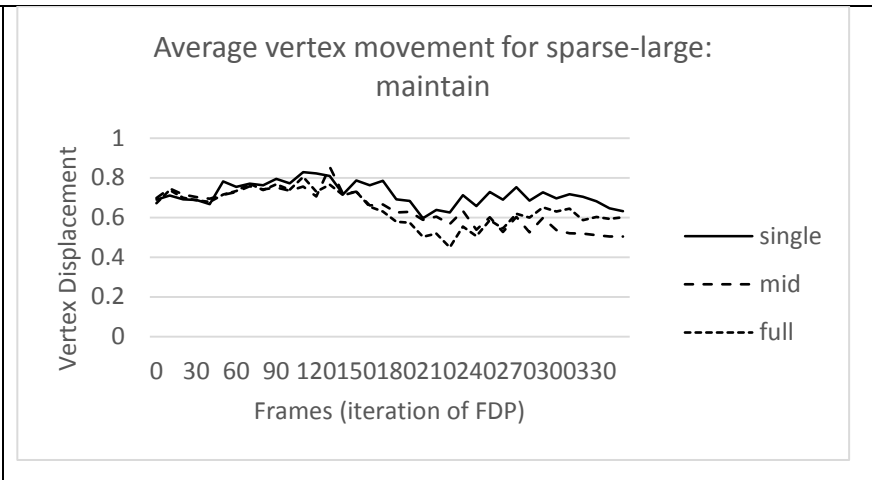
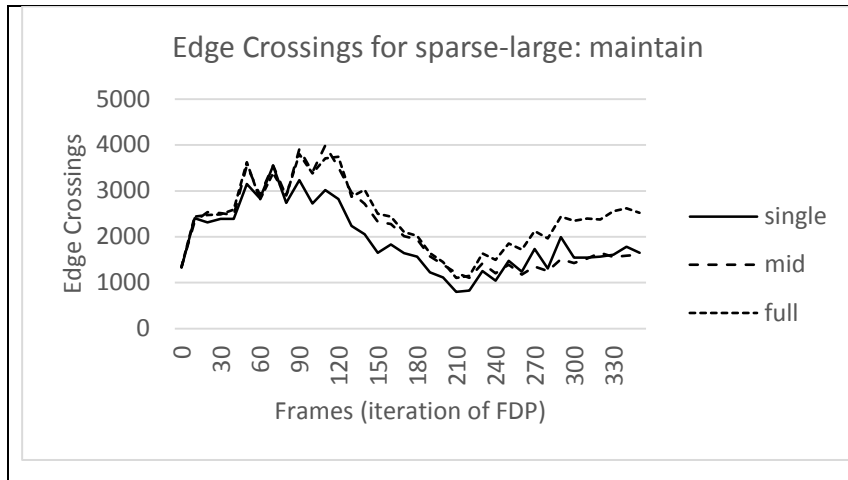


10.14.3 Large Graphs

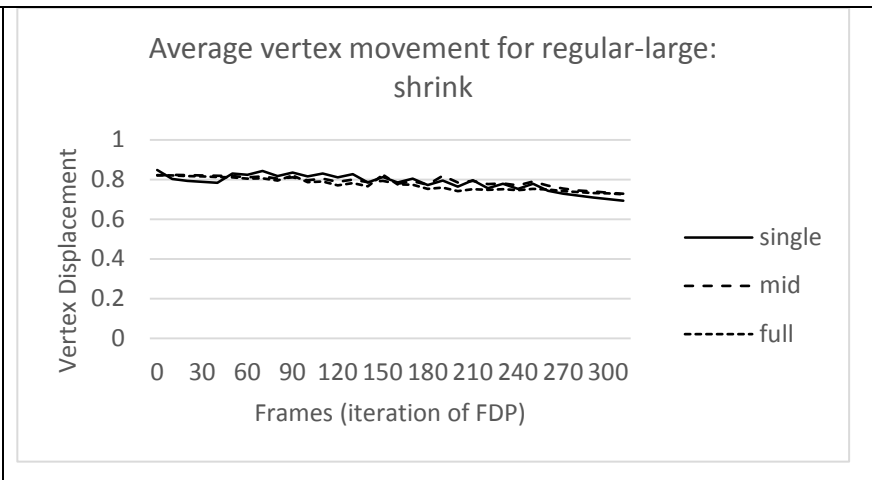
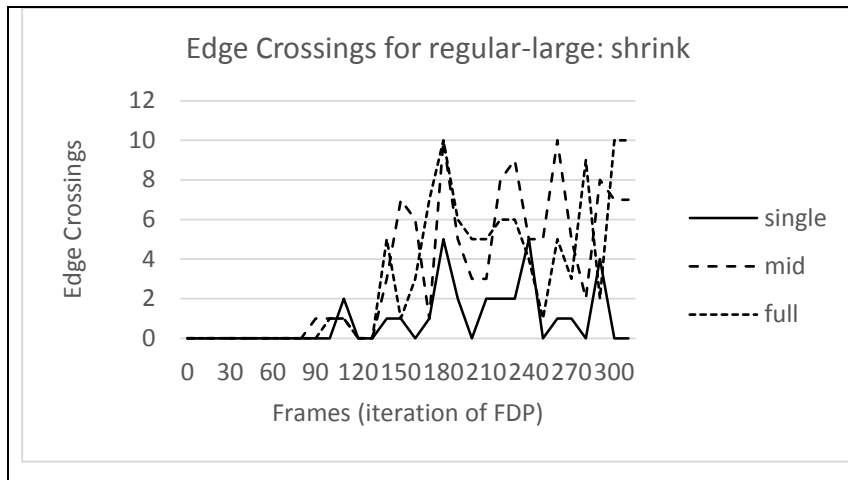
10.14.3.1 Single, Middle and Full Update Comparison

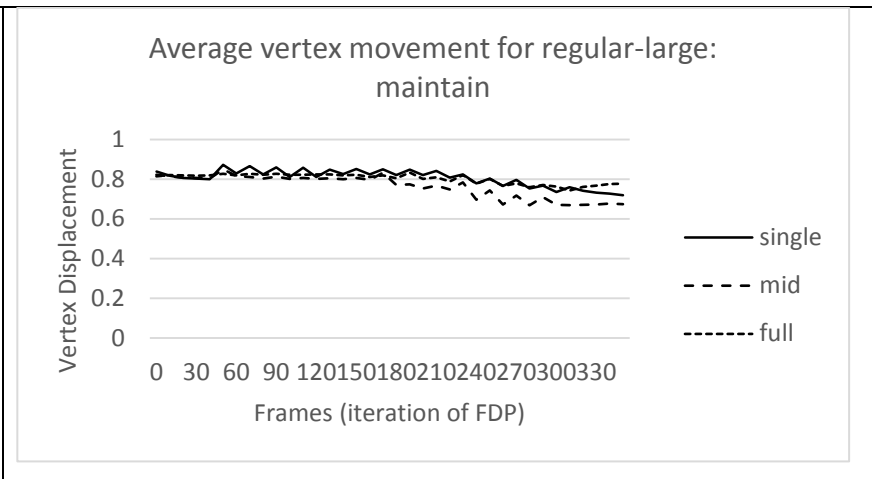
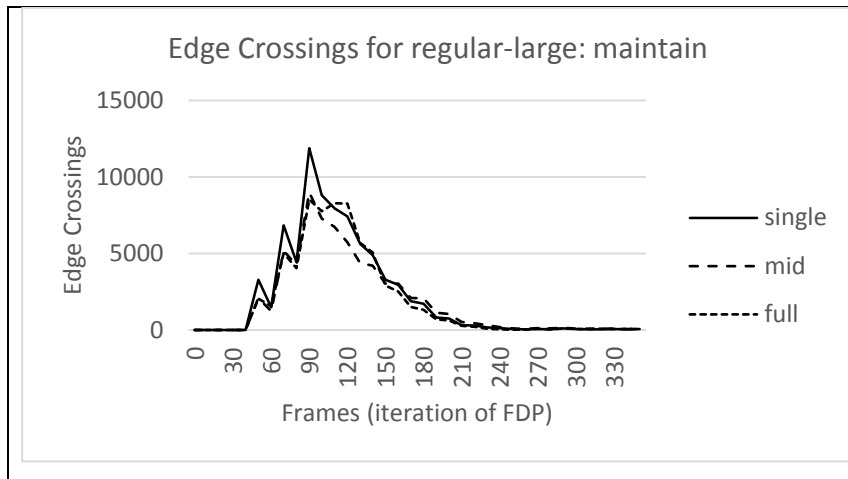
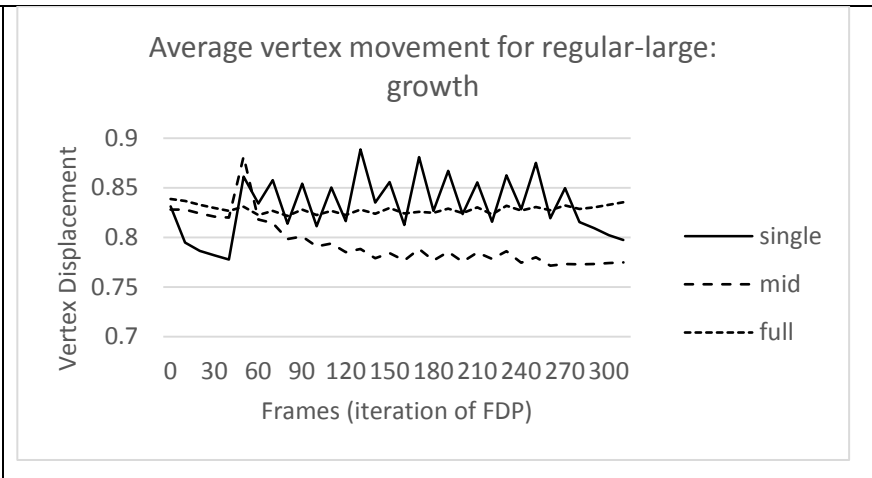
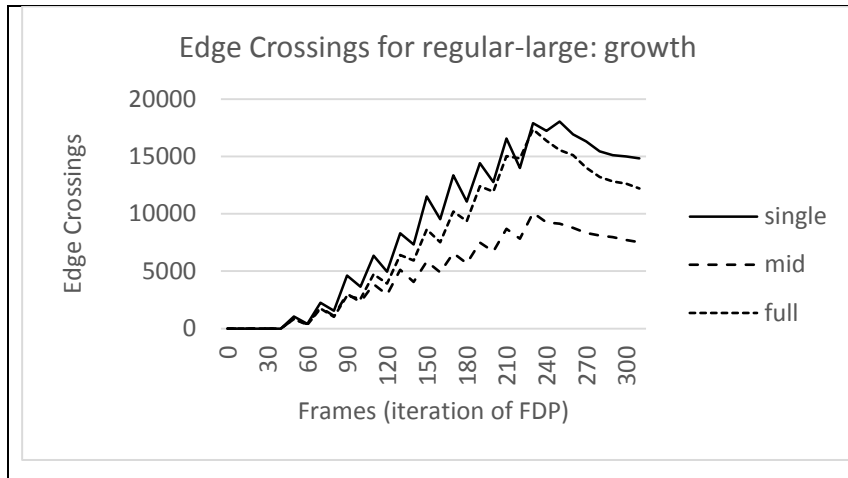
Sparse





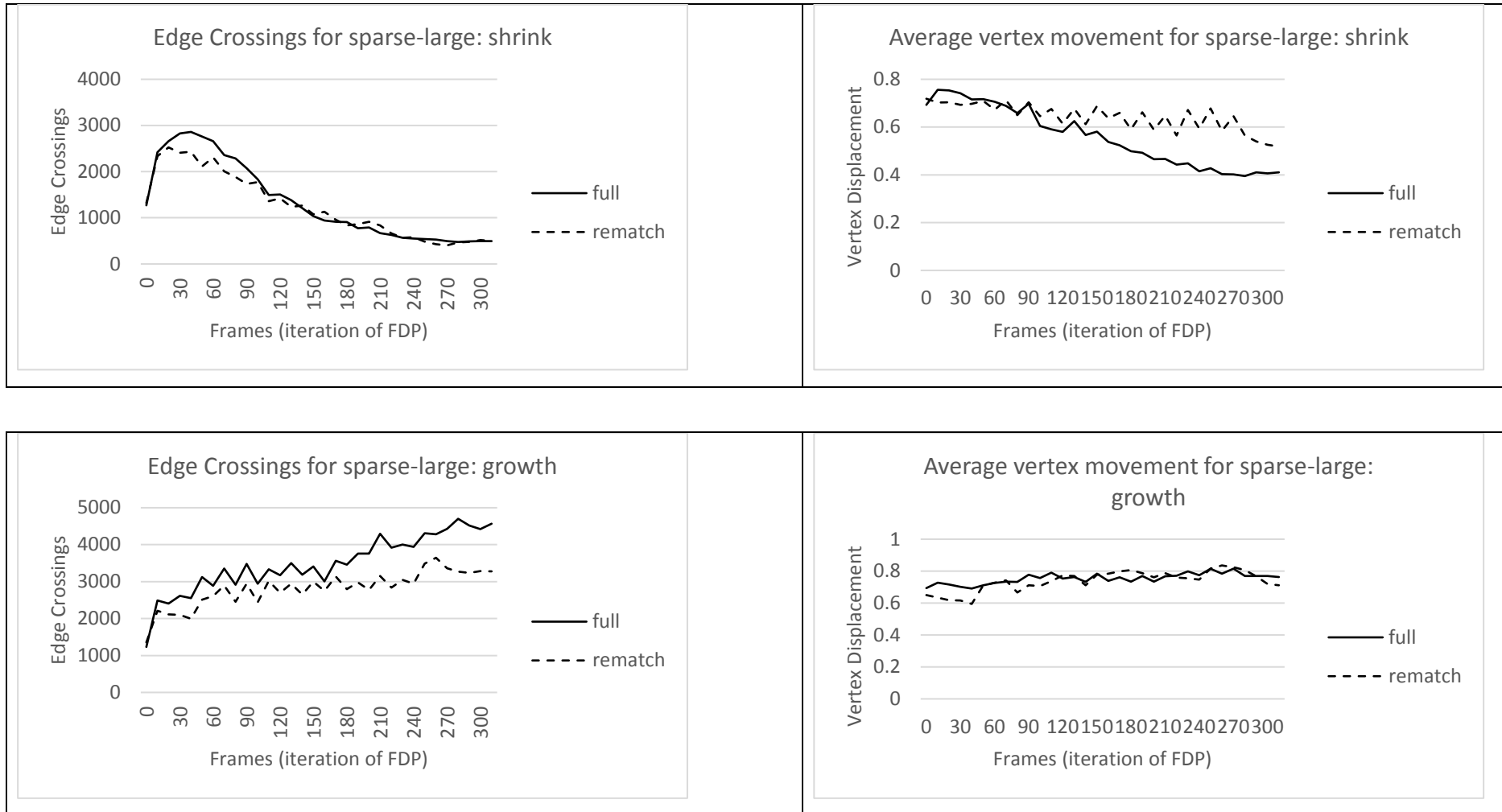
Regular

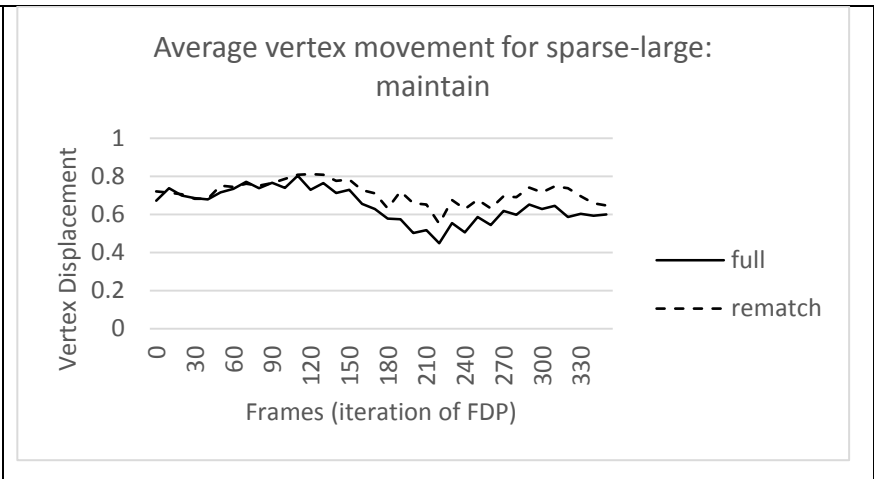
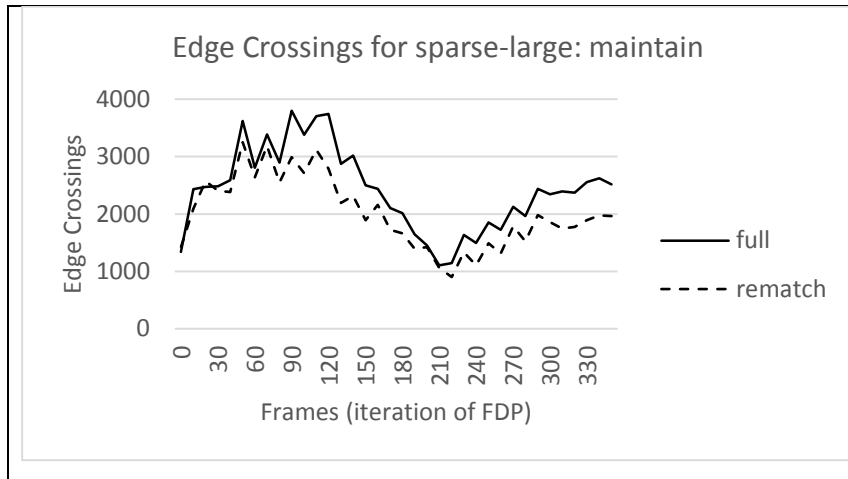




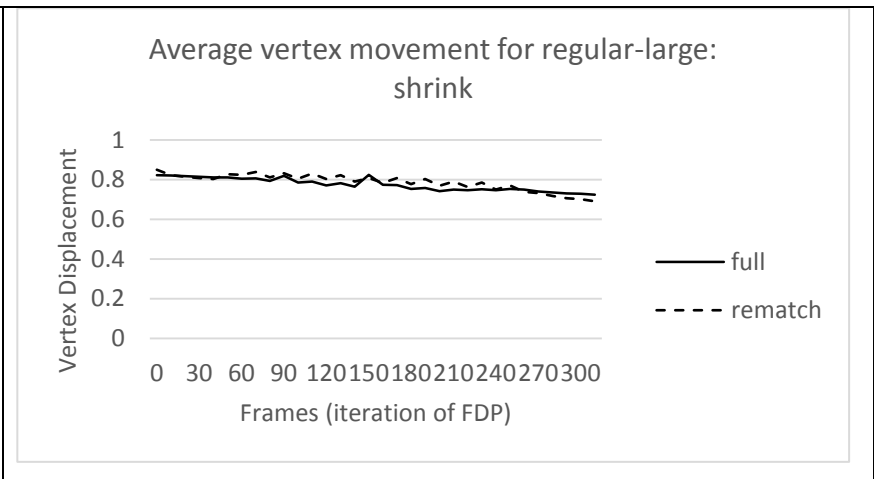
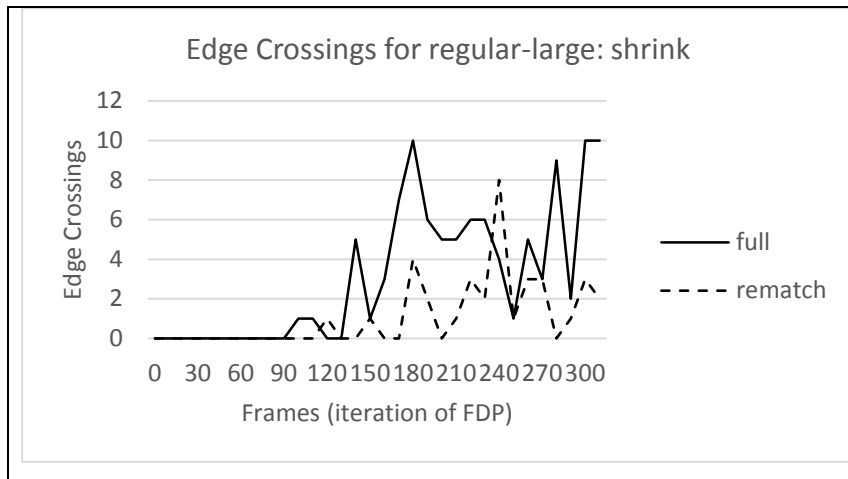
10.14.3.2 Full and Rematch Update Comparison

Sparse

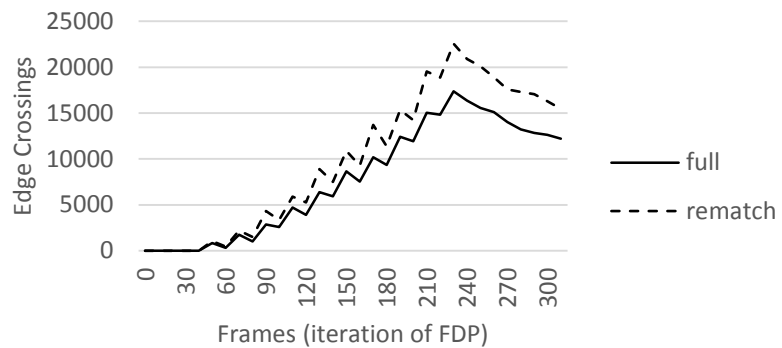




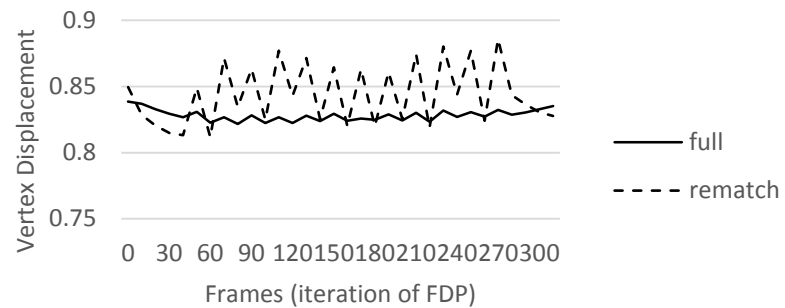
Regular



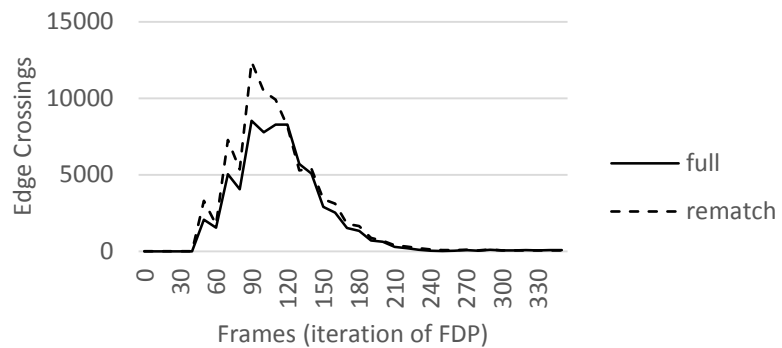
Edge Crossings for regular-large: growth



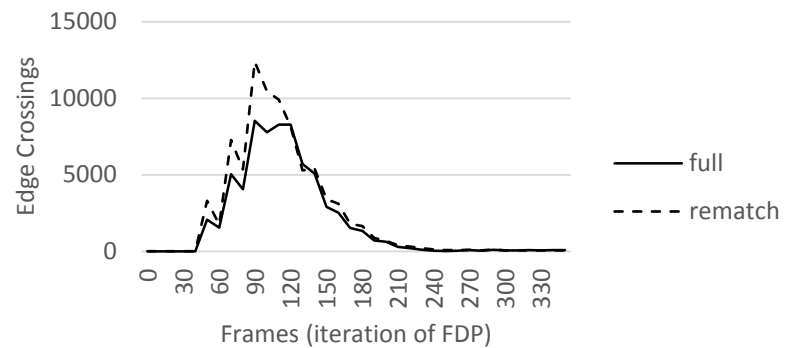
Average vertex movement for regular-large: growth



Edge Crossings for regular-large: maintain

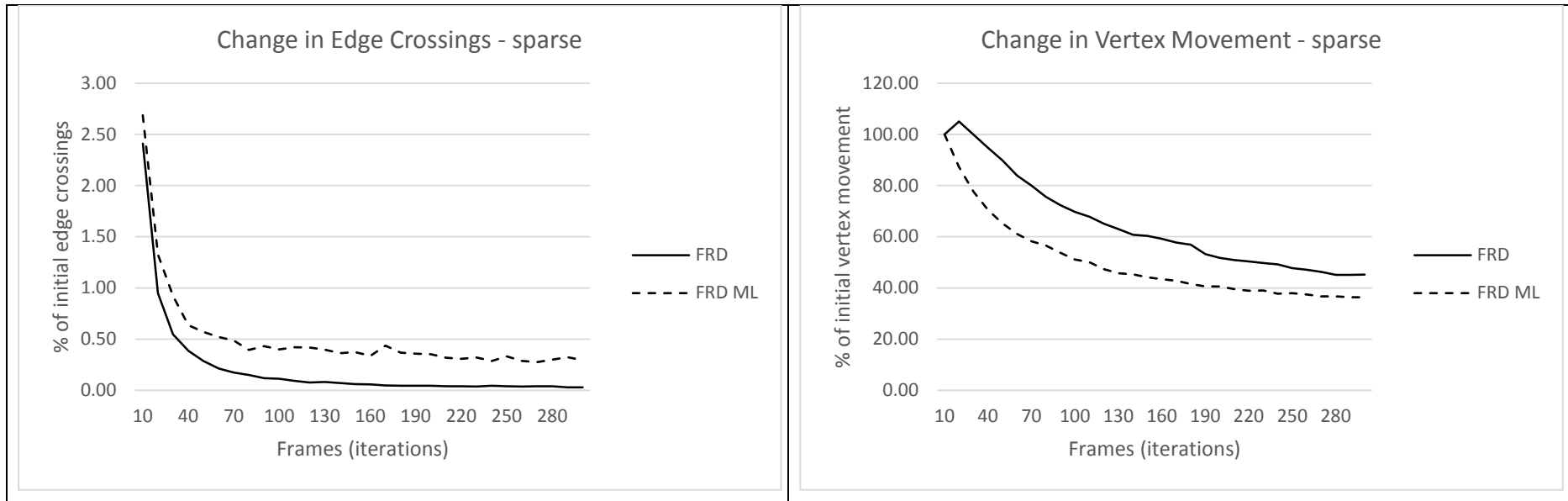


Edge Crossings for regular-large: maintain

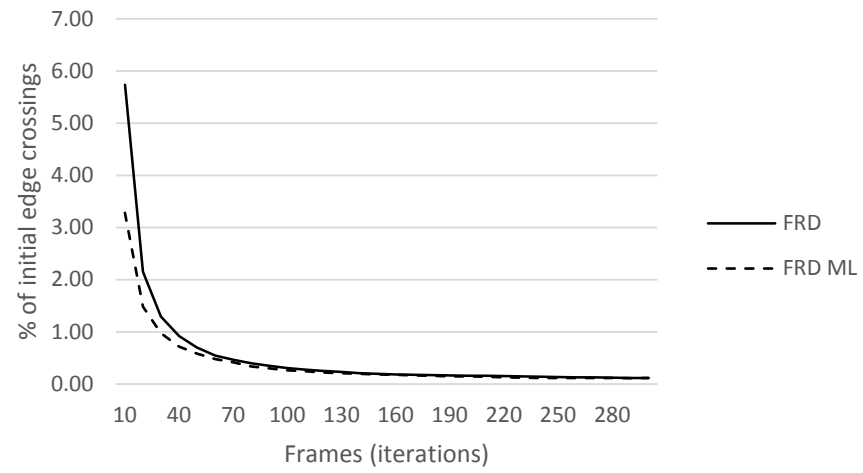


10.15 Comparison of Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder

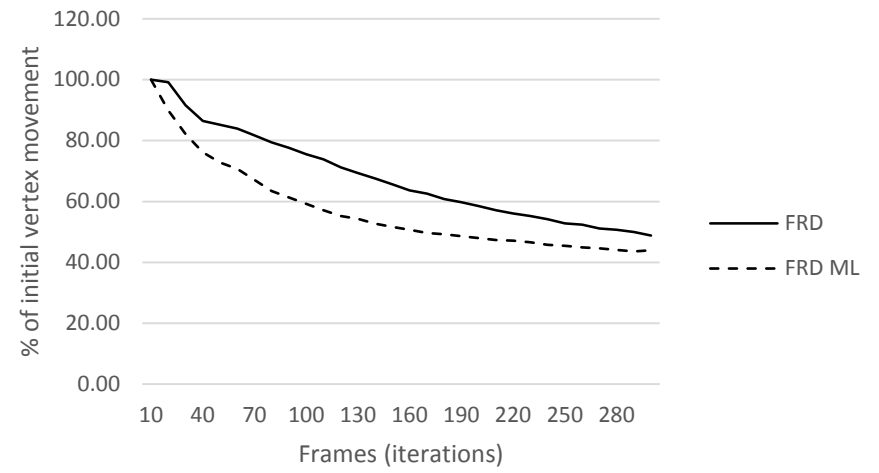
10.15.1 Comparison of Metrics for Graph Stability for *FRD* and *FRD ML* algorithms

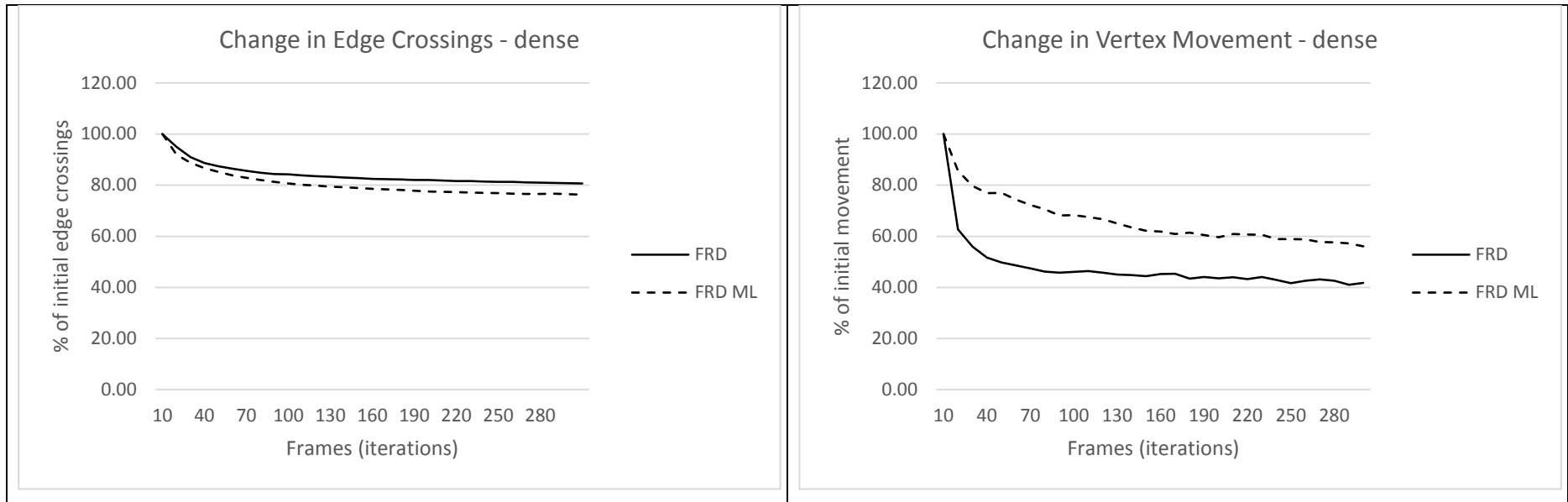


Change in Edge Crossings - regular



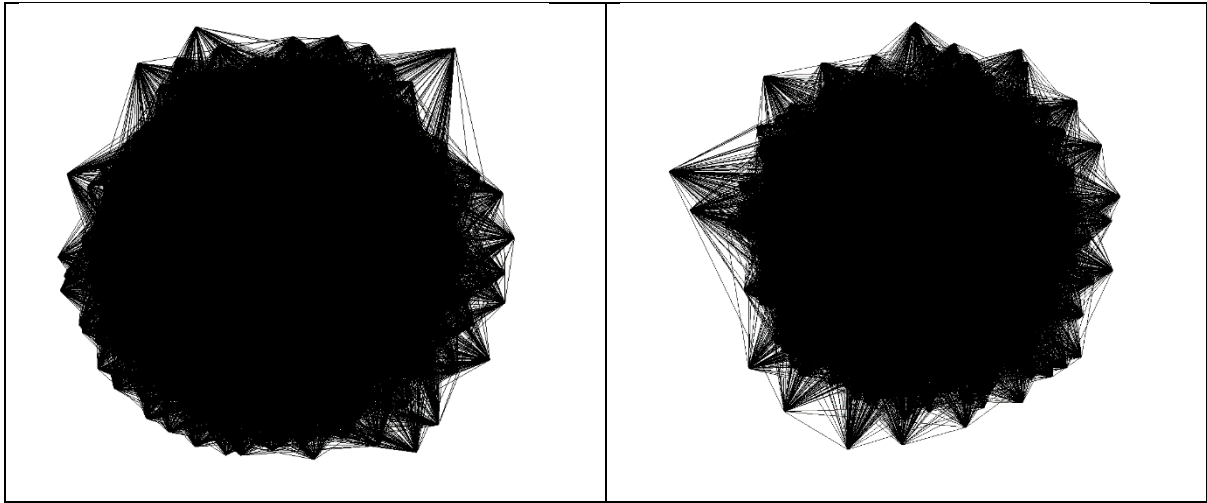
Change in Vertex Movement - regular



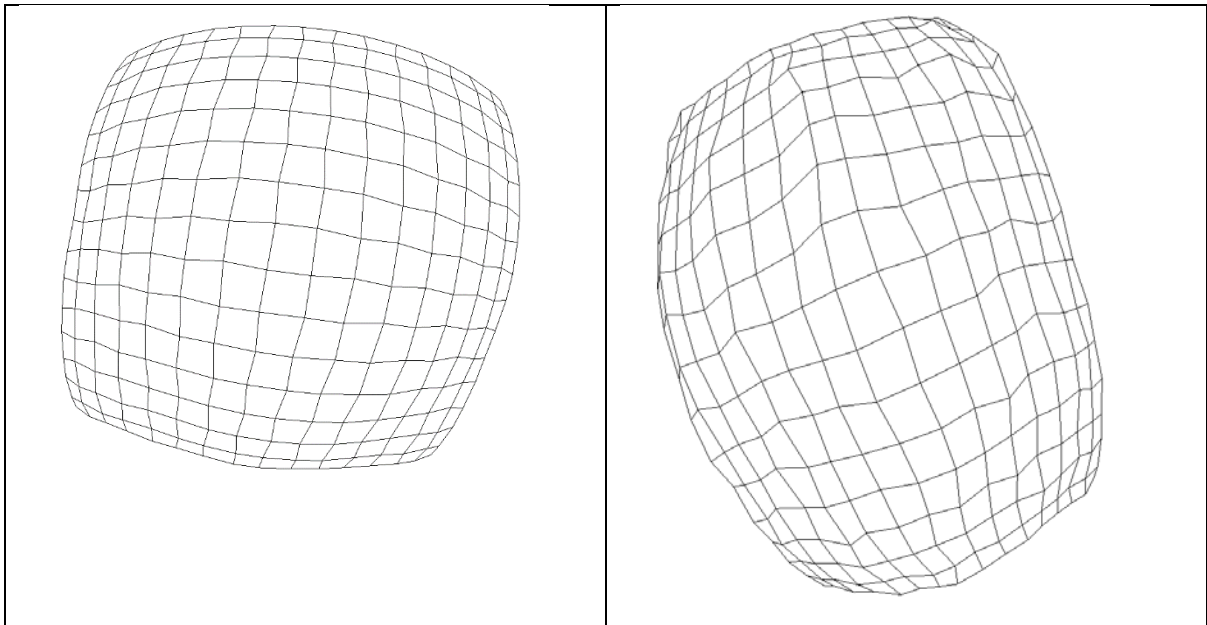


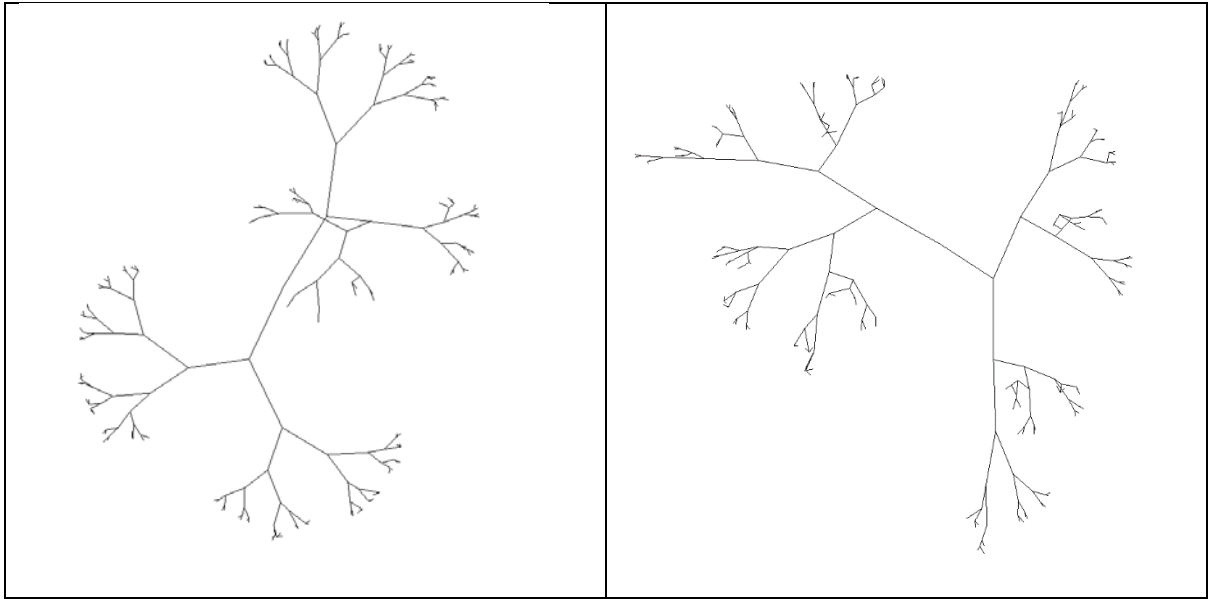
10.15.2 Comparison of *FRD* and *FRD-ML* layouts (Subjective Analysis)

Dense-medium

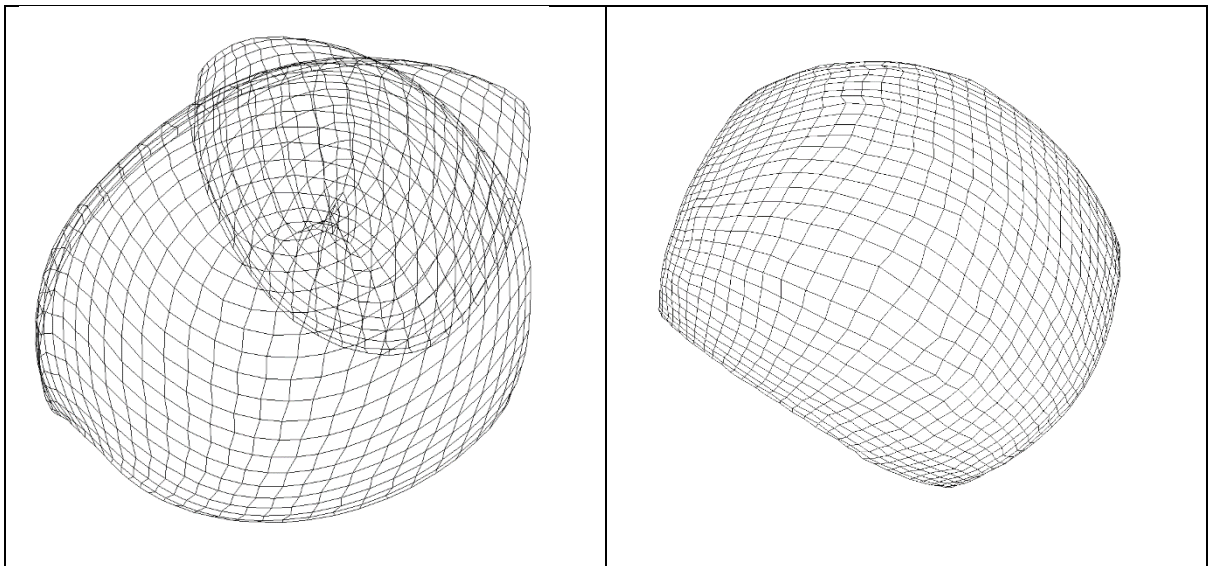


Small graphs show poor layout – regular and sparse (dense shows little change)

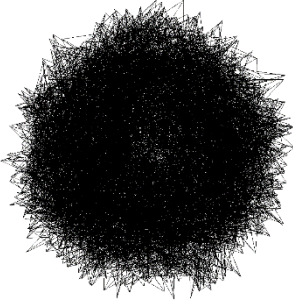
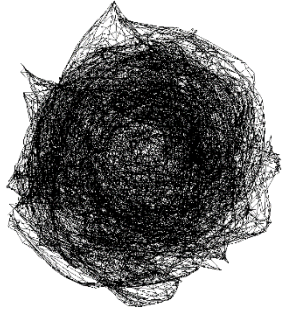
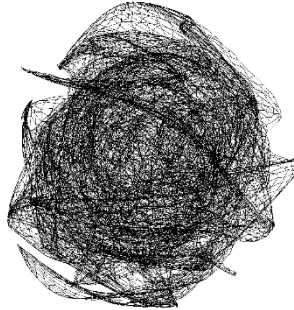
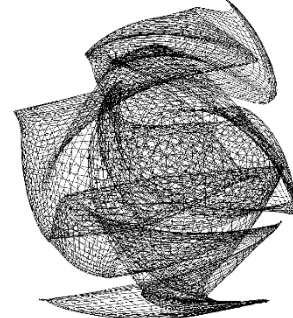
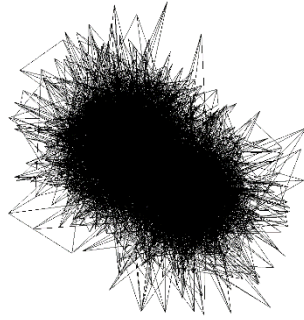
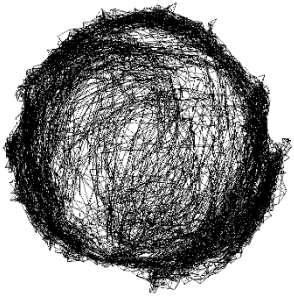
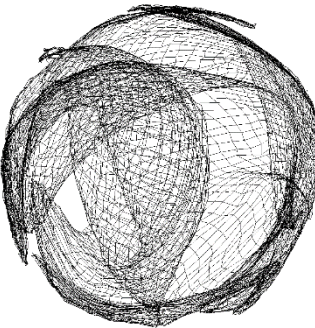
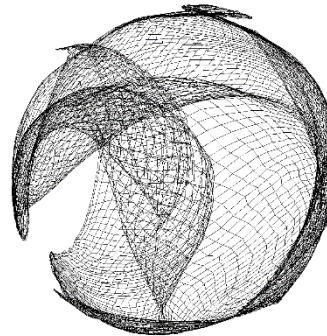
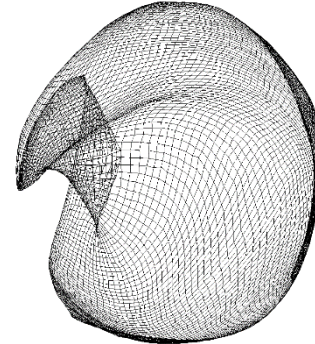


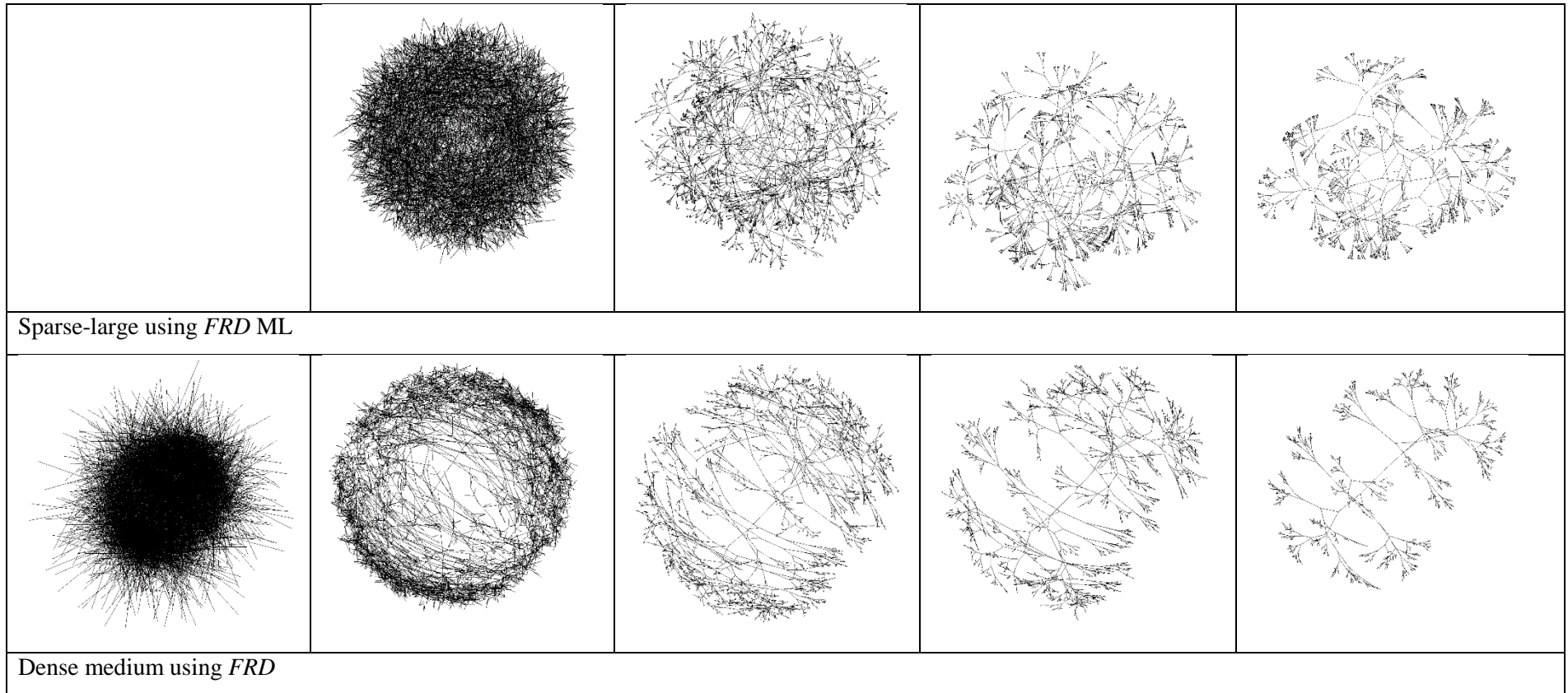


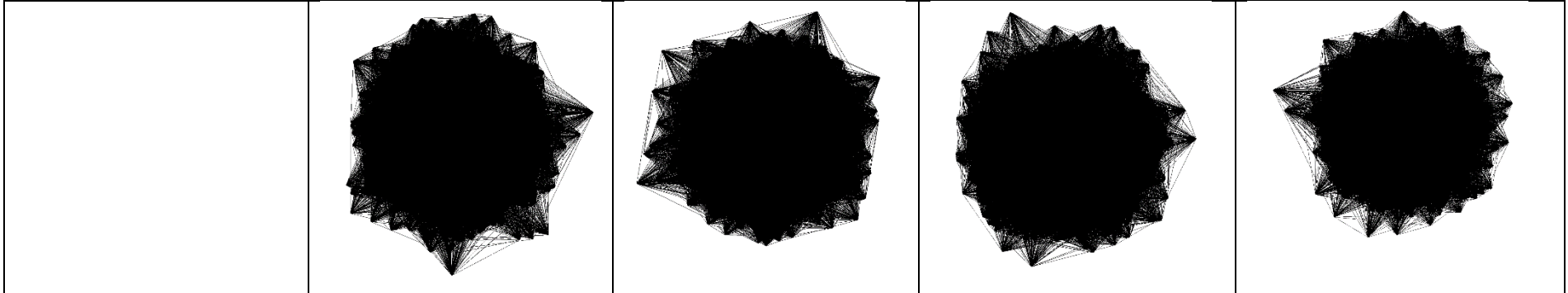
Regular suggests some improvement to medium sized graphs



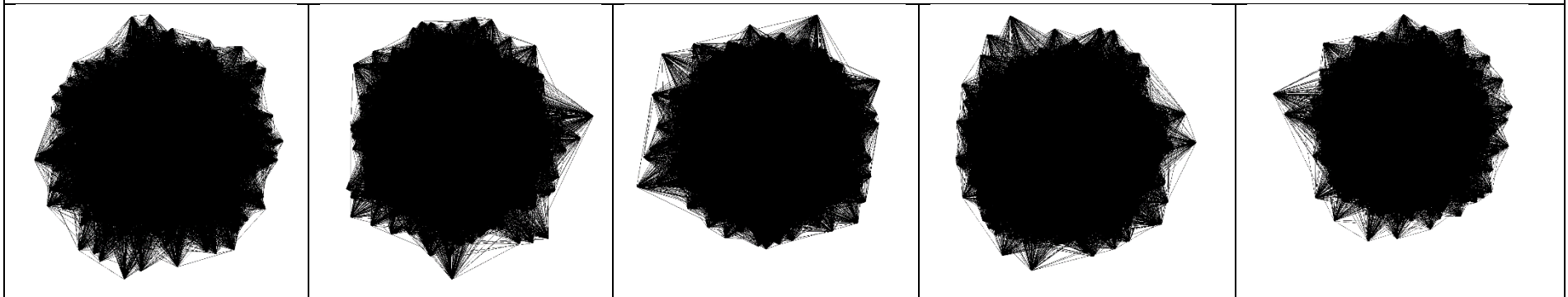
Layout Generation Overview

Frame 0	Frame 10	Frame 50	Frame 100	Frame 300
Regular-large using <i>FRD</i>				
				
Regular-large using <i>FRD ML</i>				
				
Sparse-large using <i>FRD</i>				

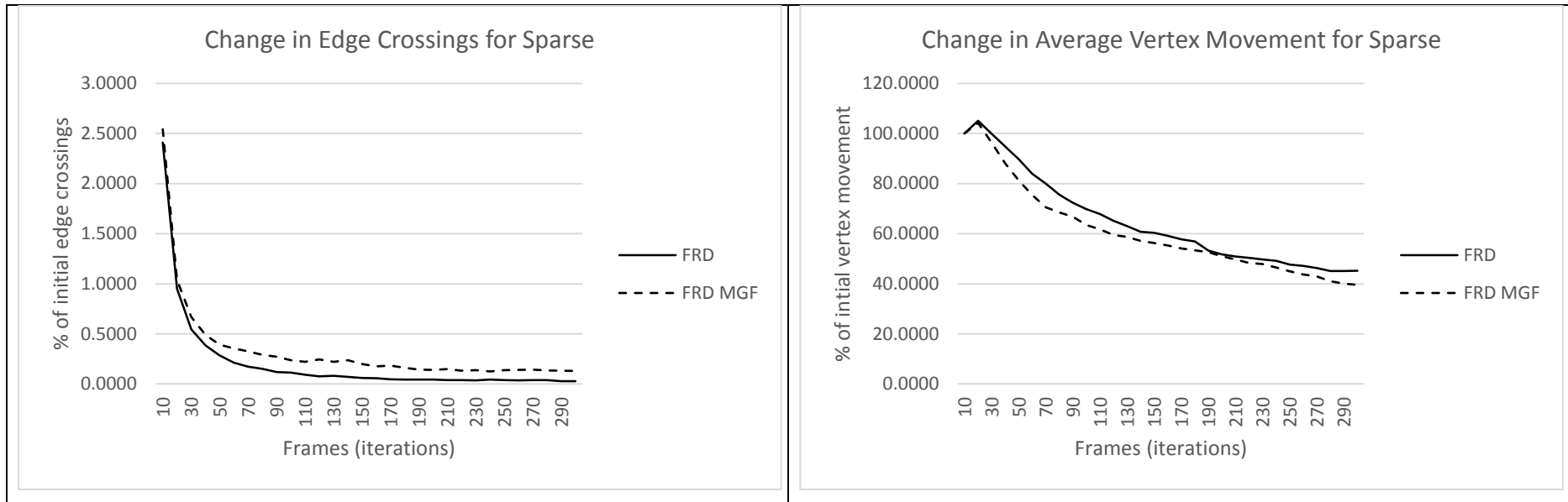


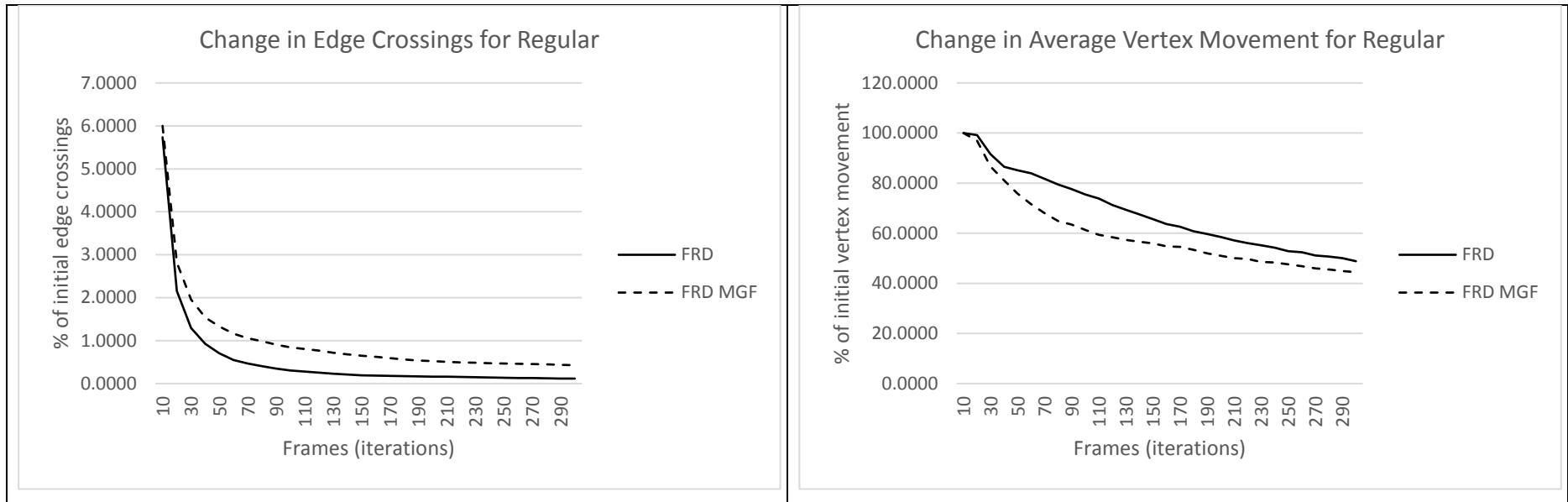


Dense-medium using FRD ML

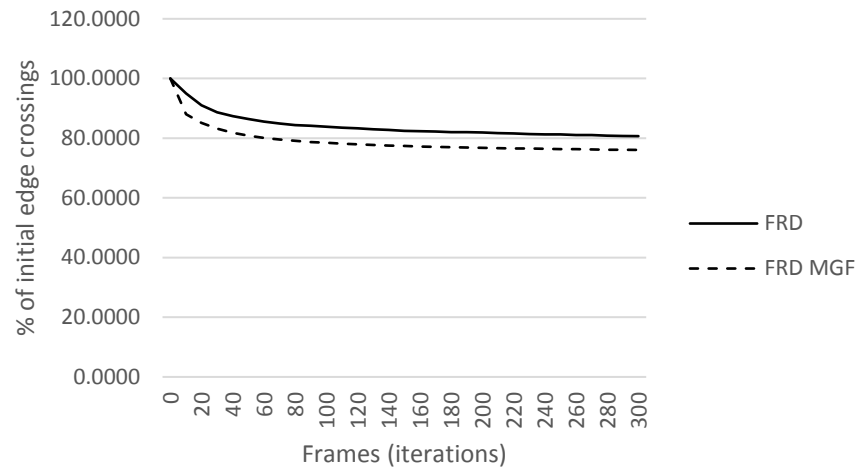


10.16 Comparison of Graph Stability for Dynamic Spring Embedder and Dynamic Spring Embedder with Multilevel Global Forces – Metrics for Graph Stability

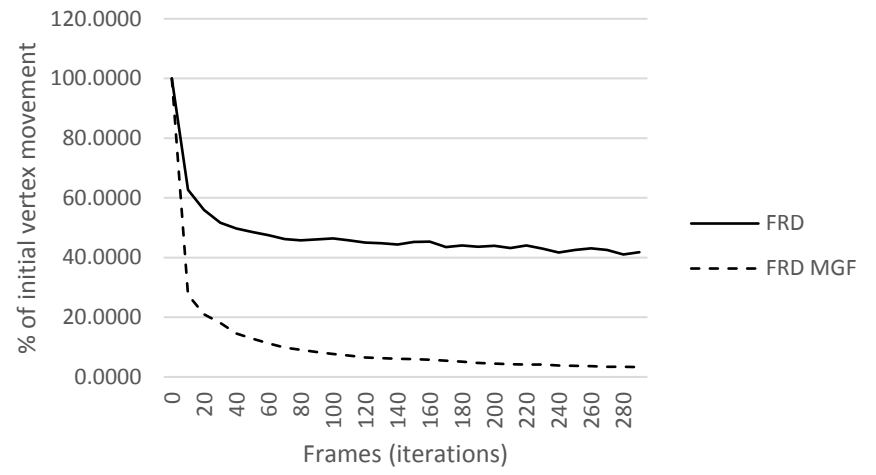




Change in Edge Crossings for Dense



Change in Average Vertex Movement for Dense



10.17 Animation and typical Frame rates for video mediums

For context, a typical computer monitor has a refresh rate of 60Hz (16.7ms per refresh), analogue TV (PAL) runs at 24fps (41.67ms per frame), and a hand drawn animation may run at 24fps, 12fps (83.33ms) or 6fps (166.67ms). Other faster methods are being introduced, for example, 1080p HD video is described as 60fps (dependent on the codecs used) with others offering up to 300fps.

10.18 Dynamic Matching: Impact of Graph Type on Multilevel Updates

It is mentioned above that the shrink operations provide erratic changes to edge crossings with no update method showing obvious improvement. Similar behaviour is observed between graph types, suggesting that removal of edges has the same impact on all graphs, independent of update method or graph type.

In contrast, differences can be seen through different update methods for different graph types when using the growth operations sets. Figure 10.47 and Figure 10.48 depict the change in edge crossings for sparse and regular type graphs as growth operations are applied. The differences are quite clear, the sparse structure showing a more erratic behaviour with single/mid/high level updates performing similarly and rematch providing a somewhat lesser change per frame during the operations phase (50-250).

The regular structured graph shows a much more obvious improvement between update methods (closer resembling the general behaviour of growth operations shown above), whereby single provides highest change in edge crossings, and high update provides least change.

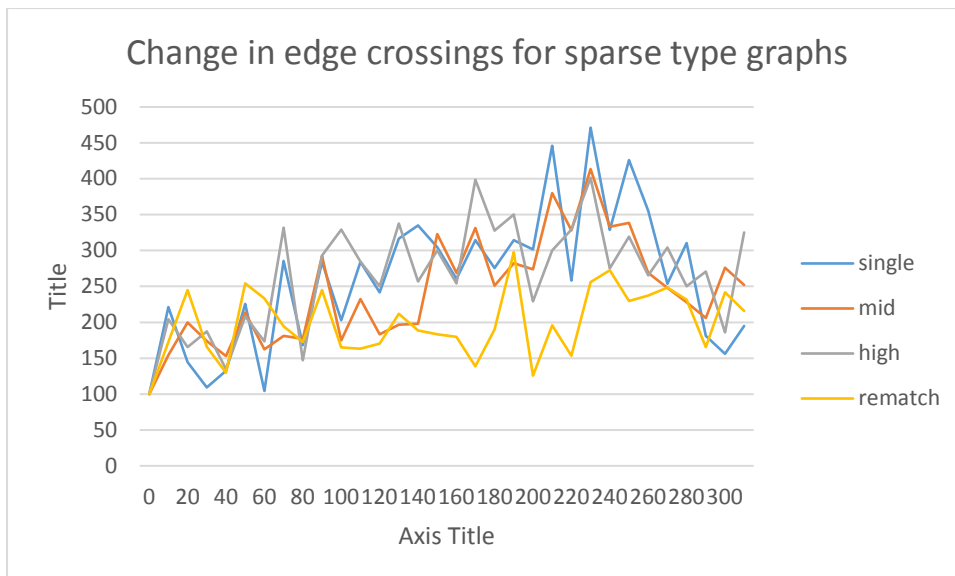


Figure 10.47 Change in edge crossings for sparse type graphs as growth operations are applied and reverberated through the multilevel scheme using the four update methods

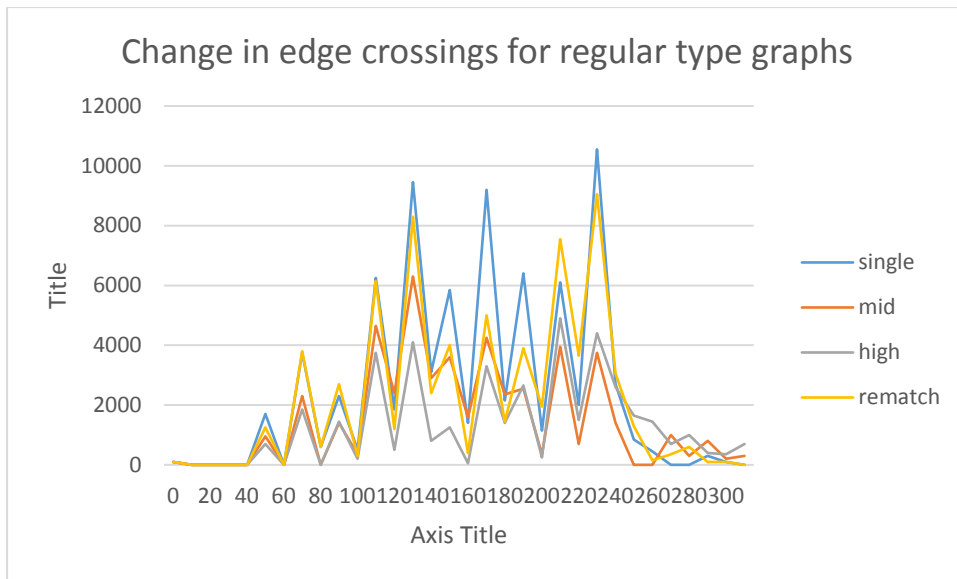


Figure 10.48 Change in edge crossings for regular type graphs as growth operations are applied and reverberated through the multilevel scheme using the four update methods

The difference indicates that sparse type graphs have less benefit from updating the multilevel scheme, whereas regular structured graphs gain a reduced change in edge crossings when incorporating amendments into the multilevel scheme.

Dense has the most persistent graph stability, with shrink and maintain operations being little affected by the update methods. As with the regular type graphs, growth shows a more notable difference between methods, with rematch and full update resulting in higher number of edge crossings in the layout, however, the difference is only 3% of the edge crossings in the initial layout.

Throughout all graph types, movement is little effected by the update methods with rematch update generally providing higher movement. Dense graphs however, show a larger amount of movement than that observed in regular and sparse graphs, but is independent of the update methods (as indicated in Figure 10.49) movement spikes to 200% however, the update methods all provide such movement – indicating that movement is not impacted by the update methods.

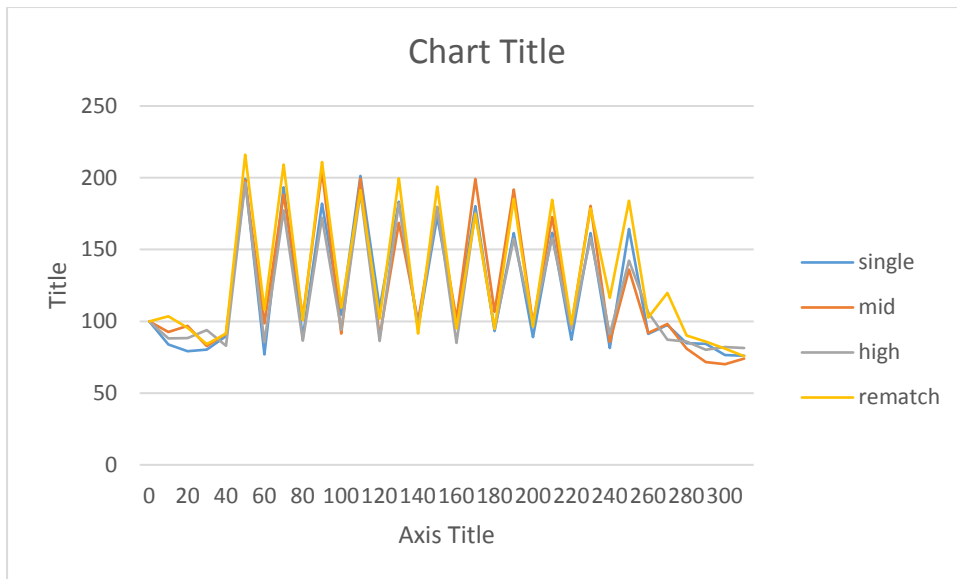


Figure 10.49 Average vertex movement measured in dense-type graphs during application of growth operations, showing that for each of the four update methods, the movement exhibits the same peaks

10.19 Isomorphic Drawings

The aim of many drawing algorithms is to provide regular high quality results (Kamada and Kawai (1989), Fruchterman and Reingold (1994)), however, force directed placement often results in differences in layout as a result of minima in the graph and differences in initial vertex placement. Experimentation is therefore performed to identify the similarity of layouts for isomorphic graphs, answering the following question:

Does the use of Multilevel Global Forces in conjunction with a Spring Embedder provide regular layouts for isomorphic graphs?

To test this, each of the static test graphs is reproduced ten times with vertices ordered differently, resulting in 10 different graphs with the same structure. Layouts are provided for the graphs (repeated 5 times) with the average number of edge crossings and an edge range recorded. The result is {5 x 10} layouts per test graph.

The average number of edge crossings and range in edge crossings are compared to determine similarity. Layouts which exhibit edge crossings within 1.5 standard deviation of the average number

of edge crossings are classified as similar, with any outside requiring further subjective investigation to identify the cause of differences in layout.

10.19.1 Numerical Results

10.19.1.1 Comparison of *MGF* to Octree

Comparison of Multilevel Global Force and Octree data structures provides some initial analysis of layout regularity suggesting that *MGF* reproduces layouts with smaller range in edge crossings between them (regular layouts). As indicated in **Table 10.13**, the results show that the range in edge crossings for *MGF* is much smaller than those generated by Octree approximation, a result of the unchanging structure of the *MGF* tree in comparison to the volatility of vertex movement relative to one another (structural versus spatial approximation).

Graph	Range in Edge Crossings		
	<i>MGF</i>	OT	Reduction
4elt	290	14161	97.95%
3025	32	2052	98.44%
add32	78	3617	97.94%
data	31	2933	98.94%
dime20	4625	35211	96.86%
finan512	1094	67566	98.38%
mesh100	4453	15250	70.80%
sierpinski10	327	3498	90.65%
			92.48%

Table 10.13. Comparison of the range of edge crossings for layouts generated using Multilevel Global Force (*MGF*) and Octree (OT) approximation, with context of the difference between them (92.48% reduced range).

10.19.2 Isomorphic Layouts using *MGF*

Analysis of the layouts generated for the randomized graphs provide the results in **Table 10.14**. The results indicate that on average there is a large range in the number of edge crossings across the randomized graphs (up to 21% below and 49% above average, a range of 70% difference), suggesting poor isomorphic drawing. The cause is due to folds in the layouts for 3025, which would normally be drawn as planar. By omitting the results for 3025 the range drops dramatically, with all layouts exhibiting edge crossings within 10% below average and 12% above average (a range of 22% difference).

Graphs	Edge Crossings				
	average	min	max	avg/min	avg/max
3025	237.2	1.2	961.6	0.5%	405.4%
data	38747.3	36377.4	39974.8	93.9%	103.2%
add32	35950.1	32684.0	41835.0	90.9%	116.4%

4elt	34202.0	28056.8	41577.2	82%	121.6%
sierpinski10	15009.5	14511.2	15566.8	96.7%	103.7%
finan512	5632952.3	5435551.4	5897981.2	96.5%	104.7%
dime20	81096.3	65180.6	106737.0	80.4%	131.6%
mesh100	1017924.9	949211.2	1093848.8	93.2%	107.5%
				79.3%	149.26%

Table 10.14. Comparison of edge crossings and range of edge crossings for layout provided by Multilevel Global Force (*MGF*), providing the range in regularity of layouts provided using the method.

With the exception of 3025, the majority of graphs are provided with similar layouts, suggesting that layouts for isomorphic drawings will be provided with up to 12% difference of an average layout. In addition, comparison to Octree suggests *MGF* provides noticeably more regular results.

10.19.3 Subjective Analysis

Due to the number of layouts generated, differences are compared for only some of the layouts generated. An example of the layout for 3025 shows two extent of folds resulting in higher and low number of edge crossings. The cause of such a fold is typically the result of a twist in the coarser layouts provided by the multilevel scheme, of which refinement is unable to rectify due to high minima in the finer graphs.

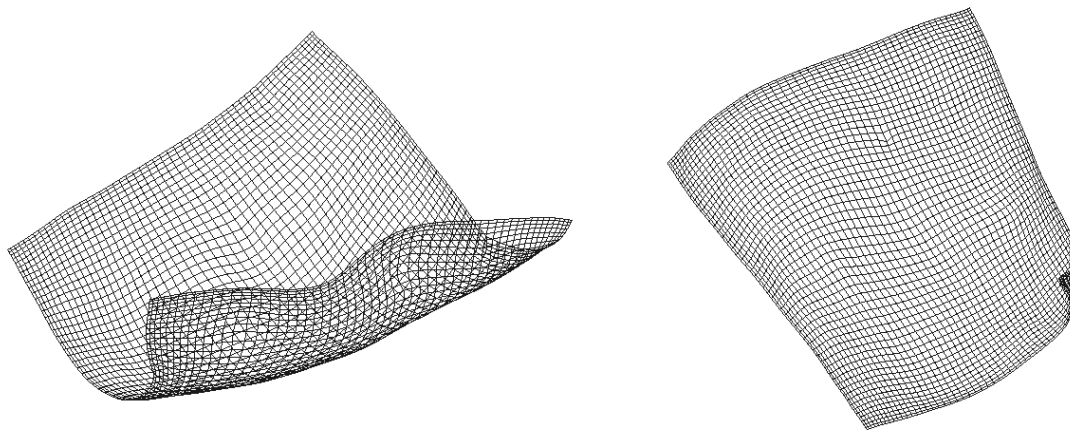


Figure 10.50. Examples layouts provided for graph 3025 using Multilevel Global Force (*MGF*), exhibiting a large fold in the layout (left) resulting in a large increase in edge crossings, and a minor fold with much fewer edge crossings (right)

The largest range in edge crossings is exhibited by layouts for dime20, and therefore this is the first to be analysed. Analysis of the randomised graphs show layouts with least edge crossings were achieved for dime20.randomised0, whereas highest edge crossings were exhibited by dime20.randomised7. Figure 10.51 provides layouts for the two graphs using the same force directed placement method, showing little observable difference (from the authors' viewpoint). The rise in edge crossings is likely

to come from compression seen on the central and left branches and the overlap of smaller structures (see bottom centre, whereby a small triangular structure overlaps layout in the left image, but not in the right).

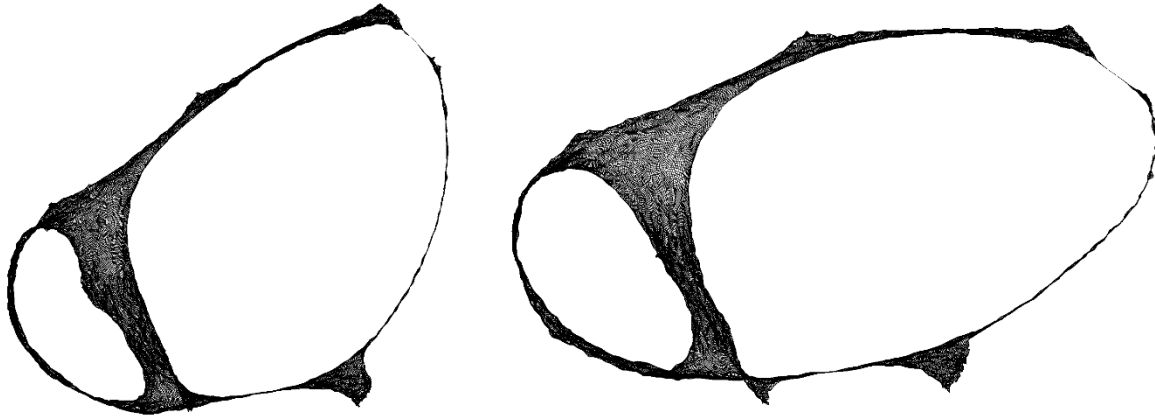


Figure 10.51. Comparison of layouts for randomised variations of dime20 (randomised0 and randomised6), showing little difference between the graphs.

Multiple layouts were regenerated for the two graphs, with little difference between them. Occasionally some part of the graph overlaps on the rest of the graph, however this is due to randomness within the automatic layout process (typically through initial positioning or matching of the multilevel scheme), and is believed to be the cause of the differences in numerical analysis.

Each of the tested graphs are checked to determine if any differences occur from the re-shuffling of the graphs, with the common conclusion that the cause of any changes in layout is due to randomness within the layout process and not due to the structure of the graph and resulting *MGF* scheme (if variations in layout were caused from structure, the layouts would be regularly different).

Figure 10.52 shows a large fold in a layout for 4elt (left), showing an alternate layout to that expected for the graph (right). As a result, the graph looks almost entirely different due to this change, and is caused by a “branch” of vertices overlapping the main body of the graph and expanding as a result. After multiple repeats, the abnormal layout could not be replicated.

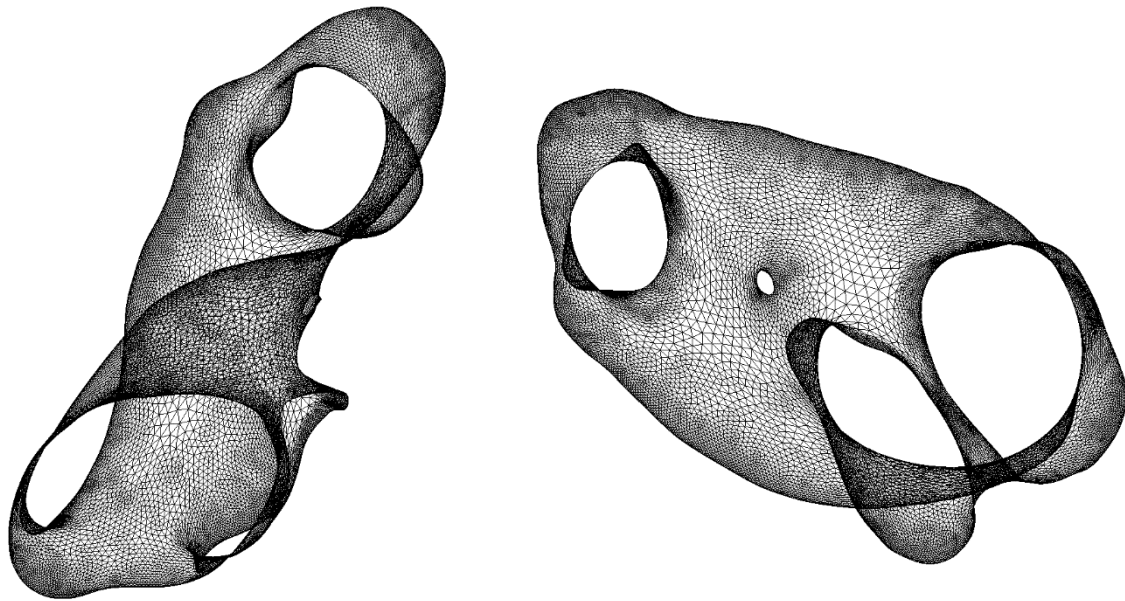


Figure 10.52. Layouts for 4elt exhibiting typical layout and an overlapping layout causing a large increase in edge crossings.

Figure 10.53 shows a common irregularity seen in the layouts generated for sierpinski10 (left), whereby parts of the layout fold other themselves. The main structure can still be identified, however the change greatly effects the readability of the layout due to the expectation of the layout on the right.

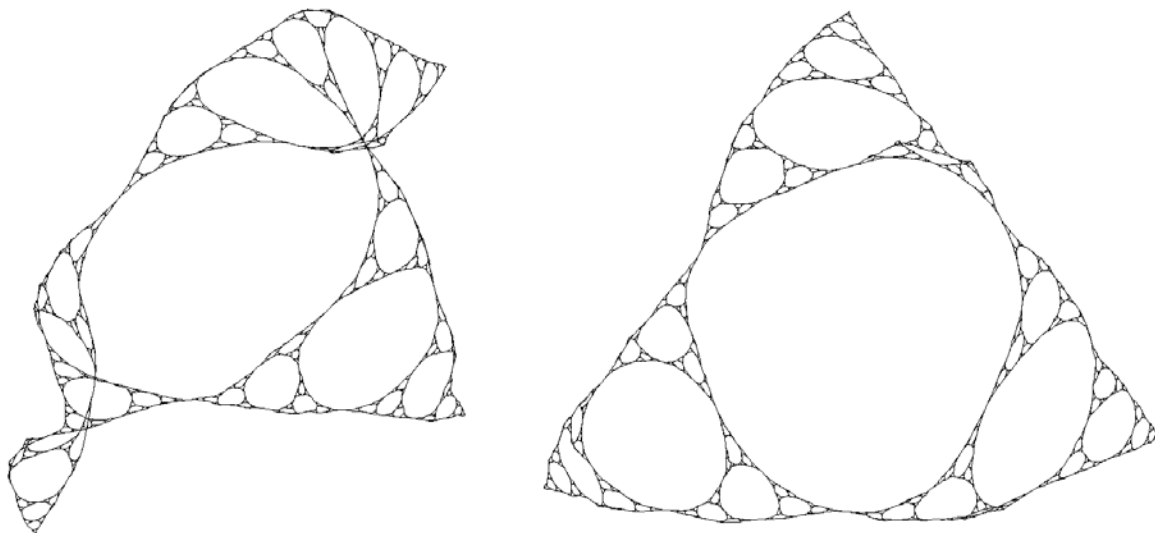


Figure 10.53. Layouts for Sierpinski10 showing a typical layout and a layout exhibiting a large fold, resulting in a large increase in edge crossings.

The rate at which these irregularities in layout occur is unpredictable and differs for different graph structures (for example, it was noticed sierpinski10 was more likely to be given layout with a fold than

3025, despite the collected results). The similarity of layouts is difficult to measure subjectively, however, on average layouts for isomorphic graphs look the same from the author's point of view.

10.20 Multimatching

Multimatching aims to optimise the approximation of graph structure during multilevel generation, altering the Multilevel Global Forces. Experimentation looks to investigate the effects on approximation and the resulting layout quality. Specifically:

Can Multimatching be used to provide improved approximation and representation of graph structure in multilevel and Multilevel Global Force schemes?

A description of the Multimatching algorithm and differences to standard Multilevel Global Force generation is described in Implementation 10.22 . Experimentation is achieved by testing the matching number, m , with incremental values between 2 and 8 for static test graphs, whereby 2 represents standard edge contraction. Each test is repeated 10 times, resulting in 60 $\{(8-2) * 10\}$ results for each graph, which are averaged and compared. Running time is also analysed and compared.

Differences in layout quality are expected as a result of changes to the value of m , with investigation identifying any correlation to structural attributes of the graph such as average and modal vertex degree. Changes in running time are also expected due to changes in the structure of the tree and the coarseness of the generated graphs of the multilevel scheme, but are lesser investigated due to the expected differences as a result of the complexity, $O(L \cdot |V| \cdot (m-1) \cdot (|G_i|-1))$ (as m increases, L will likely decrease due to coarser approximations of a graph being made).

10.20.1 Numerical Results

On average, the results suggest that as the value of m increases, so do the number of edge crossings and therefore the lower quality the layout is perceived to be, as shown in **Table 10.15**. The deterioration of layout may be attributed to coarser approximations of graph structure being generated in the multilevel scheme, resulting in refinement of local layout overwriting the interpolated layout. Despite the numerical analysis, some layouts are improved from the author's point of view, as discussed in Subjective Analysis.

Change in Edge Crossings								
m	3025	add32	data	4elt	sierpinski10	finan512	dime20	mesh100
2	0.0	16240.0	36617.5	25695.0	37189.0	5043279.0	168011.0	979875.5
3	5.6	26814.0	52163.6	26092.6	40144.4	6068587.4	221637.0	1176769.5
4	1.0	33959.6	42351.6	25255.2	40449.0	6035649.4	230021.3	1319544.5
5	12.0	38008.6	47216.2	25277.0	40824.8	5841396.8	229010.3	1302237.0
6	2.2	42877.8	46833.6	24564.8	40122.0	5930307.4	227756.7	1293845.5

7	6.2	50851.2	64430.6	25509.4	40643.8	5851011.6	224438.7	1294489.5
---	-----	---------	---------	---------	---------	-----------	----------	-----------

Table 10.15. Change in edge crossings for layouts of the test graphs for changes of multimatching number

One of the key aims of Multimatching is to change the structure of *MGF*, and is therefore expected to affect the Peripheral Effect due to the change in the number of directions in which approximation is made (children per vertex in the *MGF* tree). **Table 10.16** shows that for many of the graphs, the range in edge lengths increases, with only add32 and data providing decrease in the range of edge lengths, contrary to what is expected. Due to the results not matching the expected behavior, subjective analysis is provided to identify and describe the effect of Multimatching on layouts and the Peripheral Effect.

Range in Edge Length from m2								
m	3025	add32	data	4elt	sierpinski10	finan512	dime20	mesh100
2	0.4963	2.2534	0.8887	0.7073	0.3399	0.9266	0.5416	0.5329
3	0.4951	1.6352	0.8295	0.7084	0.3234	1.0629	0.5554	0.5650
4	0.5257	1.0013	0.6221	0.7434	0.3194	1.0748	0.5709	0.5767
5	0.5363	0.5895	0.3878	0.7581	0.3274	1.0773	0.6169	0.6058
6	0.5429	0.3816	0.2538	0.7572	0.3251	1.0993	0.5846	0.6060
7	0.5603	0.2729	0.1830	0.7553	0.3278	1.1079	0.6424	0.6150

Table 10.16. Range in edge length of layouts for changes to the Multimatching number

Running time, as expected, shows some decrease as the number of graphs in the multilevel scheme is reduced, however, use of Multimatching impacts complexity and therefore any reduction in the number of graphs is added to the number of children vertices in the *MGF* tree, resulting in little change to running time. **Table 10.17** shows that for some graphs, the saving in running time can be as much as 26% (4elt) but can also lead to an increase in as much as 50% (dime20), a result of the changing structure.

Change in Running Time								
m	3025	add32	data	4elt	sierpinski10	finan512	dime20	mesh100
2	742.5	1297.5	788.0	5530.8	38768.3	35595.0	87148.0	61184.0
3	799.8	1252.4	728.8	4357.8	40673.8	42245.6	98843.0	46847.0
4	855.4	1220.4	722.8	4057.0	39631.6	32763.4	86377.0	45459.0
5	735.0	1249.8	726.8	4253.2	41169.0	32346.0	103012.0	48126.5
6	709.0	1254.6	726.8	4315.8	46369.0	30953.8	106902.0	48305.5
7	711.2	1267.0	805.2	4091.2	52938.4	29057.8	130117.7	57850.0

Table 10.17. Change in running time as a result of changes to multimatching number

10.20.1.1 Multimatching using the Degree of Vertices

Average degree for each of the graphs is provided in Section 5.1 (Test Graphs), and is used as a base value for the Multimatching. Due to the degree being the number of adjacent vertices, the value is incremented by 1 to include the current vertex. Following analysis, there is no clear benefit when using

Multimatching values equal to the average degree of vertices, as summarised in Table 10.18. Subjective analysis confirms the findings.

Graph	m:=degree			Standard		
	RT	EC	ELR	RT	EC	ELR
3025	735.0	12.0	0.5363	742.5	0.0	0.4963
add32	1249.8	38008.6	0.5895	1297.5	16240.0	2.2534
data	795	41258.8	0.9875	788.0	36617.5	0.8887
4elt	4091.2	25509.4	0.7553	5530.8	25695.0	0.7073
sierpinski10	41169.0	40824.8	0.3274	38768.3	37189.0	0.3399
finan512	361334.7	4867662	0.8586	35595.0	5043279.0	0.9266
dime20	86377.0	230021.3	0.5709	87148.0	168011.0	0.5416
mesh100	48126.5	1302237.0	0.6058	61184.0	979875.5	0.5329

Table 10.18. Comparison of running time (RT), edge crossings (ER) and range in edge length (ELR) between a multimatching value of 2 and Multimatching value equal to the average degree of each test graph

10.20.2 Subjective Results

10.20.2.1 Multimatching Effect

Due to the way in which Multimatching affects the multilevel scheme, there are two primary effects on layout quality. Firstly, the number of vertices matched per vertex in the coarser graph means more work has to be done to untangle a graph during layout refinement. For a matching number of 7, for every vertex in a coarse graph, up to 7 vertices may exist for each, each of which take the initial position of the singular vertex and requires refinement to place them. If there are 100 coarse vertices, this could lead to 100 groups of 7 vertices attempting to have layout refined, pulling and pushing on each other and any connecting vertices. The result is tangles in the layout, as shown for sierpinski10 in Figure 10.54.

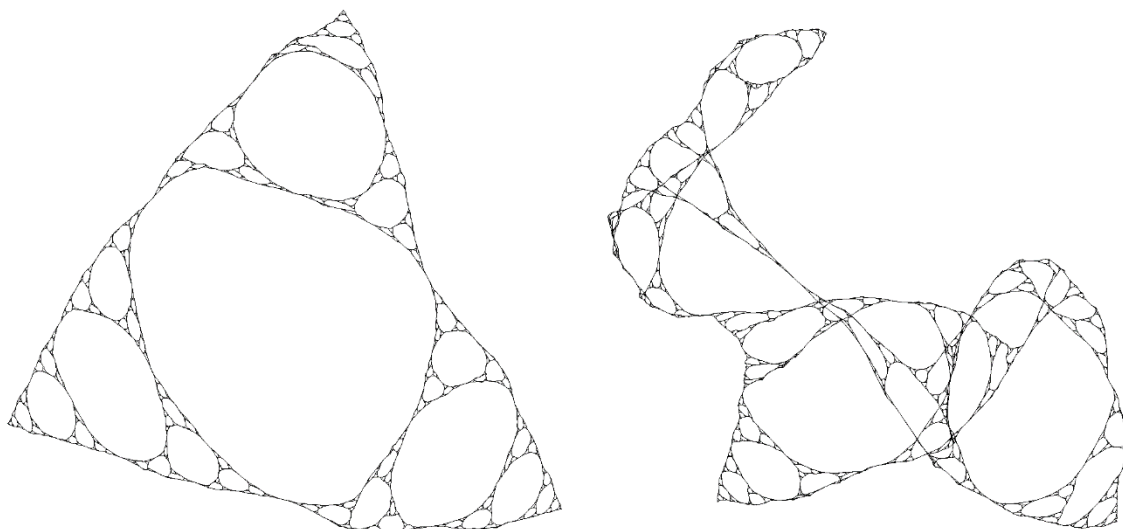


Figure 10.54. Comparison of layouts for *sierpinski10* using a Multimatching value of 2 (left) for standard edge contraction and a Multimatching value of 5 (right)

Secondly and more commonly, Multimatching affects the *MGF* structure, and as a result, the directions in which approximation is made. The higher the multimatching number, the more directions in which repulsive forces are approximated, expected to provide a more uniform layout. Figure 6.32 shows the most noticeable and appealing impact of this on the graph 3025, which shows a uniform layout that matches the known layout of the graph for a multimatching value of 6, removing the warping seen in layouts with lesser values.

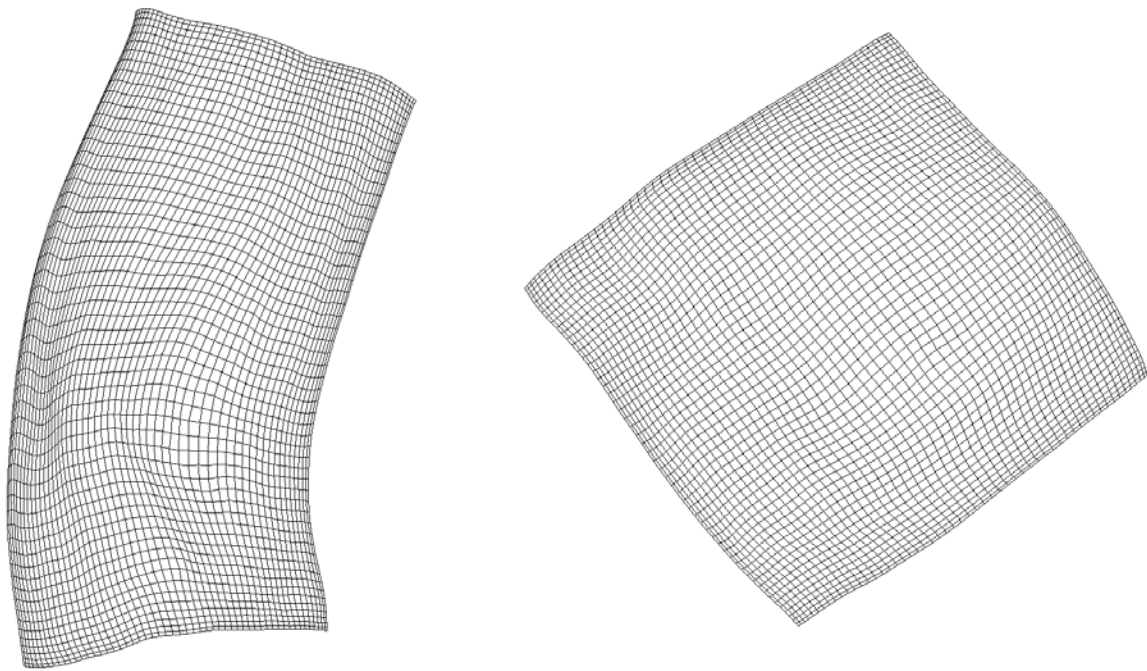


Figure 10.55. Comparison of layouts for 3025 using Multimatching values of 2 (left) and 6 (right), showing a more equilateral layout using multimatching

Figure 10.56 shows that the graph 4elt also exhibits some expansion, particularly of the branches, however the layout is still elongated, remaining visually similar to the layouts provided by standard matching methods.

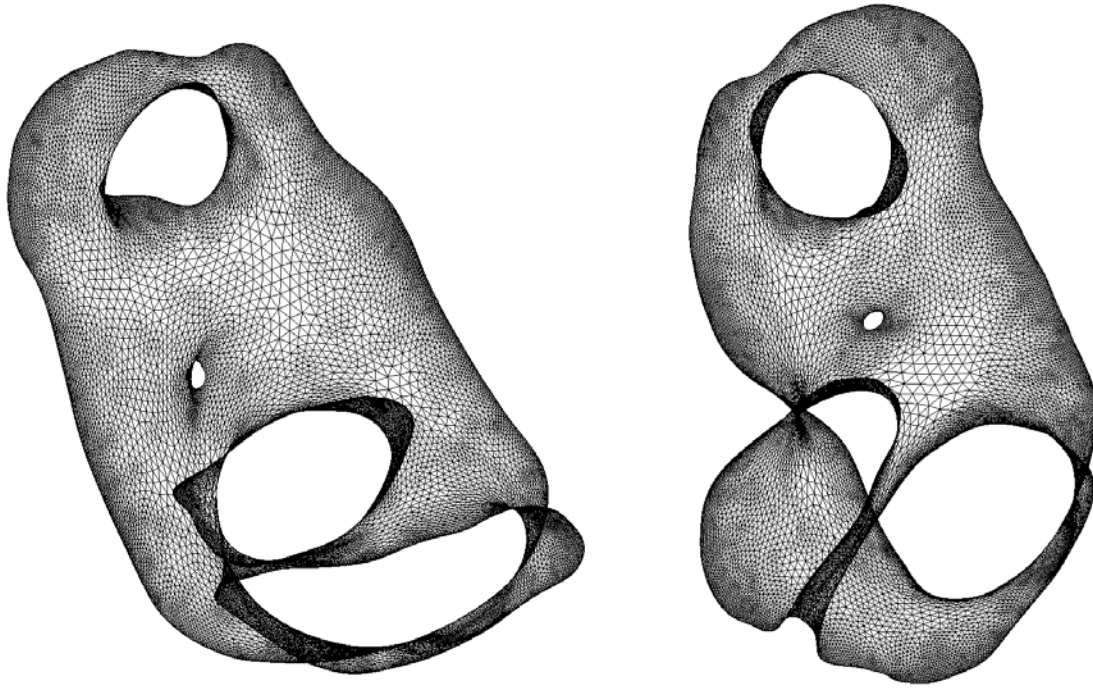


Figure 10.56. Comparison of layouts for 4elt for differing multimatching values, showing greater expansion of some branches for the greater value of 5 (right) than for standard edge contraction using a value of 2 (left)

10.20.2.2 Multimatching using the Degree of Vertices

As with the numerical analysis, subjective analysis suggests there is little improvement in layout quality for using values equal to the average degree of a graph. Some graphs, such as 3025 (Figure 10.57) show improvement in regard to reduced effect of warping, but can be provided with better layouts as a result of different Multimatching values. Other graphs show very little change such as add32 (Figure 10.58) or extreme deterioration of layout quality, such as sierpinski10 (Figure 10.59).

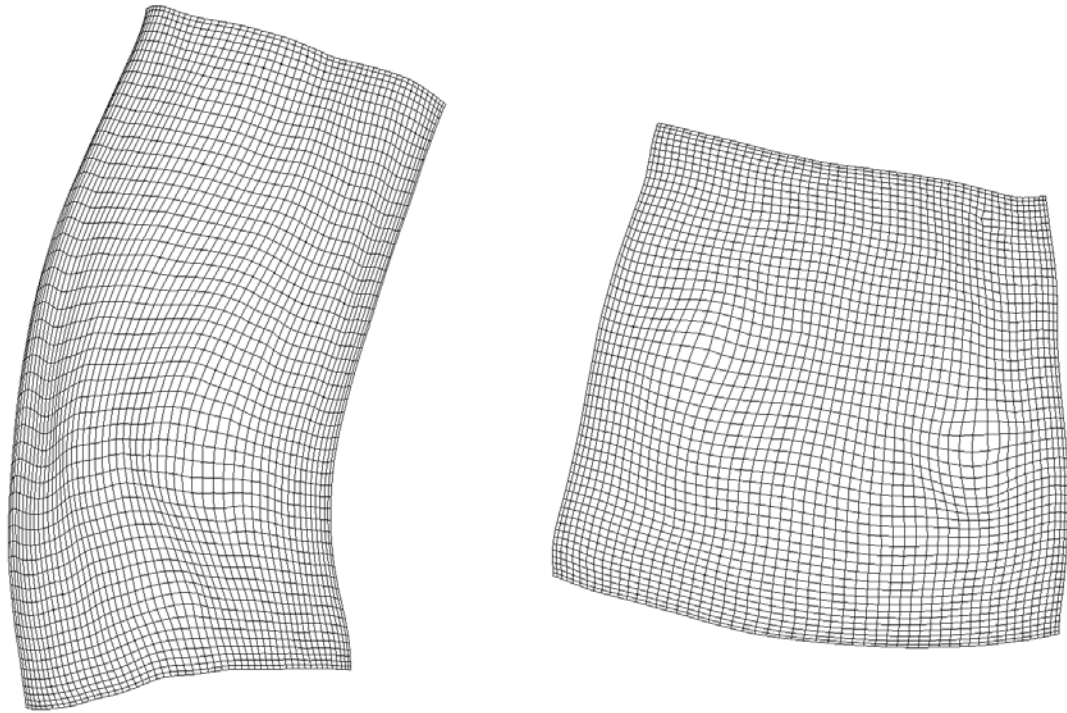


Figure 10.57. Comparison of layouts for the graph 3025 drawn using multimatching values of 2 (right) representing standard edge contraction and 5 (value of the average degree, left)

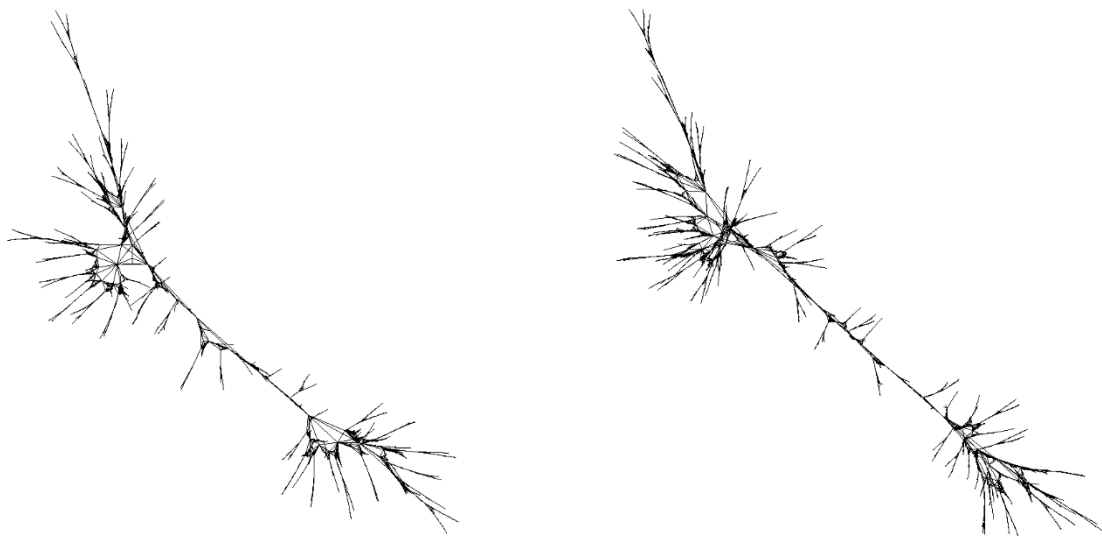


Figure 10.58. Comparison of layouts for the graph add32 drawn using multimatching values of 2 (right) representing standard edge contraction and 32 (value of the average degree, left)

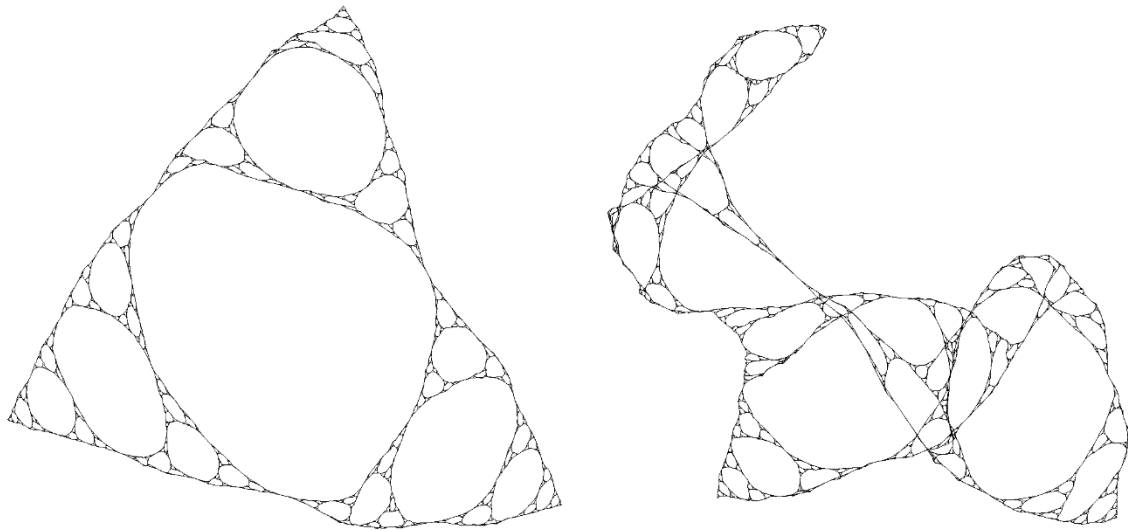


Figure 10.59. Comparison of layouts for the graph `sierpinski10` drawn using multimatching values of 2 (right) representing standard edge contraction and 5 (value of the average degree, left)

10.21 Graphs

10.21.1 Leafy Graphs

Leafy graphs are sparse graphs with a large number of vertices with a degree of 1, meaning that many vertices are connected to only one other vertex. Although there is no definition here on which graphs are considered leafy, the term is used to describe any graph which has leaf vertices. Generally, the term is used to describe graphs with many leaf vertices.

10.21.2 Dense Graphs

Dense graphs are highly connected and have a high number of edges per vertex. Such high concentrations of edges increase the energy of force directed placement systems, causing conflicting movements and preventing a low energy layout being generated. Reference to dense graphs therefore suggests such high numbers of edges that the layout is unlikely to provide a good representation of the data.

10.21.3 Sparse Graphs

Sparse Graphs are opposite to dense graphs, in that they have fewer edges per vertex.

10.21.4 Regular Graphs

Regular graphs are those which have recurring subgraph structures, and are neither dense nor sparse in quality.

10.21.5 Known Layouts for Test Graphs

Some of the test graphs used throughout the thesis have a known layout from their source. Those that do are provided below.

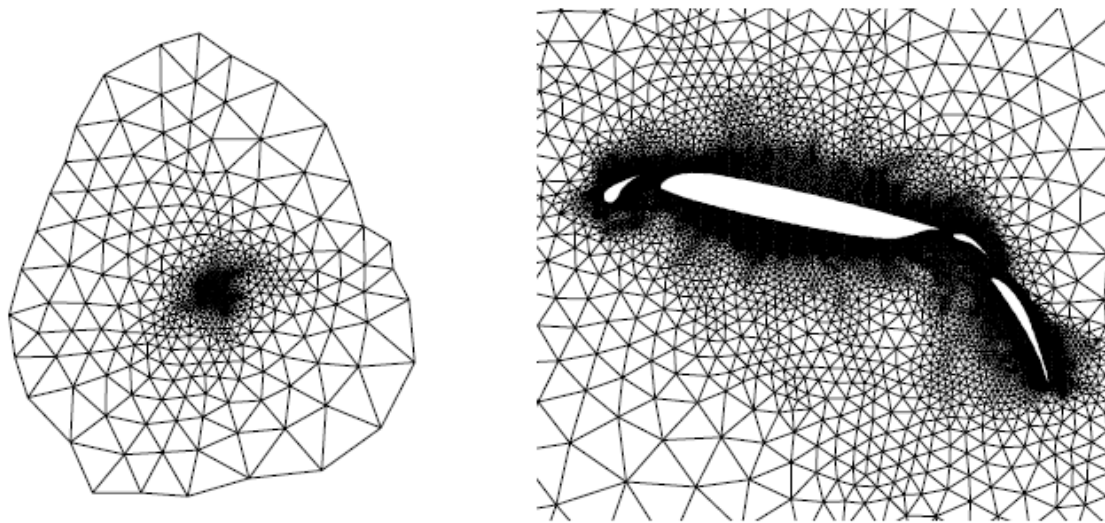


Figure 10.60 Actual layout for the graph 4elt



Figure 10.61 Actual layout for the graph data

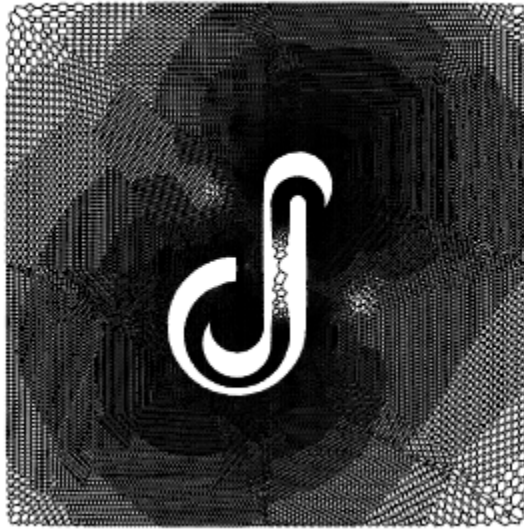


Figure 10.62 Actual layout for the graph dime20

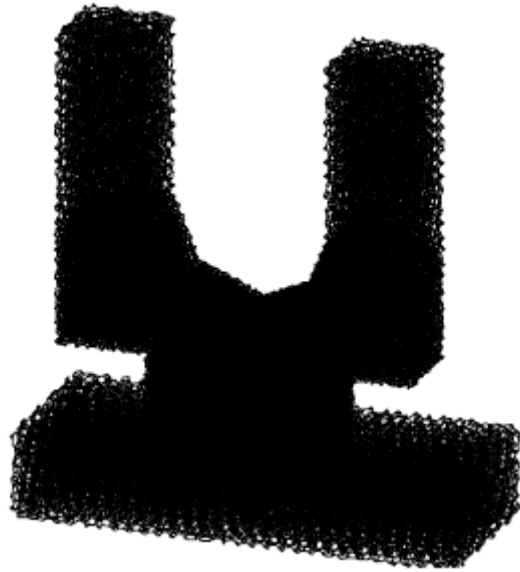


Figure 10.63 Actual layout for the graph mesh100

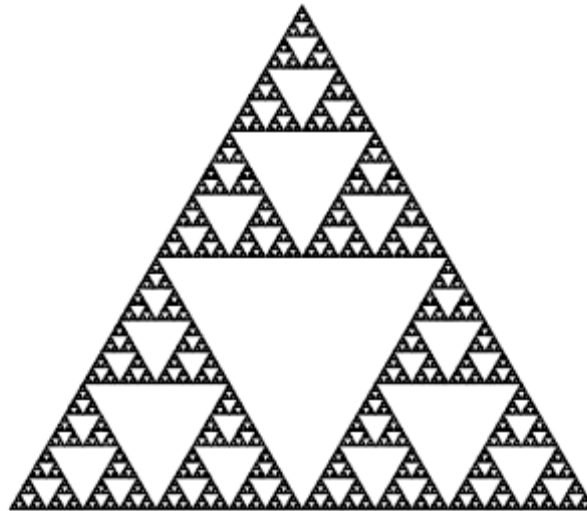


Figure 10.64 Actual layout for the graph sierpinski10

10.22 Implementation: Additional Information and Pseudocode

This chapter details the implementation of the techniques proposed in the Concepts and Experimentation chapters (Chapter 3 and 4 respectively) and used during experimentation and analysis. Each of the techniques are described in their respective subsections below with pseudo code examples provided. An object orientated programming approach is used to split the algorithms into simpler and more specific processes.

To introduce the environment for the implemented works, a description of the **Graph Drawing and Experimental Framework** is provided in Section 10.22.1 , detailing the overall intended framework and providing context for usage and the examples given. Following this, implementation of **Multilevel Global Force** is discussed in Section 10.22.3 , with details regarding parameters and coarsening methods for use on static graph drawing algorithms. **Dynamic Drawing Methods** are then described in Section 10.22.4 detailing the adaptation of the Spring Embedder for better visualising dynamic graphs, use of a multilevel scheme for generating and maintaining global layout and extension of the scheme for Multilevel Global Force approximation of long range global forces. Implementation of the Dynamic Update methods for use with a multilevel scheme and Multilevel Global Forces are included for graph modification (Section 10.22.4.4.2). A brief summary of the implementation is provided to conclude the chapter in Section 10.22.5 .

10.22.1 Graph Drawing and Experimental Framework

In order to provide a fair environment for comparisons of algorithms, the typical “components” which make up a graph drawing algorithm are implemented separately such that they can be “mixed and matched”, allowing for comparisons of differing algorithm makeups. A “component” refers to a principle part of a drawing algorithm which can be split off into its own sub-process, changes to which do not directly affect other sub-processes and data flow; for example, Multilevel Generation processes are separate from Force Directed Placement methods. A framework is used to govern and combine components, allowing for an experimental environment which can provide access to specific components.

Incorporation of such a framework can be found in existing research in the area, and is used to provide context for the intended usage of contributions (e.g. Misue et al (1995), Hu (2005), and Veldhuizen (2007)). The framework here is designed primarily for development purposes, providing an interface to control, monitor and test implementations of the new algorithmic approaches, while providing methods for automatic collection and analysis of numerical results. Although specific to the research here, it is expected the framework can be applied by others to simplify development or provide a controlled environment for experimentation. Figure 10.65 provides a basic overview of the architecture, splitting a typical algorithm into four “major” components, each of which can be split into more specific components:

- Input/Output – governing the methods of input and output, including the incorporation of input data into the model for drawing purposes. The component may be thought of as part of the model, however, is separated to keep data validation and control of input, separate from the model and data structures. In addition, the component is used to control analysis of results.
- Model – governing the data structures used, generation of multilevel schemes and approximation prior to use with drawing methods
- Automatic layout – the various methods for drawing layouts, including primitive placement and initial layout generation methods
- Visualisation – the display methods for visualizing drawings, providing methods for exploring layouts (focus, colouring, interactive controls) in addition to providing a console for use with

A controller is used as an interface to the components, combining them into a single package and providing a medium for decision making (for example, which automatic layout method to use).

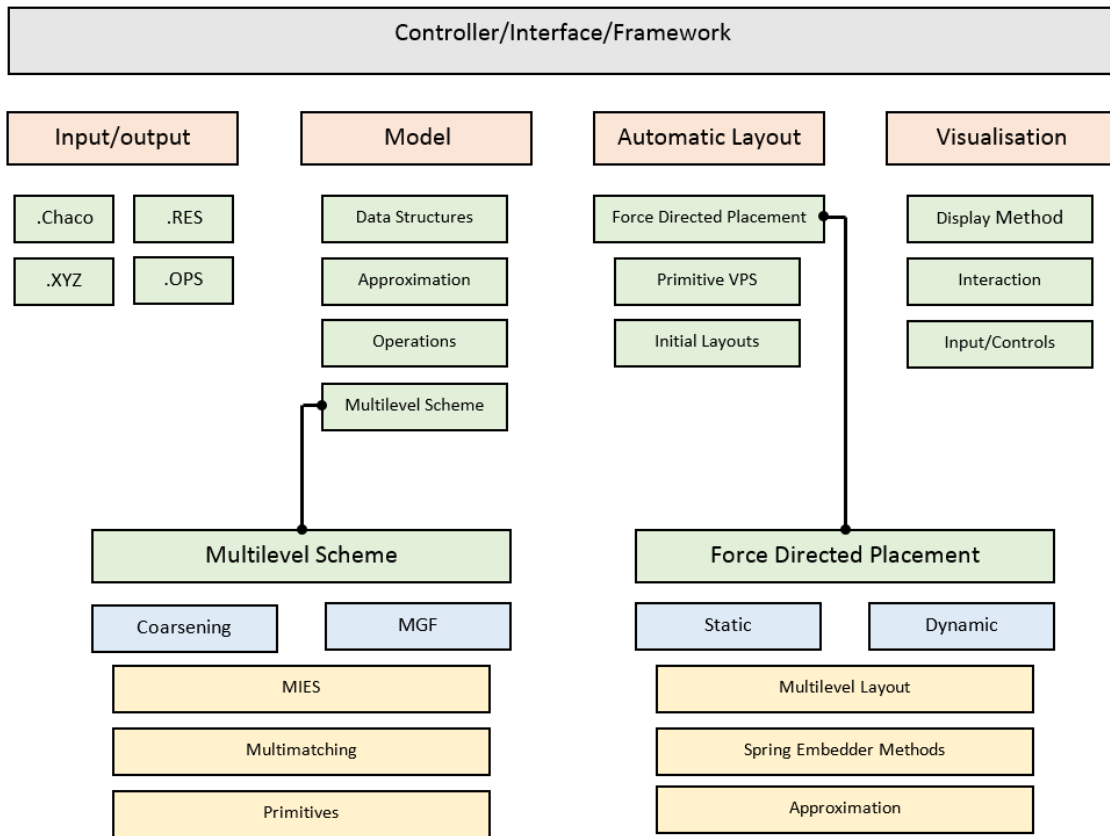


Figure 10.65 System Architecture detailing the four main components of the drawing algorithms generated here, indicating some of the functionalities available and providing context as to how components may be switched out to provide accurate comparison of techniques

The architecture is applicable to both static and dynamic graph drawing, usage of which is governed by the controller. An example of a dynamic drawing algorithm composed of the components is provided in Figure 10.66 below, showing the four major components and the selected sub-processes for each, with graph modification expected via an Operations file, and utilisation of both multilevel scheme and *MGF* approximation.

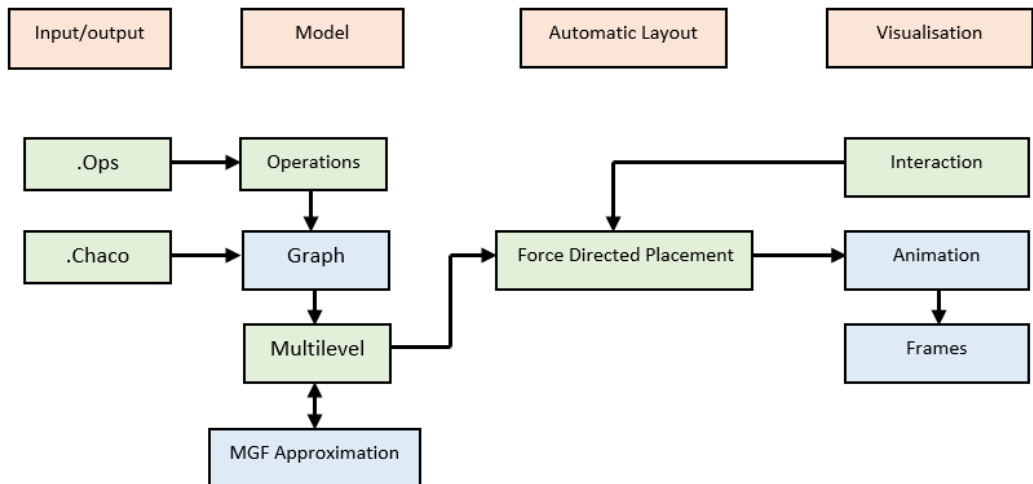


Figure 10.66 An example of a dynamic drawing algorithm using the framework, indicating usage of operations, multilevel scheme, *MGF* approximation and the intended use of animation for display purposes and the frames. Force Directed Placement is left ambiguous, suggesting the drawing method is likely to change

10.22.1.1 Data Structures

A graph, $G_0 = (V_0, E_0)$ is implemented as a collection of vertices, V_0 , with each vertex, v , containing various attributes. Edges are stored as a collection of adjacent vertices known as the neighbourhood of v , $\Gamma(v)$, within each vertex following implementation by Walshaw (2003) and Eades (1984). A graph, G_0 , is therefore composed of a collection of vertices, V_0 , with edges stored within them. A layout for a graph is the mapping of vertices to positions in a drawing area with line segments representing edges between them.

A vertex is implemented as a structure containing the following attributes:

vertex

ID : **int**

weight : **int**

xyz : **float**[]

{ neighbourhood of the vertex }

Γv : **vertex**[]

{ parent and children vertices for use in Multilevel schemes }

parent : **vertex**

children : **vertex**[]

{ store for displacement for use in Dynamic Multilevel Drawing }

Θ : **float**[]

{ secondary set of coordinates for use with multilevel aesthetics or MGF }

*xyz*_{MGF} : **float**[]

Figure 10.67 Vertex data structure

10.22.1.2 Multilevel Structure

Use of Multilevel Global Force and Multilevel schemes requires the pointers to be stored within a vertex (as shown in the example above). The pointers refer to the parent and children vertices in coarse and finer graphs, providing a means of traversing between levels of the multilevel scheme from any vertex within the graph. Storing the information within each vertex generates a double linked list structure.

10.22.1.3 Chaco and Operations Files

Static graphs are stored as Chaco files (Hendrickson and Leland, 1995) and may be accompanied by a plain text file (.xyz) file storing coordinates of vertices. A Chaco file includes a header identifying the number of vertices and edges within the file, with each vertex stored as a new line containing a list of adjacent vertex IDs separated by a space. An XYZ file similarly contains a header with the number of vertices in the document, with each line holding the coordinates of a vertex. An example of a Chaco and an XYZ file are provided below, more information can be found in the publication regarding Chaco file format by Hendrickson and Leland (1995).

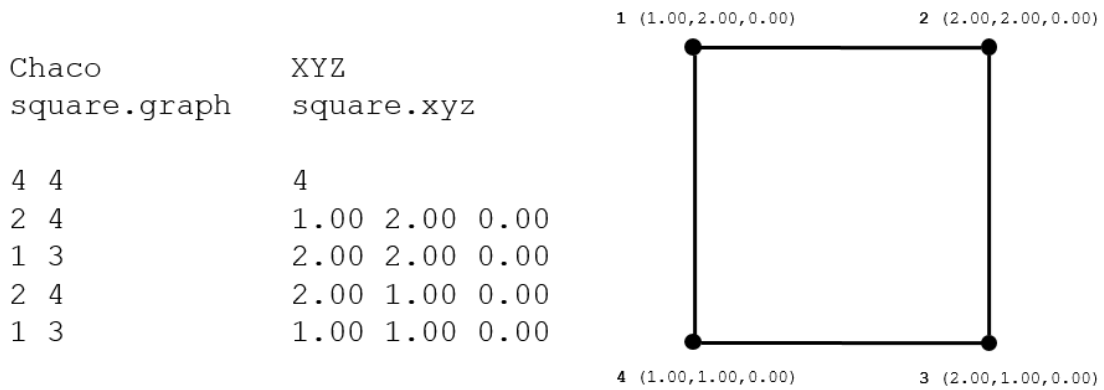


Figure 10.68 Example of Chaco and XYZ files for a simple graph of four vertices

A dynamic graph uses a Chaco file as an initial graph, with modification operations stored as an accompanying plain text operations file (.ops). The operations file stores a list of operations, following a protocol described in Table 10.19, providing one operation per line, identified by an ID with arguments to identify the target(s) of the operation (vertex ID). A dynamic graph need not include an operations file, instead receiving amendments in real time (unknown beforehand).

ID	Operation	Parameters	Description
0	Add Vertex	{ID} or {ID1, ID2}	Vertex with <i>ID</i> is added, and if given, connected to vertex <i>ID2</i>
1	Remove Vertex	{ID}	Removes vertex with <i>ID</i>
2	Add Edge	{ID1, ID2}	Adds an edge between vertices with <i>ID1</i> and <i>ID2</i>
3	Remove Edge	{ID1, ID2}	Remove an existing edge between vertices <i>ID1</i> and <i>ID2</i>
4	Break (automatic layout)	{N}	Breaks from operations, running an automatic layout algorithm for <i>N</i> iterations
5	Modification	{ID, weight} or {ID,X,Y,Z}	Modify vertex with <i>ID</i> , editing weight or position

Table 10.19 Protocol and expected parameters of input for Graph Modification Operations

More information regarding implementation of Graph Modification Operations is provided in Section 10.22.4.4.1 . Other files such as those used to store numerical results differ between experiments but are typically stored in a CSV (comma separated values) format.

It should be noted that after modification, a graph cannot have its layout saved as an XYZ file without a new Chaco file being saved due to differences between the graph and its initial state.

10.22.2 Multilevel Aesthetics

The graph aesthetic said to have greatest impact on readability of a layout are edge crossings, with uniformity of edge lengths also used to identify similarity between layouts and extent of the Peripheral Effect. Aesthetics are implemented as described in Concepts (see Section 3.5). However, usage of the multilevel scheme for analysis of global layout also requires graphs in the multilevel scheme to be updated with the average positions of vertices in the finest (original) graph as described below. Once updated, methods for analysing layout quality can be applied to any of the graphs in the multilevel scheme.

10.22.2.1 Generating Approximate Positions

Calculation of approximate positions and application to coarser graphs can be achieved through post order traversal of the Multilevel Global Force tree structure as described in Figure 10.69 below. The method starts with the second finest graph (G_1), wherein vertices take the averaged positions of their child vertices. The method is then repeated for the next (coarser) graph in the sequence, and continues until all graphs have been given approximate positions.

```

postorderXYZ( $G_L$ )
  for( $G_i \in G_L, i = 1$ )begin
    for( $v \in V_i$ )begin
       $v.xyz := 0.0$ 
      for( $u \in v.children$ )begin
         $v.xyz := v.xyz + u.xyz$ 
      end
       $v.xyz := v.xyz / |v.children|$ 
    end
  end

```

Figure 10.69 Generation of approximate positions of vertices in the multilevel scheme via post order traversal

For methods which require preservation of layout in graphs of the multilevel scheme (for example, dynamic drawing), the process above can be repeated using a secondary set of coordinates within each vertex, accessible in the above method as xyz_{MGF} .

10.22.3 Multilevel Global Force (*MGF*)

Multilevel Global Force is exploited from an existing multilevel scheme, therefore the generation of a multilevel scheme is described with examples in the subsection below. Following this, usage of *MGF* in Force Directed Placement is discussed using an example of the Fruchterman-Reingold (1991) Spring Embedder. Use of Primitives for modifying multilevel generation and altering *MGF* structure is then described, including pattern identification and Vertex Placement Scheme.

10.22.3.1 Multilevel Scheme

A coarse approximation of a graph is generated through identification and collapse of a maximal independent edge subset (a collection of edges which are not connected to one another), with each collapsed edge forming a new vertex in the coarse graph. The remaining edges of the original graph are then incorporated into the coarse graph, preserving the connections between vertices. The process is repeated on the coarser graphs until a graph of 2 vertices is generated, or some tolerance between levels is met, resulting in a collection of graphs $G_L = \{G_0, G_1, G_2 \dots G_n\}$.

Every edge that is collapsed during the coarsening process results in the merging of the two vertices connected by it to provide the generated (coarse) vertex with the sum of the weights. The two vertices are described as being “matched” with one another. Unmatched vertices are those which are not connected to a collapsed edge and are incorporated into the coarse graph on their own. When referenced, the collection of matched vertices is called the matching of a graph – indicating the current state of vertices (those merged and those not merged). A matching may change during dynamic drawing due to the addition and removal of edges and vertices.

In regard to implementation, the coarsening algorithm is split into sub-processes to achieve the above tasks. The **coarsener** method described in Figure 10.70 acts as a controller for multilevel generation, repeating the coarsening process on each graph and storing the coarsened graph for use in Force Directed Placement or *MGF*. The controller is simplistic in operation: if a graph has more vertices than a matching number m , the graph is coarsened using the coarsen method, with the generated approximation stored in G_L , repeating the process for the approximate graph until a graph with fewer than m vertices is generated, or some tolerance is met.

The tolerance refers to the difference between the number of vertices in two consecutive graphs becoming smaller than some value, by default 0 (that is generated graphs have no difference between

them). To prevent the algorithm ending early, the tolerance must be met three times consecutively. Section 10.2 provides experimentation of the tolerance for identifying whether values greater than 0 are more beneficial for some graphs.

```

coarsener( $G_0, m$ )
    { create collection to store sequence of graphs }
    create  $G_L$  []
    create  $G_n := G_0$ 
     $G_L$ .add( $G_n$ )
    count := 0;

    while( $|V_n| > m$ ) begin
        { generate a coarse approximation of  $G_n$  }
         $G_n :=$  coarsen( $G_n, m$ )
         $G_L$ .add( $G_n$ )

        { coarsening tolerance }
        if(( $|V_n| - |V_{n-1}|$ )  $\leq$  tolerance)
            count := count + 1;
            if(count = 3) end
        else count := 0
    end

    return  $G_L$ ;

```

Figure 10.70 Coarsener controller for generating graphs for a multilevel scheme

The **coarsen** function called in the **coarsener** is comprised of several sub-processes, the two most significant being **matchVertices** for identification and collapse of a maximal independent edge subset, and **importEdges** for preserving the remaining edges. Prior to this, additional functions are used to generate the empty shell of a new graph, G_{n+1} , to shuffle the vertices to avoid bias towards vertices which appear earlier in the collection, and a function to organise the shuffled vertices by weight to give priority to those vertices with smaller weight (in order to provide uniform vertex weights across coarser graphs).

```

coarsen( $G_n, m$ )
    { create a new empty graph }
    create  $G_{n+1}$ 

    { vertices are shuffled randomly and then organised by weight }
    shuffleVertices( $G_n$ )
    organiseByWeight( $G_n$ )

    { vertices of  $G_n$  are matched and edges preserved to form  $G_{n+1}$  }
    matchVertices( $G_n, G_{n+1}, m$ )
    importEdges( $G_n$ )

    { return the new coarse graph }
    return  $G_{n+1}$ ;

```

Figure 10.71 Coarsen method used to generate a coarse abstraction of a given graph

The most complex process of coarsening is the identification of a maximal matching. An independent edge subset identifies edges which do not share vertices (they are not connected to one another). Identification of an independent edge can be achieved by identifying vertices connected to one another, neither of which are connected to an already identified independent edge. Once achieved, the vertices are marked as matched such that any connecting edges cannot be identified as independent. The **matchVertices** function in Figure 10.72 performs this task, temporarily storing matched vertices until an independent edge can be confirmed, and then contracting the edge to form a vertex in the coarse graph.

A matching number m is used to determine the number of vertices matched together with a default value of 2 used for standard edge contraction. Higher values identify independent edge clusters (those connected to the same vertex) and is used in Multimatching (see Section 10.20). A maximal matching refers to a matching whereby a maximal number of independent edges or edge clusters are identified (i.e. there are no remaining unidentified independent edges). Unmatched vertices which are not connected to an independent edge or edge cluster are matched with themselves in the coarse graph.

After completion, all vertices will be marked as matched, each with a designated parent vertex that has the combined weight of the merged vertices which form it.

```

matchVertices( $G_n, G_{n+1}, m$ )
  while( $m > 0$ ) begin
    for ( $v \in V_n$ ) begin
      if( $v.matched = false$ )
        { generate array to store vertices as matched }
        create  $match[m]$ 
         $match.add(v)$ 

        { check neighbourhood of  $v$  for unmatched vertices }
        for( $u \in \Gamma v$ ) begin
          if( $u.matched = false$ )
             $match.add(u)$ ;
          if( $|match| = m$ ) end
        end

        { if  $m$  unmatched vertices are found }
        if( $|match| = m$ )
          create vertex  $w$  in  $V_{n+1}$ 
           $w.children = match$ 
          for( $u \in w.children$ ) begin
             $u.matched := true$ 
             $u.parent := w$ 
             $|w| := |w| + |u|$ 
          end
          else  $match := null$ 
        end
      end
       $m = m - 1$ ;
    end
  end

```

Figure 10.72 Function for matching vertices and identifying a maximal independent edge subset (MIES), a sub-process of coarsening

After a maximal matching is found, the vertices are coarsened to form a new coarse graph. Edges which have not been matched are preserved and incorporated between the coarser vertices in the coarse graph using the **importEdges** function in Figure 10.73. The approach determines which edges to preserve through analysis of parent and child relationships: a vertex and adjacent vertex will have the same parent

if matched, therefore preservation of the edge is not required; in contrast, a vertex and adjacent vertex with different parents requires the preservation of the edge to connect the parents. If multiple edges connect two vertices, only one is used. Weighting can be applied if required but is not used here following implementation by Walshaw (2003).

```

importEdges( $G_n$ )
  for( $v \in V_n$ )begin
    { get parent vertices of adjacent vertices }
    for( $u \in \Gamma v; v.parent \neq u.parent$ )begin
       $w_0 := v.parent$ 
       $w_1 := u.parent$ 

      { check if edge currently connects parent vertices }
      if( $e(w_0, w_1) \notin E$ )
         $\Gamma w_0$ .add( $w_1$ )
         $\Gamma w_1$ .add( $w_0$ )
      end
    end
  end

```

Figure 10.73 Function for incorporating edges of a finer graph G_i into a coarse approximation of that graph G_{i+1} , a sub-process of coarsening

After completion of the **coarsen** function, a coarse graph is returned to the **coarsener**, which repeats the process until the multilevel scheme is generated.

10.22.3.1.1 Multimatching and Brute matching

Multimatching uses the same implementation as above, with differing values of m for the desired matching number. Similarly, brute matching is achieved by setting the matching number to a value greater than the maximum vertex degree within the graph. Neither methods require changes to the implementation.

10.22.3.2 Spring Embedder with Multilevel Global Force

Multilevel Global Force is designed for application in approximating global repulsive forces in force directed placement. Multilevel Global Force is generated as described in the Concepts chapter (Section 3.6), and implemented using a multilevel scheme described by Walshaw (2003) in addition to the Fruchterman-Reingold (1991) Spring Embedder as the method of Force Directed Placement. For use as approximation however, vertices must point to both parent and children vertices, and graphs in the

multilevel scheme must be updated with the approximate positions of vertices in the finer graph using the method described in Section 3.6.1 . The spring embedder is modified such that N-body repulsive force calculation is replaced with the use of *MGF*, which traverses the multilevel scheme using the parent-children relationships stored within each vertex.

```

{ initialisation }
function  $f_r(x, w) = \text{begin return } -Cwk^2/x \text{ end}$ 
function  $f_a(x) = \text{begin return } x^2/k \text{ end}$ 

converged := false;
while(converged = false) begin
    converged := true
    create  $\Theta[V]$ 

    for( $v \in V$ ) begin
        { calculate global repulsive forces using MGF }
        create vertex  $w := v$ 
        while( $w.parent \neq null$ ) begin
            for( $u \in w.parent.children, u \neq w$ ) begin
                 $\Delta := u.xyz - v.xyz$ 
                 $\Theta[v] := \Theta[v] + (\Delta/\|\Delta\|) \cdot f_r(\|\Delta\|, |u|)$ 
            end
             $w := t$ 
        end

        { calculate local spring forces }
        for( $u \in \Gamma v$ ) begin
             $\Delta := u.xyz - v.xyz$ 
             $\Theta[v] := \Theta[v] + (\Delta/\|\Delta\|) \cdot f_a(\|\Delta\|)$ 
        end
    end

    { reposition vertices }
    for( $v \in V$ ) begin
         $\Delta := (\Theta[v]/\|\Theta[v]\|) \cdot t$ 
         $v.xyz := v.xyz + \Delta$ 
        if( $\|\Delta\| > k \cdot tol$ ) converged := false
    end

     $t = cool(t)$ 
end

```

Figure 10.74 Spring Embedder with Multilevel Global Forces for approximating repulsive forces

10.22.3.2.1 Approximation Limit

In addition to the approximation of global repulsive forces, an approximation limit is used to prevent repulsive forces between distant approximations. The approach is used for comparison against the standard technique of limiting calculation by distance, r , described by Fruchterman and Reingold (1991) and used by Hu (2005), however, uses the structural connectivity (distant as derived by the multilevel scheme) to inhibit repulsive force calculations.

The method is applied as a limit and a counter, whereby for each approximation made against the current vertex, the counter is incremented. If the limit is reached ($counter := limit$), calculation of repulsive forces is discontinued, preventing further traversal of the *MGF* tree.

```
{ initialisation }
w := v;
count := 0;

while(w.parent ≠ null) begin
    if(count ≥ limit) end

    { calculation of repulsive forces }
    for(u ∈ w.parent.children, u ≠ temp) begin
        Δ := u.xyz - v.xyz;
        Θ[v] := Θ[v] + (Δ/||Δ||) · fr(||Δ||, |u|);
    end
    w := w.parent;

    { count is incremented as the next level of approximations begin }
    count := count + 1;
end
```

Figure 10.75 Inclusion of an approximation into repulsive force calculation using *MGF*

10.22.3.3 Multilevel Static Graph Drawing Algorithm

Use of the multilevel scheme to provide global layout in addition to Multilevel Global Force requires a combination of the methods above. The algorithm initially reads in a Chaco file and generates a multilevel scheme for the graph, followed by the application of an initial layout to the coarsest graph by random placement of vertices within the drawing area. Following this initialization, graphs are

provided with a layout using Force Directed Placement, which is then interpolated, resulting in refinement of the layout at each level (coarsest to finest). The Multilevel Global Force is then updated with positions using the **postOrderXYZ** function described in in Figure 10.76. The method continues until the original graph, G_i , is provided with a layout which is displayed to a computer monitor.

```

{ initialise }
 $G_0 := \text{readIn}(\text{graph})$ 
 $G_L := \text{coarsener}(G_0)$ 
 $i = |G_L|$ 

{ provide coarsest graph with initial layout }
initialLayout( $G_n$ )

{ apply layout to each graph and interpolate layout }
for( $G_i \in G_L, i --$ )begin
    FDP( $G_i$ )
    interpolateLayout( $G_i, G_{i-1}$ )

    { update MGF with new approximate positions }
    postOrderXYZ( $G_i$ )
end

display( $G_i$ )

```

Figure 10.76 Multilevel Static Graph Drawing Algorithm design

10.22.3.4 Primitives

Primitives are used in two cases –coarsening of primitives and primitive graphs, as described in Appendix 10.10 . Both methods are initially applied in the **coarsener** controller (Figure 10.70) with calculated layout provided during application of Force Directed Placement. Figure 10.77 below provides an example of the methods usage in coarsening, utilising the **isPrimitive** function to identify if a graph, or coarsened graph, has a known structure which can be given a calculated layout using the **Vertex Placement Scheme** (Section 10.22.3.4.2). In addition, the **hasLeaves** function is used to identify and coarsen a graph featuring leaf vertices which can also be given a calculated layout from the **Vertex Placement Scheme**. Specifics of the two methods are provided in the subsections below.

```

{ generate coarse graphs }
while( $|G_n| > m$ ) begin

    { check if primitive graph }
    if(is Primitive( $G_n$ ))
        return  $G_L$  end
    { check if graph has primitive }

    if(has Leaves( $G_n$ ))
         $G_n :=$  coarsenPrimitives( $G_n$ )
         $G_L$ .add( $G_n$ )
    else
        { normal coarsening scheme as depicted in Figure 4 }
        ...
    end
return  $G_L$ 

```

Figure 10.77 Incorporation of primitive processing into the coarsening example provided in **Figure 10.70**

10.22.3.4.1 Pattern Identification and Coarsening

Each primitive has some known pattern exhibited by the vertices, the existence of which can be confirmed by analysing the connectivity of vertices within the graph. Implementation uses the criteria to first identify whether a graph exhibits any primitive, after which, the effects to coarsening are applied. Table 10.20 provides a description of the patterns and example pseudo code of their application as identification for general connected graphs. The primitives are split into those which may apply to vertices within a graph (coarsening) and those classified as primitive graphs (graph).

Primitive	Pattern	Pseudo code
Leaf (coarsening)	Any vertex with a degree of 1.	for ($v \in V$) begin if ($ \Gamma v = 1$) return true ; end

Chain (coarsening)	Any leaf vertex connected to a vertex with a degree of 2. A chain can have more than two vertices, providing a leaf vertex and all connecting vertices having a degree of 2.	<pre> for($v \in V$) begin if($\Gamma_v = 1$) if($u \in \Gamma_v, \Gamma_u = 2$) return true end </pre>
Star (graph)	All vertices have a degree of 1, except one vertex with degree $ V - 1$	<pre> $centre := 0$; for($v \in V$) begin if($\Gamma_v > 1$) $centre := centre + 1$; if($centre > 1$) return false; end return true; </pre>
Ring (graph)	Each vertex in the graph has a degree of 2	<pre> for($v \in V$) { if($\Gamma_v > 2$) return false; if($\Gamma_v < 2$) return false; end return true </pre>
Chain (graph)	A graph will exhibit two leaf vertices (each with degree of 1) with all other vertices exhibiting a degree of 2	<pre> $ends := 0$; for($v \in V$) if($\Gamma_v > 2$) return false; if($\Gamma_v = 1$) $ends := ends + 1$; if($ends > 2$) return false; end if($ends = 2$) return true; </pre>

Table 10.20 Methods for identifying primitives through pattern recognition

Primitive graphs end the coarsening scheme and therefore require no additional coarsening methodologies. However, the primitive coarsening patterns (leaf and chains of vertices) require additional coarsening methods. A general coarsening method for both primitives is described in Figure 10.78 due to shared qualities (leaf vertices).

The coarsening method identifies leaf vertices within a graph, then determines if they are part of a chain of vertices. If a chain is identified, a recursive function, **trail**, is used to find the end of the chain - a vertex with degree greater than 2 which connects the chain to the body of a graph. For each, the vertices which make up the primitive are coarsened to form vertices of a simplified graph. All non-primitive vertices are matched with themselves and preserved with connecting edges in the new coarse graph. Coarsening then continues as described Figure 10.71 and Figure 10.70.

```

create  $G_{n+1}$ 
for ( $v \in V_n$ ) begin
    if ( $|\Gamma_v| = 1, v.matched = false$ )
        create vertex  $u := \Gamma_v.get(0)$ ;
        if ( $|\Gamma_u| = 2$ )
            create  $match[]$ ;
            trail( $v, u, match$ )
        else
            if ( $u.matched = false$ )
                create vertex  $w$  in  $V_{n+1}$ 
                 $w.children.add(\{v, u\})$ 
            else
                create vertex  $w := u.parent$ 
                 $w.children.add(v)$ 
                 $v.parent := w$ 
        end
    end

function :trail( $v, u, match$ ) begin
     $v.matched := true$ ;
     $match.add(v)$ ;
    if ( $|\Gamma_u| = 2$ )
        create vertex  $w := \Gamma_u[0 | 1], w \neq u$ 
        trail( $u, w, match$ )
    else
        create vertex  $w$  in  $V_{n+1}$ 
         $w.children := match$ 
    end

```

Figure 10.78 Method for pattern coarsening of leaf vertices and chains (primitives)

10.22.3.4.2 Vertex Placement Scheme

The vertex placement scheme is designed to provide primitive vertices with calculated positions. The scheme is a collection of methods which can be called upon to provide a layout for the primitive shapes described in Table 10.20. Similarly, Table 10.21 provides a list of the patterns used to generate the positions and example pseudo code used to calculate layouts for general connected graphs. The methods are described for two dimensional drawings only.

Primitive	Layout Pattern	Pseudo code
Leaf (coarsening)	Each leaf is connected by an edge which can be orientated such that the anchor points towards the centre of mass and the leaf points outwards away from the centre of mass	<pre> gxyz = centreOfMass() for(v ∈ V, Γv = 1) begin create vertex u := Γv.get(0) Δ := v.xyz - u.xyz v.xyz := u.xyz - ((gxyz - u.xyz) / Δ) · k end </pre>
Chain (coarsening)	Chains are split into leaf nodes, beginning from the anchor, using the leaf layout method recursively to provide layout for each vertex in the chain until the end leaf is given layout	
Star (graph)	The central vertex is placed at the centre of the drawing area with all other vertices placed evenly around it a distance of k away	<pre> θ := 2π / (G - 1); i := 0; for(v ∈ V) begin if(Γv > 1) v.xyz := [0.0,0.0,0.0]; else v.x := k cos(i · θ); v.y := k sin(i · θ); i := i + 1; end </pre>

Ring (graph)	Vertices are placed evenly around the centre of the drawing area, with spacing of k between each. Vertices must be in order to ensure connected vertices are next to each other (achieved by organising vertices based on connectivity)	<pre> <i>circ</i> := <i>G</i> · <i>k</i>; <i>r</i> := (<i>circ</i> / π) / 2.0; θ := $2\pi / G$; <i>i</i> := 0; for($v \in V$) begin <i>v.x</i> := <i>r</i> · cos(<i>i</i> · θ) <i>v.y</i> := <i>r</i> · sin(<i>i</i> · θ) <i>v.z</i> := 0.0 <i>i</i> := <i>i</i> + 1; end </pre>
Chain (graph)	Vertices are placed in a long line, with the centre of the chain located in the centre of the drawing area (with even number of vertices on each side).	<pre> create vertex [<i>ends</i>] <i>i</i> = 0; for($v \in V$) begin if($\Gamma v = 1$) <i>ends</i>[<i>i</i>] := <i>v</i> <i>i</i> = <i>i</i> + 1 end Δ := <i>ends</i>[0].<i>xyz</i> - <i>ends</i>[1].<i>xyz</i> <i>ends</i>[0].<i>xyz</i> := $\Delta - (V / 2) \cdot k$ create vertex <i>u</i> := <i>ends</i>[0] create vertex <i>v</i> := Γv[0] while($v \neq ends[1]$) begin <i>v.xyz</i> := <i>u.xyz</i> + <i>k</i> create vertex <i>t</i> := <i>u</i> <i>u</i> := <i>v</i> <i>v</i> := Γv[0 1], $\neq t$ end </pre>

Table 10.21 Methods for layout calculation for primitive graphs and shapes

The vertex placement scheme is designed for use within the multilevel scheme, prior to application of force directed placement. Primitive graphs are provided a layout as soon as the coarsening method ends. However, leaf vertices and chains of vertices are provided a layout during multilevel layout generation described in Table 10.21, with an example of usage shown in Figure 10.79, showing the use of

primitive layout as a method for providing initial layout, and use of primitive layout during multilevel layout generation.

```

 $G_L := \text{coarsener}(G_0)$ 
 $i := |G_L|$ 

{ if primitive graph, provide calculated layout }
if( $G_i$ .isPrimitive())
    setLayout( $G_i$ )
else
    initialLayout( $G_i$ )

{ apply layout to each graph and interpolate layout }
for( $G_i \in G_L, i --$ )begin

    {if graph exhibits primitives provide layout to them }
    if( $G_i$ .hasPrimitives())
        setLayout( $G_i$ )
    else
        FDP( $G_i$ )

    interpolateLayout( $G_i, G_{i-1}$ )
    postOrderXYZ( $G_i$ )
end

display( $G_0$ )

```

Figure 10.79 Alteration of Multilevel Static Drawing algorithm to include Primitive processing

10.22.4 Dynamic Graph Drawing

The methods used in dynamic graph drawing are largely similar to those used in static drawing above using the same underlying spring embedder (same forces and algorithmic design), but including adaptations to optimise for visualisation of dynamic graphs. The methods are described below, firstly detailing an adaptation of the spring embedder described by Fruchterman and Reingold (1991), followed by a description of Multilevel Scheme usage for generating global layout, finishing with the

implementation of Dynamic Matching update methods for incorporating operations into the multilevel scheme for global layout and approximation.

10.22.4.1 Dynamic Spring Embedder Adaptation

The adaptation refers to the alteration of the cooling schedule of the modified spring embedder described by Fruchterman and Reingold (1991), optimising the visualisation of the drawing process. The approach replaces the cooling schedule with constants to better control the displacement resulting from individual forces, and provides much smoother vertex movement over the original (which features erratic and energetic vertex movement). The method is therefore similar to the method described by Fruchterman and Reingold (1991), however differences can be seen in the calculation of displacement as shown in Figure 10.80 below. The method stores the displacement of forces separately (Θ_a and Θ_r) using the constants t_a and t_r to alter the equilibrium of the forces when combined to generate vertex movement.

```

{ initialisation }
 $t_a := 0.9$ 
 $t_r := 0.6$ 
create  $\Theta_a$  [ |  $V$  | ]
create  $\Theta_r$  [ |  $V$  | ]

for( $v \in V$ ) begin
    for( $u \in V, u \neq v$ ) begin
         $\Delta := u.xyz - v.xyz$ 
         $\Theta_r[v] := \Theta_r[v] - (\Delta / \|\Delta\|) \cdot f_r(\|\Delta\|, |u|)$ 
    end

    for( $u \in \Gamma v$ ) begin
         $\Delta := u.xyz - v.xyz$ 
         $\Theta_a[v] := \Theta_a[v] + (\Delta / \|\Delta\|) \cdot f_a(\|\Delta\|)$ 
    end
end

for( $v \in V$ ) begin
     $v.\Theta := ((\Theta_r[v] / \|\Theta_r[v]\|) \cdot t_r) + ((\Theta_a[v] / \|\Theta_a[v]\|) \cdot t_a)$ 
     $v.xyz := v.xyz + v.\Theta$ 
end

```


Figure 10.80 Dynamic Spring Embedder, an adaptation of the Fruchterman-Reingold (1991) static spring embedder for use in Dynamic graph drawing

10.22.4.2 Multilevel Layout Generation

In order to provide a multilevel layout, an approach similar to the static methodologies is provided. A layout is generated for each level of the multilevel scheme as before, however, force directed placement is applied once per level (as opposed to iteratively) with the displacement of vertices interpolated between layouts, resulting in global movement in addition to local vertex movement. The method, described below in Figure 10.81, reuses the spring embedder above using a similar approach to that described by Walshaw (2003), however, the force directed placement method is run for 1 iteration per level (unlike static drawing which runs *FDP* until a layout is generated for each level) with the entire multilevel scheme repeated to provide a layout.

```

function  $k(x, a) = \mathbf{begin}$  return  $n.n$  end

 $G_L := \mathbf{coarsener}(G_0)$ 
initLayout( $G_L$ )
 $i := |G_L|$ 

for( $G_i \in G_L, i--$ ) begin
     $k := k(|V_i|, A)$ 
    FDP( $G_i, k$ )

    if( $i > 0$ )
        for( $v \in V$ ) begin
            for( $u \in v.children$ ) begin
                 $u.xyz := u.xyz + v \cdot \Theta$ ;
            end
        end
    end

end

checkOperations()

```

Figure 10.81 Multilevel algorithm for layout generation and adjustment using the Dynamic Spring Embedder Adaptation

The method can be applied to both layout generation and adjustment, however, operations can only be applied before or after each iteration of the method in order to prevent interference during calculation

of layout changes. Conversely, in order to provide layout generation during graph modification, operations provide “breaks”, periods of uninterrupted Force Directed Placement, to ensure layout generation is applied and allow the layout can react to the changes.

10.22.4.2.1 Initial Positions

Usage of the multilevel scheme for dynamic drawing requires vertices in the original graph to be provided with initial positions which coincide with positions of approximate vertices in the coarser graphs, in order to prevent conflicting movements as a result of local and global changes to layout. As such, a simple heuristic is provided in Figure 10.82, which uses a multilevel scheme to provide initial positions of vertices such that closely connected vertices are drawn near one another.

The method provides an initial random layout to the coarsest graph, with the initial layout interpolated to child vertices, using an ideal separation factor described by Walshaw (2003) to place vertices near the parent vertex to avoid expensive untangling of the graph and avoiding conflicting movements during multilevel layout generation. In addition to separation factor, a scale value is provided to expand the layout such that early forces shrink the layout (avoiding expansion which may overpower layout generation in initial layouts, resulting in poorer vertex placement).

$$separation := \sqrt{4/7}$$

randPos(*xyz*, *k*)

return *xyz* + (**random**(0.0,1.0) – 0.5) · (*separation* · *k*)

initLayout(*G_L*, *size*, *scale*)

i = | *G_L* |

initRandPos(*G_i*, *size*);

for(*G_i* ∈ *G_L*, *i* – –) **begin**

k := *scale* · *k*(*size*, | *G_i* |);

for(*v* ∈ *V_i*) **begin**

v.*xyz* := **randPos**(*v*.*parent*.*xyz*, *k*);

end

end

Figure 10.82 Method for calculating initial positions of vertices in a multilevel scheme, ensuring vertices are near their coarser representations in the multilevel scheme

10.22.4.3 Dynamic Spring Embedder with Multilevel Global Force

Due to the need to preserve coordinates of coarser graphs for global layout, approximate positions used for Multilevel Global Force do not overwrite the positions of coarse vertices. As such, the difference requires force directed placement to use an additional coordinates system such that approximate position and actual position of vertices in the multilevel scheme are stored separately. Consequently, the algorithm in Figure 10.80 is altered such that repulsive forces are calculated as indicated in Figure 10.83.

```
for( $v \in V$ )begin
    for( $u \in V, u \neq v$ )begin
         $\Delta := u.xyz_{MGF} - v.xyz$ 
         $\Theta_r[v] := \Theta_r[v] + (\Delta / \|\Delta\|) \cdot f_r(\|\Delta\|, |u|)$ 
    end
    ...
end
```

Figure 10.83 Alteration of repulsive force calculation in the Dynamic Spring Embedder for use of Multilevel Global Force

In addition to the alteration to repulsive forces, the method for updating the multilevel scheme with approximate vertex positions must also be updated to use the secondary coordinate system as described below in Figure 10.84. The method initially uses the “real” positions of vertices in the original graph, approximating them for the secondary coordinate system in the parents, then continues to approximate the secondary coordinate system thereafter.

```

postorderXYZMGF ( $G_L$ )
  { initial approximation of real positions }
  for( $v \in V_1$ )
     $v.xyz_{MGF} := 0.0$ 
    for( $u \in v.children$ ) begin
       $v.xyz_{MGF} := v.xyz_{MGF} + u.xyz$ 
    end
     $v.xyz_{MGF} := v.xyz_{MGF} / |v.children|$ 
  end

   $i := 2$ 
  for( $G_i \in G_L$ ) begin
    for( $v \in V_i$ ) begin
       $v.xyz_{MGF} := 0.0$ 
      for( $u \in v.children$ ) begin
         $v.xyz_{MGF} := v.xyz_{MGF} + u.xyz_{MGF}$ 
      end
       $v.xyz_{MGF} := v.xyz_{MGF} / |v.children|$ 
    end
  end
end

```

Figure 10.84 Alteration to post order update of approximate vertex positions

10.22.4.4 Dynamic Matching

Dynamic matching is the process of updating the multilevel and multilevel global force schemes with any operation performed on the original graph. The implementation is for four basic methods differing in the extent of the update, with discussion of each after description of the implemented operations available.

10.22.4.4.1 Operations

The operations are typical for graph drawing algorithms (Veldhuizen, 2007). For each, it is usual for an initial query to determine if the operation can be performed, leading to a worst case linear search of the sets ($O(n)$). However, use of pre-existing programming structures (specifically Hashing mechanisms) can largely reduce the complexity of each ($O(1)$). As a result, a worst case and average case complexity is described for each of the operations as a guide on usage.

Operation	Description	Pseudo code
-----------	-------------	-------------

Modification	Modify a vertex weight or position in layout, typically achieved through encapsulation of attributes (set methods of a vertex). Complexity: $O(1)$ average, $O(n)$ worst	if ($v \in V$) v . set (<i>new xyz</i>) v . set (<i>new weight</i>)
Add Vertex	Create and insert a new vertex v into the vertex set (V). Initial position can be given randomly. In addition, an edge can also be used to immediately connect a vertex to the graph and provide position near the anchoring vertex. Complexity: $O(1)$ average, $O(n)$ worst	if ($v \notin V$) create vertex v V . add (v)
Add Edge	Insert a new edge between two vertices, $e(v,u)$. Complexity: $O(1)$ average, $O(n)$ worst	if ($e(v,u) \notin E$) Γv . add (u) Γu . add (v)
Remove Vertex	Remove a vertex and any edges connecting the vertex to the graph. Complexity: $O(\Gamma v)$ average, $O(n)$ worst	if ($v \in V$) for ($u \in \Gamma v$) begin Γu . remove (v) end V . remove (v)
Remove Edge	Remove an edge between two vertices. Complexity: $O(1)$ average, $O(n)$ worst	if ($e(v,u) \in E$) Γv . remove (u) Γu . remove (v) E . remove ($e(v,u)$)

Table 10.22 Implementation of graph modification operations for dynamic graphs

10.22.4.4.2 Multilevel Update

Multilevel updates extend the usage of the methods above to impact the graphs of the multilevel scheme in one of four ways below with information regarding their implementation;

- **Low (l)** – updates are only applied to the original graph, G_0 , whereby changes do not alter the multilevel scheme in any significant way other than to join new vertices or remove them from the scheme for the purpose of *MGF*. Updates can be applied using the operations described in Table 10.22 and maintenance via the **caretaker** method described in Figure 10.85.

- **Medium (m)** – updates are applied to all graphs up to the middle of the multilevel scheme, thereafter which changes are **joined** to the mid-coarsest graph to ensure their inclusion in the multilevel scheme
- **High, Full (h)** – updates are made to the full multilevel scheme up to the coarsest graph, however, the number of graphs in the multilevel scheme is preserved
- **Rematch** – The entire multilevel scheme is rebuilt using the coarsening scheme as described in Figure 10.70, requiring no additional update methods for modifications (using those provided in Table 10.22)

The update process is split into two main processes: firstly, the possible modifications of the graph are described in Table 10.23 together with alterations for the multilevel scheme; and secondly, maintenance of the multilevel scheme and the extent of the updates. The adaptation of modifications for use in multilevel updates is minor, with most requiring parents or children being removed such that the maintenance (caretaker) can be applied to them or alteration to the preserved edges within the multilevel scheme. Both caretaker and edge preservation methods are described in the Figures below, which details the changes to the modification operations.

The following methods are used to implement updates to the multilevel scheme, using the graph modification methods above as a basis.

Description of Operation for Multilevel Usage	Pseudo code example
<p>Addition of a new vertex v means an unmatched vertex in V, providing no parent. Ensuring the parent is null will ensure the caretaker will find or generate a parent.</p> <p>In addition a new vertex connected to a vertex u results in a new edge to be preserved in coarser graphs.</p>	<pre> addVertex(v,u) {additional} $v.parent := null;$ preserveEdges($v,u,addition$) </pre>
<p>Addition of an edge will result in either a new matching, arranged by the caretaker, or will require preservation of the edge in coarser graphs.</p>	<pre> addEdge(v,u) {additional} if($v.parent \neq u.parent$ or $u.parent.child < m$) preserveEdges($v,u,addition$) </pre>

<p>Removal of a vertex is required from both the graph and its parent vertices children collection. Further to this, edges between the vertex and adjacent vertices are removed in both the original graph and possibly in coarser graphs.</p>	<pre> removeVertex($v \in V$) {additional} $t := v.parent$ $t.children.remove(v)$ for($u \in \Gamma v$) begin preserveEdges($v, u, removal$) end </pre>
<p>Removal of an edge will result in one of two things;</p> <p>if the vertices are matched with one another, the matching is undone and the parent vertex removed from the coarse graph</p> <p>if the vertices are unmatched, the removal of the preserved edge from coarser graphs</p>	<pre> removeEdge(v, u) {additional} if($v.parent = u.parent$) $v.parent := null$ $u.parent := null$ $G_{i+1}.remove(v.parent)$ else preserveEdges($v, u, removal$) </pre>

Table 10.23 Adaptation of graph modification operations for use as multilevel updates

The caretaker is a process which maintains the integrity of the multilevel scheme after modification operations have been performed. Such tasks included joining changes to the multilevel scheme by ensuring all vertices have a parent within the mapping tree, removing any ghost vertices and identifying new matches. The process is straightforward and is essentially an update version of the coarsen method in Figure 10.71 – matching unmatched vertices and preserving edges between graphs. The method is described in Figure 10.85 and is implemented as a collection of patterns, such that if a pattern is identified, some known behaviour can be incorporated for the current level. Due to the method’s focus on an individual level in the multilevel scheme, updates are designed to provide a means of continuing alteration on the next level (for example, a vertex unmatched can be provided with a new parent vertex which can be matched on the next coarse level when the caretaker is run again). In addition, the caretaker is responsible for incorporating the level limit associated with the four multilevel update methods.

```

while( $i < \{m \mid h\}$ )begin
    for( $v \in V_i$ )begin
        checkMatches( $v$ )
        checkParents( $v, i$ )
        checkGhostVertices( $v, i$ )
    end
end

{for level at which limit is set, join any vertices without parents }
for( $v \in V$ )begin
    if( $v.parent = null$ )
        join( $v, \Gamma v.get(0)$ )
end

```

Figure 10.85 Caretaker method used to maintain the multilevel scheme after operations have been performed

The caretaker method is split into the following processes to check if any new matches can be made (**checkMatches**), whether any vertices require parents (**checkParents**) and whether any ghost vertices exist (**checkGhostVertices**). The primary aim of the methods is to retain accuracy of the approximation by including or removing vertices and edges where required. After checking the multilevel scheme, if the limit is reached, any vertices which remain disconnected from the multilevel scheme are included using the **join** function, described in Figure 10.90 which forces a connection with any adjacent vertex, regardless of matching.

```

checkMatches( $v$ )
    if( $|v.par.children| = 1$ )
        for( $u \in \Gamma v$ )begin
            if( $|u.parent.children| = 1$ )
                 $u.parent.children.add(v)$ 
                 $v.parent := u.parent$ 
            end
        end
    end

```

Figure 10.86 Caretaker sub process - check whether any matches are available for any unmatched vertex


```

checkParents( $v, i$ )
  if( $v.parent = null$ )
     $match := false$ 
    for( $u \in \Gamma v$ ) begin
      if( $|u.parent.children| = 1$ )
         $u.parent.children.add(v)$ 
         $v.parent := u.parent$ 
         $match := true$ 
    end
  if( $match = false$ )
    create vertex in  $G_{i+1}$ 
     $w.children.add(v)$ 
     $v.parent = w$ 

```

Figure 10.87 Caretaker sub process - check if a vertex has been provided with a parent, if not, check any adjacent vertices for a parent with only one child, or create a new vertex in the next coarsest graph to act as a parent

```

checkGhostVertices( $v, i$ )
  if( $i > 0, |v.children| = 0$ )
    removeVertex( $v, G_i$ )

```

Figure 10.88 Caretaker sub process - identify any vertices in the coarse graphs which no longer represent vertices in the original, and remove them

In addition to the caretaker, a method is used to maintain or remove the preservation of edges through the multilevel scheme. Unlike the caretaker however, the process requires analysis of vertex neighbourhoods and therefore should only be run when required, on only the vertices involved, in order to keep required running time reduced. The process ensures edges are incorporated into the multilevel scheme, where required, or are removed if outdated (after vertex or edge removal for example).

```

preserveEdges( $v, u, \{\text{addition} \mid \text{remove}\}$ )
     $v = v.\text{parent}$ 
     $u = u.\text{parent}$ 

    if( $\text{addition} \ \& \ v \neq u$ )
        if( $e(v, u) \notin E$ )
             $\Gamma v.\text{add}(u)$ 
             $\Gamma u.\text{add}(v)$ 
            preserveEdges( $v, u, \text{addition}$ )

        if( $\text{remove}$ )
            if( $v \neq u$ )
                 $t = \text{false}$ 
                for( $w \in v.\text{children}$ ) begin
                    for( $r \in u.\text{children}$ ) begin
                        if( $w = r$ )
                             $t = \text{true}$ 
                        end
                    end
                end

                if( $t = \text{false}$ )
                     $v.\text{remove}(u)$ 
                     $u.\text{remove}(v)$ 
                    preserveEdges( $v, u, \text{remove}$ )

            end

        end

```

Figure 10.89 PreserveEdges function used to check the preservation or the removal of edges throughout the multilevel scheme

```

join( $G_i$ )
    for( $v \in G_i$ ) begin
        if( $v.\text{parent} = \text{null}$ )
            create vertex  $t := \Gamma v.\text{get}[0].\text{parent}$ 
             $v.\text{parent} := t;$ 
             $t.\text{children}.\text{add}(v)$ 
        end
    end

```

Figure 10.90 Join method for use with multilevel update methods, ensuring graph modifications are joined to the existing multilevel scheme

10.22.5 Summary

Implementation of the new algorithmic approaches into methods for experimentation and results collection is provided, with a description of each and inclusion of pseudo code examples. Overall, the following methods have been implemented:

- Data Structures and Input/Output method
- Multilevel Global Force (*MGF*)
 - Generation of Multilevel Scheme including Coarsening methods
 - Use of a Coarsening Limit
 - Use of Primitives, including identification and Vertex Placement Scheme
 - PostOrder update method to update approximation of vertex positions
 - Spring Embedder example using *MGF*
 - Use of an Approximation limit for comparison to Distance limit for repulsive forces
- Dynamic Graph Drawing Methods
 - Dynamic Spring Embedder Adaptation
 - Incorporation of Multilevel Layout Generation
 - Use of *MGF* alone and in combination with Multilevel Layout Generation
 - Operations
 - Matching Update Methods for Dynamic Matching in the Multilevel Scheme

10.23 Multilevel Aesthetics: Detailed Results

10.23.1 Comparison of G0, and GN and partition layout quality

10.23.1.1 Layouts for the Modified Spring Embedder without Multilevel Refinement

	edge crossings in G0	G0 vertices	edge crossings	GN vertices	vertices per partition in GN	edges per partition in GN	average edge crossings per partition	maximum edge crossings in a partition	minimum edge crossings in a partition
3025	10229.0	3025	123	40	76	149	41.7	152.0	0.0
3025	12202.0	3025	94	40	76	149	46.6	173.0	0.0
3025	11304.0	3025	161	40	76	149	42.1	166.0	0.0
3025	13394.0	3025	208	43	70	138	35.3	151.0	0.0
3025	12789.0	3025	121	39	78	152	51.7	200.0	0.0
3025	11983.6	3025	141	40	75	147	43.5	168.4	0.0

data	96119.0	2851	261	32	89	472	892.3	2880.0	0.0
data	102510.0	2851	315	31	92	487	970.2	2715.0	0.0
data	108293.0	2851	173	33	86	457	1078.9	2665.0	0.0
data	134667.0	2851	210	31	92	487	1164.6	2680.0	0.0
data	124954.0	2851	145	34	84	444	1009.4	2826.0	0.0
data	113308.6	2851	221	32	89	469	1023.1	2753.2	0.0
add32	30643.0	4960	194	92	54	103	85.7	365.0	0.0
add32	24605.0	4960	54	57	87	166	173.3	619.0	0.0
add32	27564.0	4960	99	64	78	148	158.5	561.0	0.0
add32	33034.0	4960	183	90	55	105	100.3	525.0	0.0
add32	26692.0	4960	101	66	75	143	130.9	729.0	0.0
add32	28507.6	4960	126	74	70	133	129.7	559.8	0.0
4elt	316400.0	15606	1335	108	145	425	278.1	1096.0	0.0
4elt	348508.0	15606	852	111	141	413	326.4	1030.0	0.0
4elt	280914.0	15606	863	111	141	413	321.0	971.0	0.0
4elt	259768.0	15606	851	101	155	454	307.8	1074.0	0.0
4elt	315810.0	15606	941	106	147	433	375.6	1335.0	0.0
4elt	304280.0	15606	968	107	145	428	321.8	1101.2	0.0
sierpinski10	633262.0	88575	1100	245	362	723	485.0	1272.0	0.0
sierpinski10	639251.0	88575	867	242	366	732	483.0	1884.0	0.0
sierpinski10	830746.0	88575	1072	243	365	729	547.2	1353.0	0.0
sierpinski10	730379.0	88575	1032	243	365	729	487.7	1431.0	0.0
sierpinski10	624948.0	88575	1097	245	362	723	460.8	1475.0	0.0
sierpinski10	691717.2	88575	1034	244	364	727	492.8	1483.0	0.0

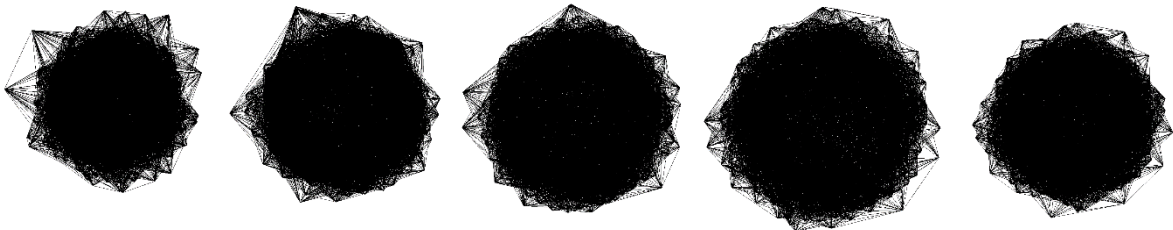
10.23.1.2 Layouts for Modified Spring Embedder with Multilevel Refinement

	edge crossings in G0	G0 vertices	edge crossings	GN vertices	vertices per partition in GN	edges per partition in GN	average edge crossings per partition	maximum edge crossings in a partition	minimum edge crossings in a partition
3025	0.0	3025	0	41	74	145	0.0	0.0	0.0
3025	0.0	3025	0	40	76	149	0.0	0.0	0.0
3025	2114.0	3025	48	41	74	145	8.6	107.0	0.0
3025	2053.0	3025	28	41	74	145	7.7	58.0	0.0
3025	0.0	3025	0	40	76	149	0.0	0.0	0.0
3025	833.4	3025	15	41	75	146	3.3	33.0	0.0
data	40253.0	2851	135	31	92	487	731.8	2594.0	0.0
data	39242.0	2851	61	32	89	472	678.2	1403.0	0.0
data	34328.0	2851	310	59	48	256	273.9	935.0	0.0
data	35035.0	2851	108	34	84	444	608.4	1619.0	0.0
data	46304.0	2851	63	32	89	472	800.8	2568.0	0.0
data	39032.4	2851	135	38	80	426	618.6	1823.8	0.0
add32	12761.0	4960	37	55	90	172	131.5	444.0	0.0

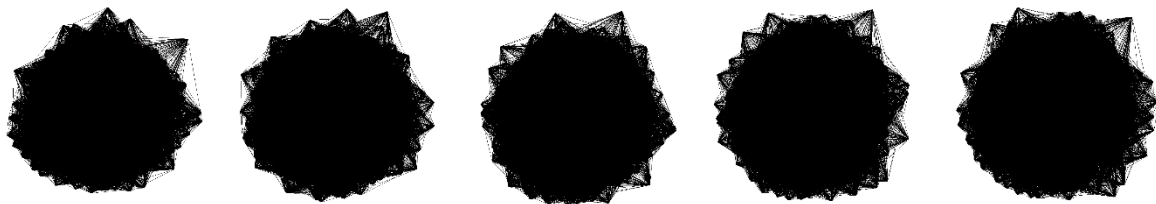
add32	12059.0	4960	42	59	84	160	124.4	408.0	0.0
add32	11547.0	4960	40	96	52	99	72.7	408.0	0.0
add32	10947.0	4960	17	58	86	163	129.5	390.0	0.0
add32	13013.0	4960	38	66	75	143	116.2	433.0	0.0
add32	12065.4	4960	35	67	77	147	114.9	416.6	0.0
4elt	18130.0	15606	75	108	145	425	32.7	346.0	0.0
4elt	18036.0	15606	116	110	142	417	30.1	361.0	0.0
4elt	20284.0	15606	108	111	141	413	41.5	591.0	0.0
4elt	22003.0	15606	162	111	141	413	32.7	472.0	0.0
4elt	31274.0	15606	169	109	143	421	36.1	523.0	0.0
4elt	21945.4	15606	126	110	142	418	34.6	458.6	0.0
sierpinski10	19246.0	88575	3	236	375	751	73.7	246.0	0.0
sierpinski10	18439.0	88575	0	248	357	714	65.2	202.0	0.0
sierpinski10	18591.0	88575	9	248	357	714	65.5	228.0	0.0
sierpinski10	19311.0	88575	20	245	362	723	68.6	210.0	0.0
sierpinski10	19759.0	88575	2	248	357	714	72.9	275.0	0.0
sierpinski10	19069.2	88575	7	245	362	723	69.2	232.2	0.0

10.24 Dynamic Spring Embedder: Development of Layouts

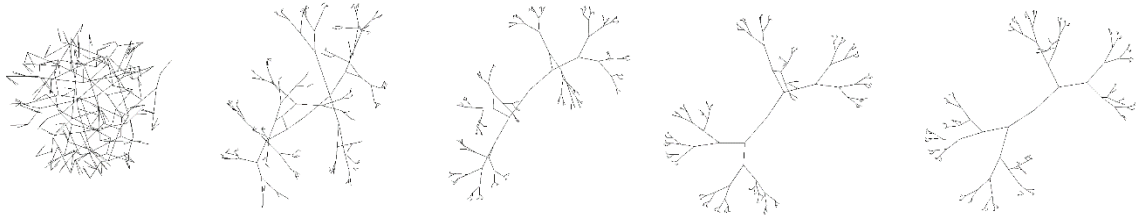
10.24.1 Layout Generation



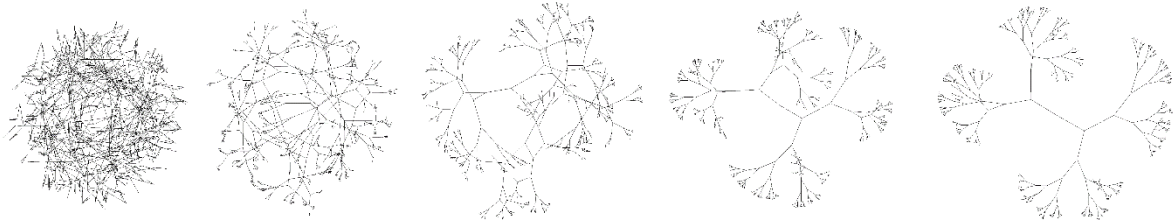
Dense small for 10, 50, 100, 200 and 300 frames



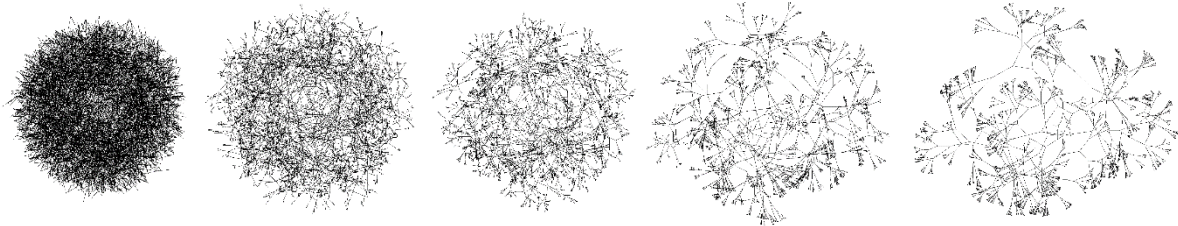
Dense Medium



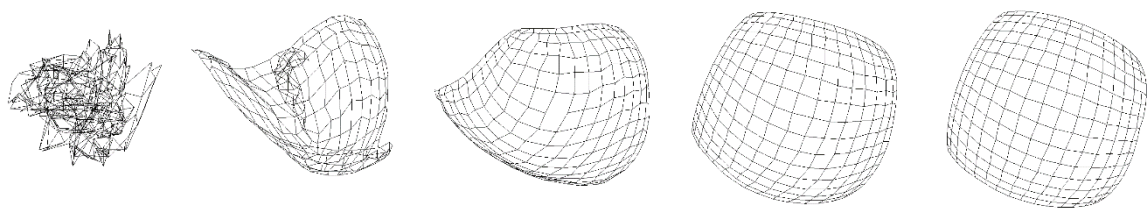
Sparse small for 10, 50, 100, 200 and 300 frames



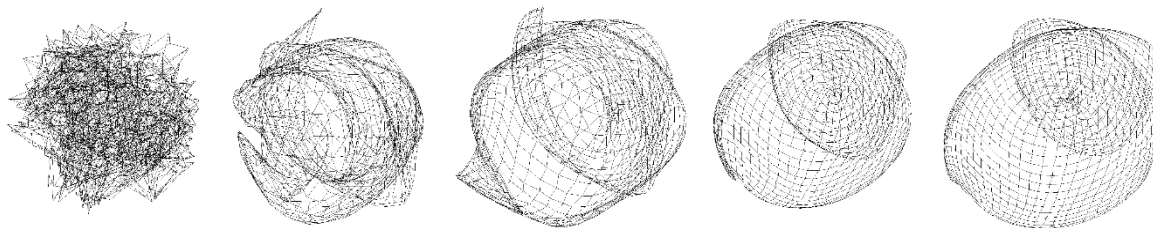
Sparse Medium



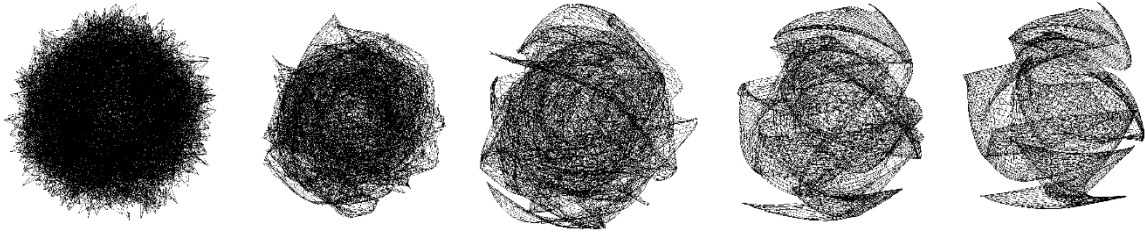
Sparse Large



Regular small for 10, 50, 100, 200 and 300 frames



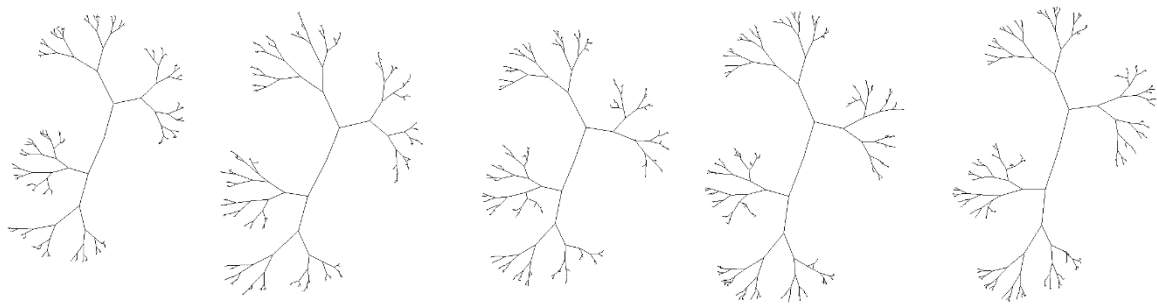
Regular Medium



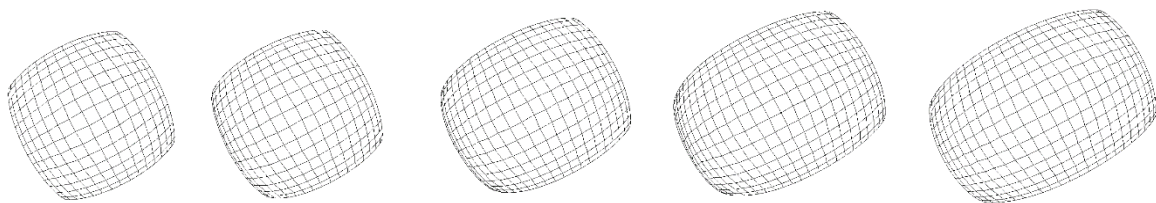
Regular Large

10.24.2 Layout Modification

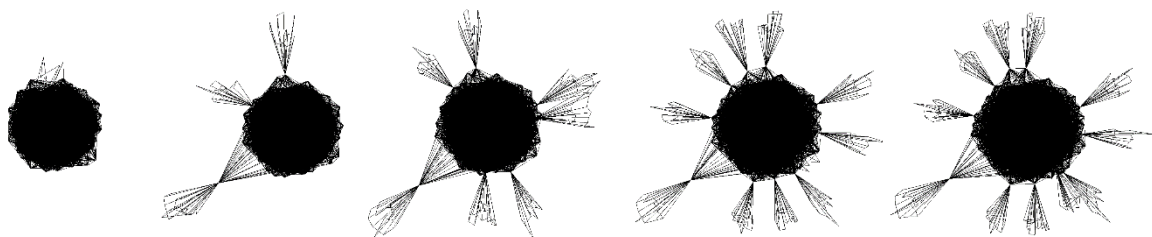
10.24.2.1 Growth



Sparse small showing changes to layout at frames 50, 100, 180, 250, and 300.

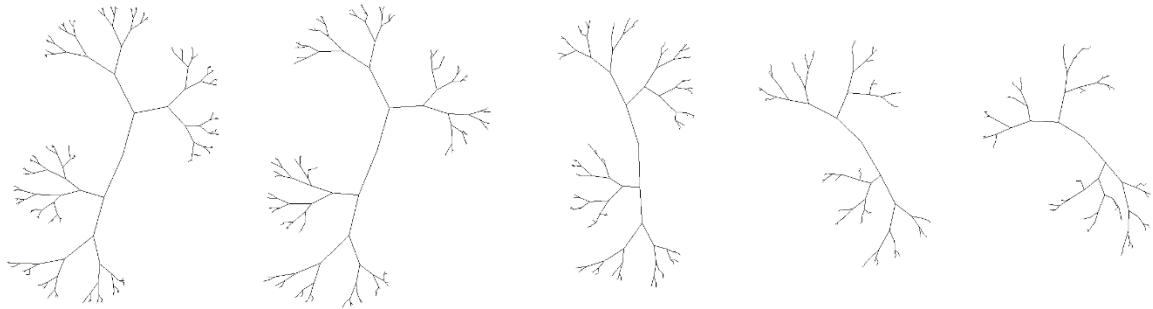


Regular small showing changes to layout at frames 50, 100, 180, 250, and 300.

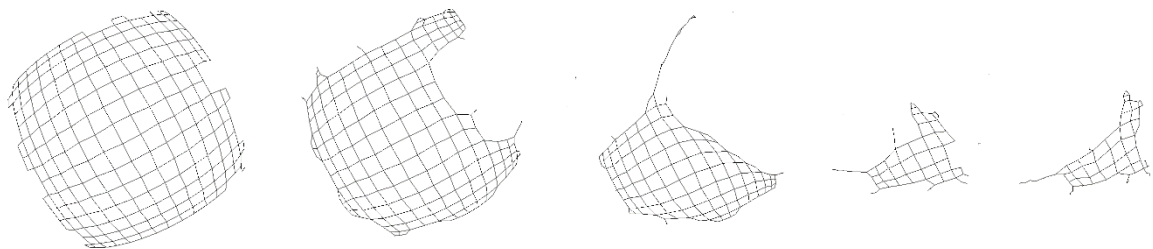


Dense small showing changes to layout at frames 50, 100, 180, 250, and 300.

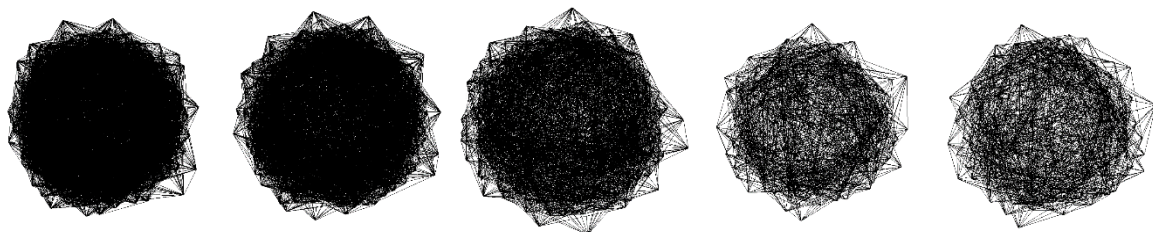
10.24.2.2 Shrink



Sparse small showing changes to layout at frames 50, 100, 180, 250, and 300.



Regular small showing changes to layout at frames 50, 100, 180, 250, and 300.



Dense small showing changes to layout at frames 50, 100, 180, 250, and 300.

10.25 Comparison of Dynamic Spring Embedders

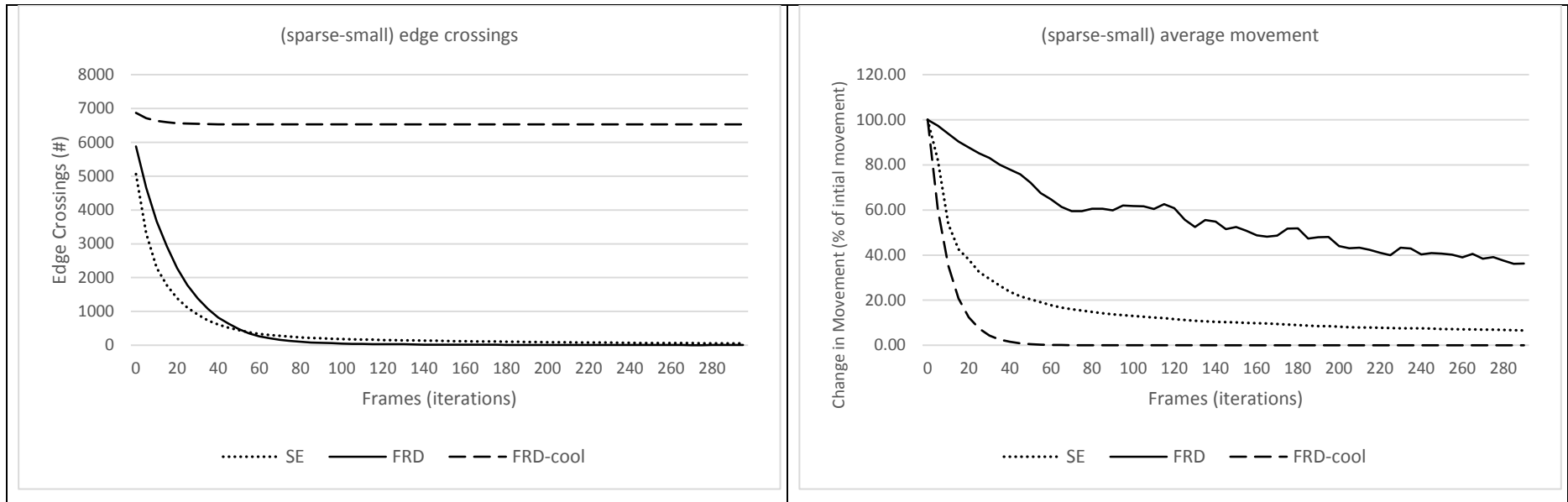
- *FR* – Standard Fruchterman and Reingold Modified Spring Embedder
- *SE*- Eades Spring Embedder
- *FRDC-cool* – Dynamic Spring Embedder using a cooling schedule like that used by *FR*
- *FRDC-nocool* – Dynamic Spring Embedder using no cooling schedule, instead using constants closer resembling *SE*
- *FRDC-YT* – Dynamic Spring Embedder using cooling values as suggested in correspondence with Darko (Appendix 10.3)

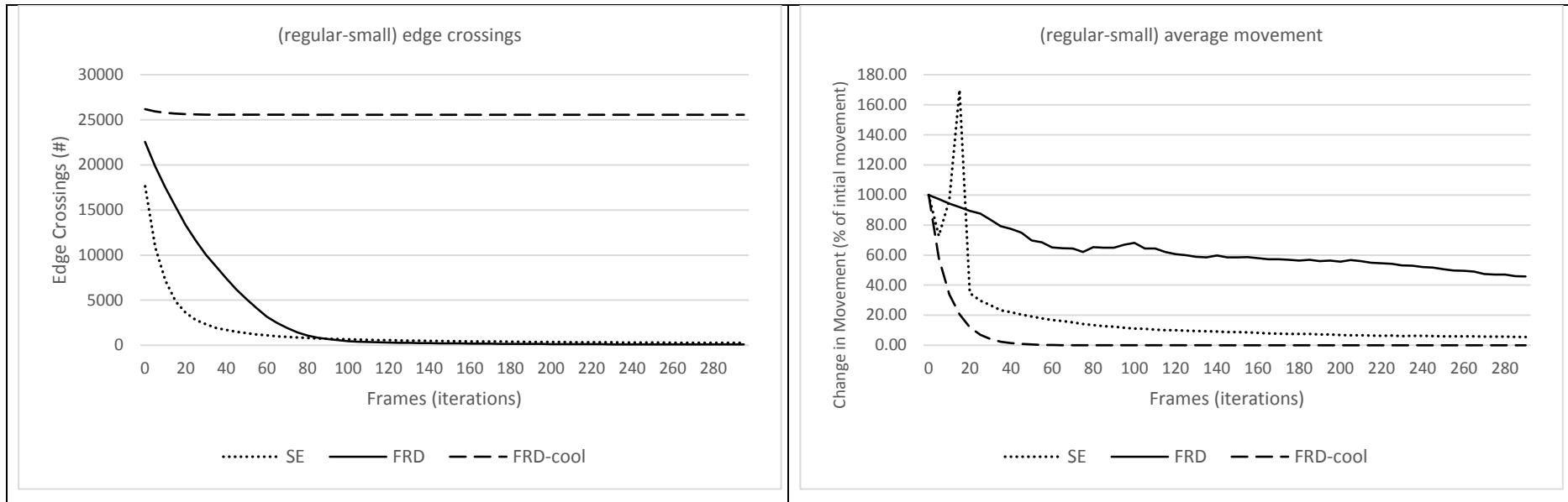
	runtime		
	dense	regular	sparse
<i>FR</i>	3702.4	2554.2	1668
<i>SE</i>	78137.4	3981	2197.9
<i>FRDC-cool</i>	3634.7	1688.2	1667
<i>FRDC-nocool</i>	3492.8	1567.9	1533.6
<i>FRDC-YT</i>	3589.9	1685	1674.3

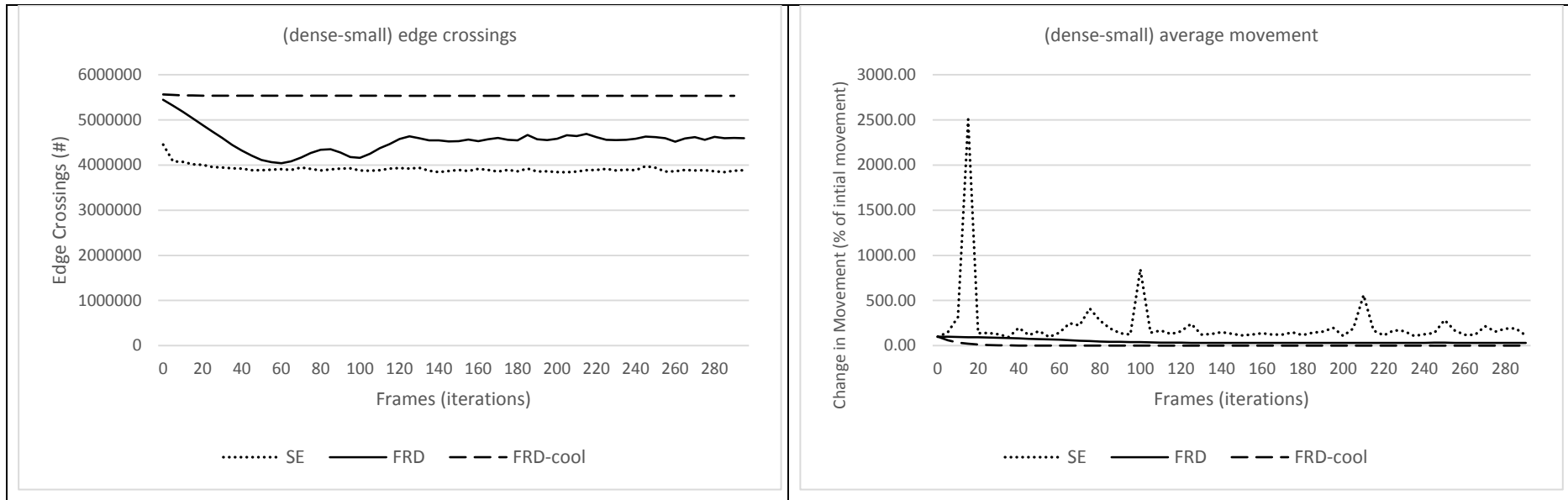
	edge crossings		
	dense	regular	sparse
<i>FR</i>	4816331	15929.4	3388.2
<i>SE</i>	3892754	244.4	49.1
<i>FRDC-cool</i>	5581766	26402.5	7030.7
<i>FRDC-nocool</i>	4613519	45.7	3.4
<i>FRDC-YT</i>	5621761	26497.3	7390.7

	edge range / average edge		
	dense	regular	sparse
<i>FR</i>	3.10969	3.06804	3.06245
<i>SE</i>	-	-	-
<i>FRDC-cool</i>	2.49206	2.23015	2.17592
<i>FRDC-nocool</i>	3.79919	1.54535	5.63744
<i>FRDC-YT</i>	2.43953	2.20350	2.17534

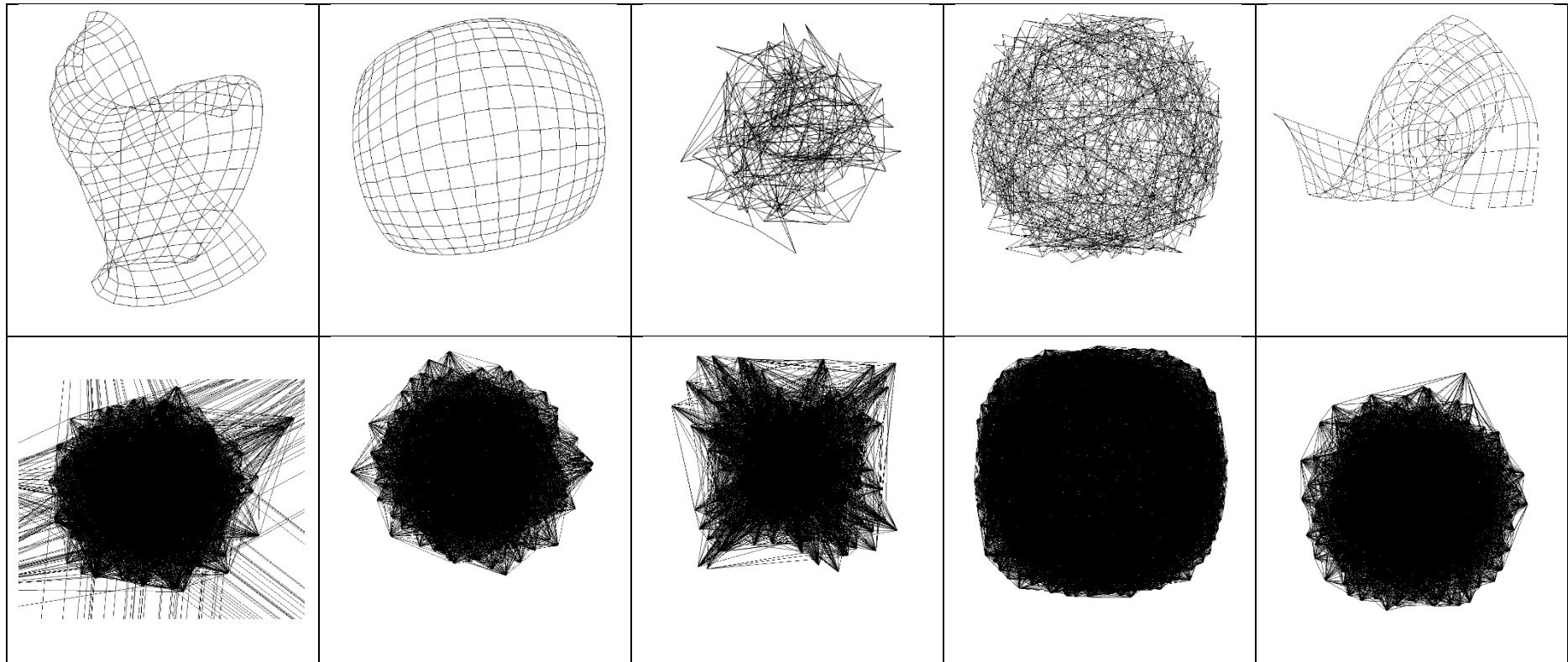
Graph Stability:







SE	FRD-nocool	FRD-cool	FRD-YT	FR



10.26 Comparison of Change in Edge Crossings for Dynamic Spring Embedder and Multilevel Dynamic Spring Embedder

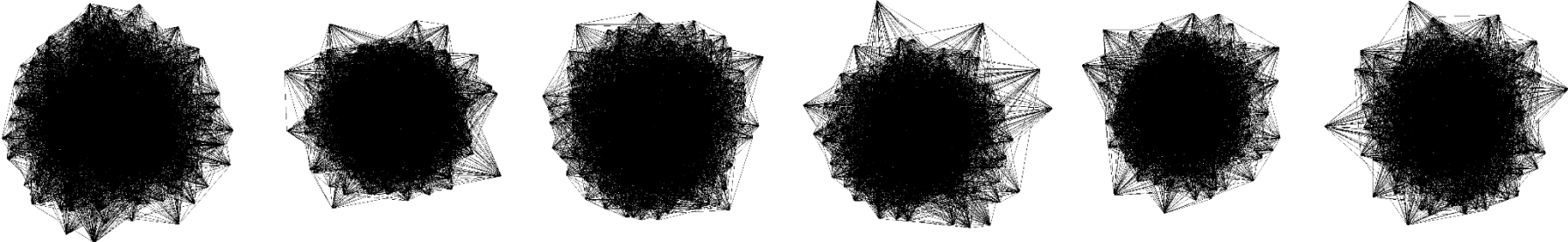
FRD: Dynamic Spring Embedder, *MLFRD*: Multilevel Dynamic Spring Embedder

regular-large		sparse-large		<i>dense-medium</i>		average	
<i>FRD</i>	<i>MLFRD</i>	<i>FRD</i>	<i>MLFRD</i>	<i>FRD</i>	<i>MLFRD</i>	<i>FRD</i>	<i>MLFRD</i>
100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
5.94	3.75	2.74	3.27	99.81	98.49	36.16	35.17
2.15	1.70	1.16	1.30	96.59	96.54	33.30	33.18
1.19	1.11	0.66	0.79	95.09	94.75	32.31	32.22
0.84	0.84	0.44	0.59	94.17	93.36	31.81	31.59
0.65	0.69	0.32	0.47	93.40	92.20	31.46	31.12
0.54	0.59	0.24	0.39	92.97	91.21	31.25	30.73
0.47	0.52	0.20	0.34	92.57	90.49	31.08	30.45
0.42	0.47	0.16	0.30	92.29	89.70	30.96	30.16
0.38	0.44	0.14	0.27	92.03	89.04	30.85	29.92
0.35	0.40	0.12	0.25	91.90	88.46	30.79	29.70
0.33	0.37	0.10	0.24	91.61	88.41	30.68	29.67
0.31	0.35	0.10	0.22	91.51	87.68	30.64	29.42
0.29	0.33	0.08	0.21	91.37	87.50	30.58	29.35
0.27	0.31	0.08	0.20	91.30	87.32	30.55	29.27
0.26	0.29	0.07	0.19	91.10	86.89	30.48	29.12
0.25	0.28	0.06	0.18	91.03	86.63	30.45	29.03
0.24	0.27	0.06	0.17	90.89	86.49	30.39	28.97
0.23	0.25	0.05	0.16	90.91	86.32	30.40	28.91
0.22	0.24	0.05	0.15	90.95	85.89	30.41	28.76
0.21	0.23	0.05	0.15	90.83	85.88	30.36	28.75
0.20	0.22	0.05	0.14	90.78	85.65	30.34	28.67
0.19	0.21	0.04	0.14	90.75	85.61	30.33	28.65
0.19	0.20	0.04	0.13	90.56	85.37	30.26	28.57
0.18	0.19	0.04	0.12	90.62	85.24	30.28	28.52
0.18	0.19	0.04	0.12	90.51	85.01	30.24	28.44

0.17	0.18	0.04	0.12	90.60	84.91	30.27	28.41
0.17	0.18	0.04	0.12	90.54	84.73	30.25	28.34
0.16	0.18	0.04	0.11	90.34	84.86	30.18	28.38
0.16	0.17	0.03	0.11	90.23	84.62	30.14	28.30
0.16	0.17	0.03	0.10	90.18	84.45	30.12	28.24

10.27 Multilevel Dynamic Spring Embedder: Development of Layouts

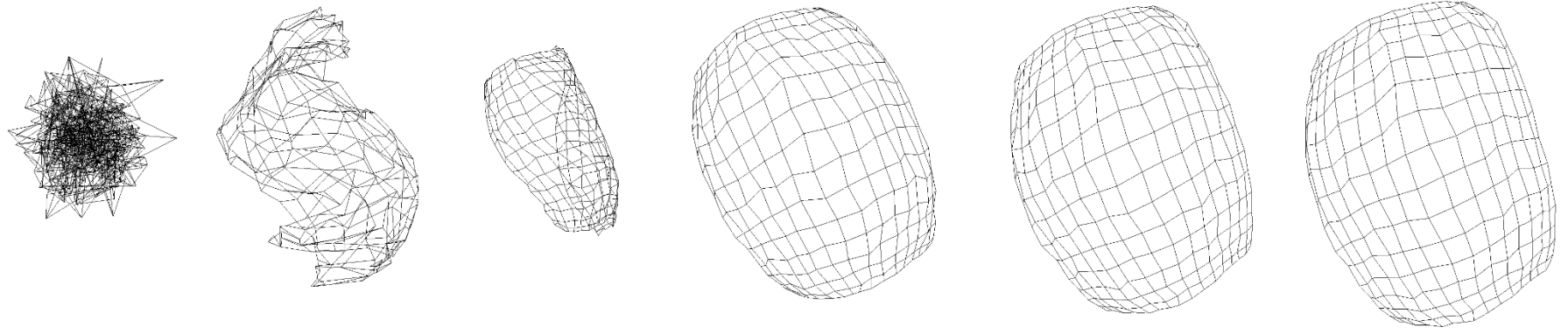
Dense Small



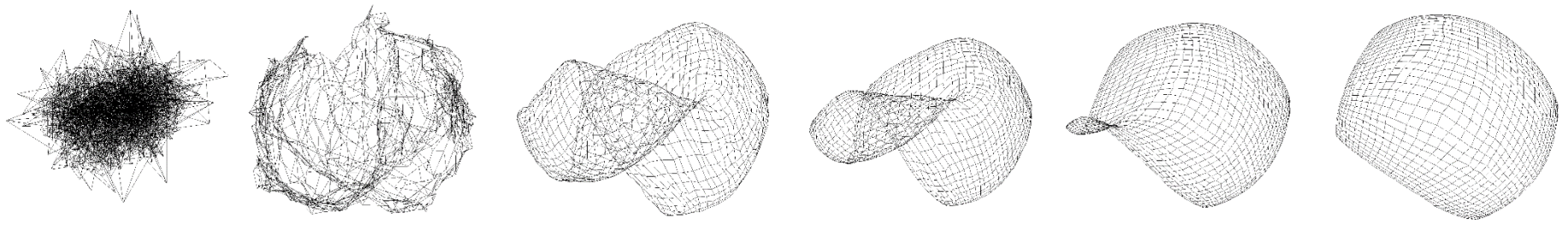
Dense Medium



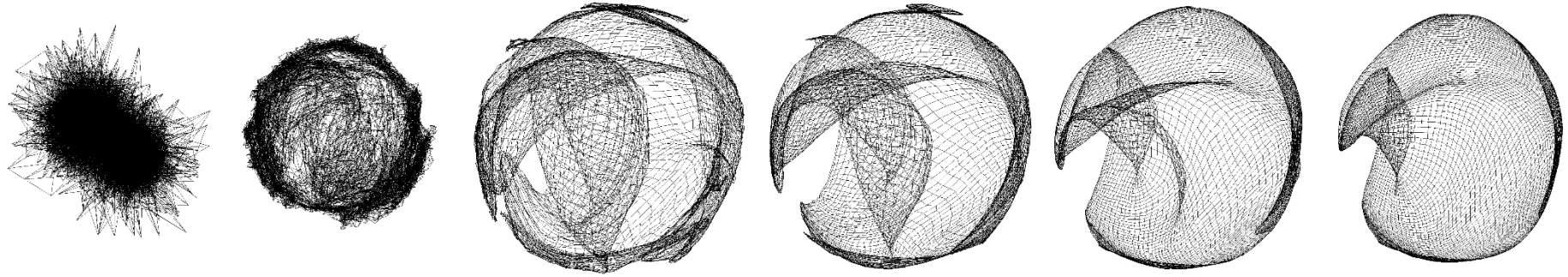
Regular Small



Regular Medium



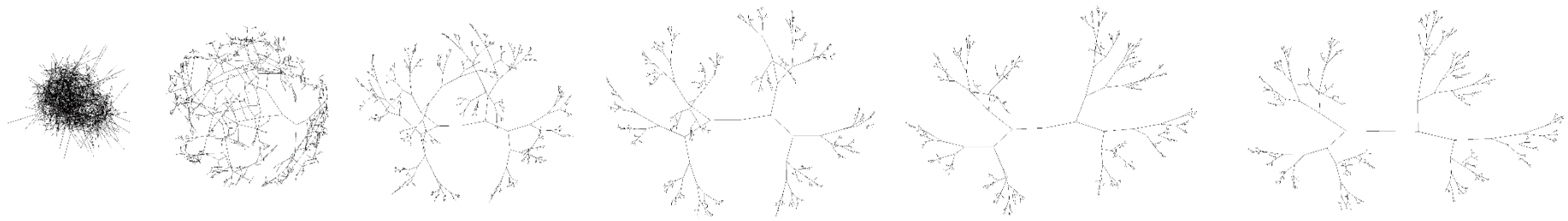
Regular Large



Sparse Small



Sparse Medium



Sparse Large

