University of New Mexico
UNM Digital Repository

Computer Science ETDs | Engineering ETDs

5-1-2016

Stochastic Methods for One-Sided Bipartite Crossing Minimization and its Variants

Tanya Jeffries

Follow this and additional works at: https://digitalrepository.unm.edu/cs_etds

Recommended Citation
Jeffries, Tanya. "Stochastic Methods for One-Sided Bipartite Crossing Minimization and its Variants." (2016).
https://digitalrepository.unm.edu/cs_etds/65

This Thesis is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Computer Science ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

www.manaraa.com

Tanya Jeffries
Candidate

Computer Science
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Thomas Hayes, Chairperson

Patrick Kelley

Terry Loring

www.manaraa.com

STOCHASTIC METHODS FOR ONE-SIDED BIPARTITE CROSSING MINIMIZATION AND ITS VARIANTS

by

TANYA JEFFRIES

PREVIOUS DEGREES
B.A. ECONOMICS, UNIVERSITY OF NEW MEXICO, 2010

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico
Albuquerque, New Mexico

May, 2016

www.manaraa.com

Acknowledgements

This work would not have been possible without the help of several people that I would like to mention here. First I would like to thank my advisor Thomas Hayes for his support throughout the entire process. His careful proofreading of the manuscript, suggestions and critiques, along with his support of my research interests helped me to create a work that I am truly proud of. Thank you Tom. I would also like to thank my family and friends. To my family: Mom, Dad, and my wonderful brother Tyler, you not only provided the words of encouragement to keep me motivated during my thesis work, but also had the patience to listen to me when I needed an audience for brainstorming (or post-code venting). Thank you! To my dearest friend Mabel, you have always been there for me through the hard times and the good. Thank you. And to my late friend Dorothy, you were not only one of my dearest friends but also an inspiration to me. You taught me to persist, to work hard no matter the circumstances, and to cherish every day. Thank you so much Dorothy.

And last but most certainly not least I would like to thank my animal friends! To Argos, our family dog, you have brought so much fun and happiness into our lives. Thank you for begging me for table scraps and the occasional belly and ear rub rub. To Penny, my dearest dog friend, your exuberance and playful antics continue to live on in my loving thoughts. The time that I met you marked the beginning of a new chapter in my life, and as my master's studies draw to an end, I remember the time we shared with much warmth and happiness. And finally to my dearest hedgehog Fiona. You were truly exceptional and I feel so blessed to have had you in my life. Thank you thank you and always remember that mommy loves you.

www.manaraa.com

Stochastic Methods for One-Sided Bipartite Crossing Minimization and its Variants

Tanya Jeffries

B.A., Economics, University of New Mexico, 2010

M.S., Computer Science, University of New Mexico, 2016

ABSTRACT

The one-sided bipartite graph drawing problem has been extensively studied in the graph drawing literature, with numerous papers appearing over the years showing novel algorithms and heuristics for minimizing associated edge crossings. Although stochastic methods have been highly successful when applied to bipartite graph drawing, a large, comprehensive study to compare said methods has not been carried out for one-sided crossing minimization on traditional, unweighted graphs. Even more so, the variant problems of weighted and bottleneck bipartite crossing minimization have seldom been studied, with only one published algorithm, 3-WOLF (designed for one-sided weighted crossing minimization), and one published heuristic, MEC (designed for multi-layer bottleneck crossing minimization) appearing in the literature. To date there has been no published work for handling one-sided bottleneck crossing minimization.

This thesis helps to close some of these gaps in the graph drawing literature. A comparative study of stochastic methods for one-sided bipartite graph drawing is carried out on both unweighted and weighted bipartite graphs. Novel variants of two previously published genetic algorithms, a one-sided version of a hybrid simulated annealing algorithm, a stochastic hill climbing procedure, and a top performing approximation algorithm, 3-WOLF, are compared against each other and the standard benchmark barycenter heuristic on weighted and unweighted bipartite graphs. Both unweighted and

www.manaraa.com

weighted versions of barycenter are considered. In addition, a novel bottleneck crossings based stochastic hill climbing procedure is compared against a one-sided variant of the MEC heuristic for one-sided bottleneck crossing minimization. The algorithms in this latter set of tests are conducted with and without barycenter preprocessing.

Stochastic hill climbing was found to obtain the best results for all of the bipartite graph drawing problems, significantly outperforming the other methods in the weighted and bottleneck crossing contexts. The results of bottleneck crossing based stochastic hill climbing were slightly improved with barycenter preprocessing. These experimental findings are particularly notable in the weighted and bottleneck crossing cases, since 3-WOLF is one of the best methods for weighted one-sided crossing minimization and MEC the best for multi-layer bottleneck crossing minimization. In regards to barycenter, although it didn't perform well for weighted crossing minimization, it was interesting to find that its traditional unweighted variant worked better for weighted crossing minimization than its edge-weighted version.

www.manaraa.com

Table of Contents

www.manaraa.com

www.manaraa.com

Chapter 1: Introduction

Graph Drawing is a branch of graph theory concerned with the automated layout of graphs. Algorithms that can clearly render the structural information of a graph are sought in numerous fields including network science, biology, cartography, and circuit design. General algorithms exist for visualizing arbitrary graphs (e.g. force-directed, spectral-based, and circular drawing methods), while others are tailored to specific graph structure (e.g. methods for the orthogonal layout of graphs and tree drawing algorithms). One such problem belonging to the latter class is that of bipartite graph drawing. In bipartite graph drawing the goal is generally to order the nodes of the partitioned vertex set on two parallel lines in a way that minimizes edge crossings. The problem is computationally hard so heuristics are often used to approximate good vertex orderings.

Over the years, researchers have explored different strategies for the problem, both randomized and exact. Thus far there have been no comprehensive studies comparing the numerous stochastic methods for bipartite drawing against each other and against benchmark heuristics. Even more so there has been very little experimentation in the closely related problem of weighted bipartite graph drawing, which incorporates edge weights into crossing calculations. To date there has been only one published study that addresses this problem [9]. It does so with a 3-approximation algorithm to the one-sided weighted crossing minimization problem, which the authors call 3-WOLF.

One of the goals of this thesis is to conduct comparative studies of stochastic methods for weighted and unweighted bipartite graph drawing. In the unweighted case

www.manaraa.com

multiple stochastic methods for one-sided bipartite drawing will be compared against each other and against the standard barycenter heuristic and an unweighted version of 3-WOLF on random bipartite graphs. In the second set of tests the same algorithms will be adapted so as to handle edge weights and will again be tested against each other and against barycenter. Both weighted and unweighted versions of the barycenter heuristic will be considered.

The other goal of this thesis is to give an initial look at the one-sided bottleneck crossing minimization problem, which to date has never been studied. So far only two papers, a technical report and a published article, have been produced to address the multi-layer bottleneck crossing problem. Out of these studies one heuristic, MEC, has experimentally proven to be the most dominant approach for the problem. In this last portion of the thesis the goal is to see if a one-sided variant of MEC still maintains dominance when compared to a stochastic hill climbing procedure that has been modified to handle bottleneck edge crossings. Both methods are also tested on graphs that have had barycenter preprocessing applied to them to initially minimize edge crossings.

Within the broad scope of these general thesis goals some specific questions/ideas addressed throughout this study are as follows:

1. Do linear arrangements serve as a good proxy for the one-sided bipartite crossing minimization problem? That is, if a method is made to minimize the total linear arrangement value of a bipartite graph assuming the coordinates on one side are fixed, will it also give a low number in terms of total edge crossings?

www.manaraa.com

2. Thus far few genetic algorithms have been published in the bipartite graph drawing literature, primarily because of their comparatively slow running times. If such genetic algorithms are modified to evolve solutions with low linear arrangements rather than crossings, will their running times improve enough to be competitive with other methods?
3. Only two genetic algorithms have been published for bipartite graph drawing, one by Mäkinen and Sieranta [21] and the other by Zoheir Ezziane [22]. Both compared their average crossing performance against the traditional one-sided barycenter heuristic, but only one, that of Mäkinen and Sieranta, was implemented to handle one-sided bipartite graph drawings. The algorithm of Ezziane allowed nodes within both layers of a bipartite graph to be permuted. If both algorithms evolved layouts with the same fitness measure (i.e. low linear arrangement values), and that of Zoheir was modified to handle only one free layer, how would the average performance of the two algorithms compare against each other?
4. A linear ordering based simulated annealing algorithm has successfully been applied to two-sided unweighted bipartite crossing minimization with positive results [18], in many cases outperforming the iterated barycenter heuristic. Will it achieve the same dominance for unweighted one-sided bipartite graph drawing? How about in the weighted scenario?

www.manaraa.com

5. Previous studies have opted to use a weighted variant of the barycenter heuristic for drawing weighted bipartite graphs. Does using the traditional unweighted barycenter heuristic perform any better, worse, or just about the same as its weighted variant for minimizing weighted edge crossings in bipartite graphs?

The remainder of the thesis proceeds as follows. In Chapter 2 background information explaining the unweighted and weighted bipartite drawing problems is provided along with some background discussion on genetic algorithms and their context in graph drawing, simulated annealing, and linear arrangements. A subsection is also devoted to explaining work that has been done for bottleneck crossing minimization. Chapter 3 describes previously published stochastic methods from the bipartite graph drawing literature, as well as previous work on weighted and bottleneck crossing minimizations. Chapter 4 follows with a description of the study's methods. This includes details regarding the generation of random graphs, the algorithms and heuristics chosen for testing, parameter values, and relevant implementation details. Chapter 5 summarizes the test results and Chapter 6 provides concluding remarks and ideas for future research. Tables providing full test results are provided at the end of the thesis in Appendix A.

www.manaraa.com

Chapter 2: Background

2.1 Bipartite Graph Drawing

A graph G consists of a vertex set V and edge set E connecting pairs of distinct vertices in V. The vertex set of a bipartite graph on n vertices can be partitioned into disjoint subsets V_1 and V_2. $V_1 \cup V_2 = V$, and every edge in G has one endpoint in V_1 and the other in V_2.

The visualization of a bipartite graph is often done in a layered fashion such that all nodes in V_1 and V_2 are assigned constant y coordinates of y_1 and y_2 respectively. Vertex subsets V_1 and V_2 form the so-called "layers" of the graph. Assignment of x coordinates is then determined by the particular algorithm or heuristic being used. The graph drawing solution most often sought is the one that minimizes total edge crossings. Fewer edge crossings lead to less clutter and make for easier studying of the graph. This is apparent in the following drawing which highlights a marked decrease in crossings after applying a one-sided crossing reduction strategy to the leftmost graph drawing.

Applications in which such crossing minimized drawings of bipartite graphs are sought occur in VLSI circuit layout design and in hierarchical graph drawing, a generalized problem in which graphs are drawn with 3 or more layers.

It is important to observe that the number of edge crossings depends not on the actual coordinates assigned to nodes, but rather on the ordering of nodes within their assigned layers. Thus if $|V_1| = m$ and $|V_2| = n$, distinct x coordinates from the sets $S_1 = \{1,2,...,m\}$ and $S_2 = \{1,2,...,n\}$ can be assigned to the vertices of layers 1 and 2 respectively. The

www.manaraa.com

problem of layered bipartite graph drawing then reduces to finding permutations of sets S_1 and S_2 that induce the fewest number of edge crossings. The layer permutation or crossing minimization problem leaves the coordinates of one layer fixed while permuting those of the other. In two-sided crossing minimization both layers are free to be altered allowing for the permutation of both layers.

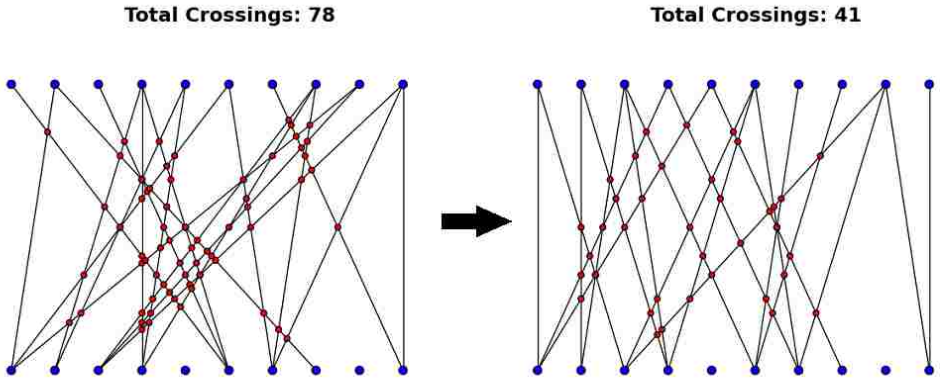

Figure 1: Reduction in edge crossings from drawings from left to right after application of one-sided bipartite drawing strategy.

Unfortunately, the bipartite drawing problem has been proven to be hard. Garey and Johnson proved the NP-completeness of the 2-layer permutation problem [4]. Even if the x coordinates of one layer are already assigned and just one layer is being permuted, Eades and Wormald found that the problem of minimizing edge crossings is still NP-Hard [5]. Given their computational intractability, algorithms and heuristics are sought that give good approximations to optimal crossing numbers. Numerous heuristics have been proposed over the years, a couple of the most popular being the so-called median

www.manaraa.com

and barycenter heuristics (a.k.a. the "averaging heuristics"). The barycenter heuristic of Sugiyama et. al. [6] is a one-sided permutation heuristic that calculates a node's x coordinate as the average of its neighbors' x coordinates in the fixed opposing layer. The median heuristic of Eades and Wormald [5] sets a node's x coordinate to be the median of its neighbors' x coordinates.

Both averaging heuristics are fast with running times of O(|E|). Assuming that V_2 is the layer being permuted, the median heuristic has the additional cost of $|V_2|\log|V_2|$ for the sorting part of median value calculation. Of the two heuristics the median-based has the better theoretical performance guarantees. Eades and Wormald [5] initially proved a performance ratio of 3 for the median heuristic and $\Theta(\min(\sqrt{n},n/\delta))$ for barycenter, where δ is the minimum node degree of the graph. Li and Stallman subsequently showed that even if the bipartite graph is connected (e.g. when ignoring isolated nodes) the performance ratio for barycenter is still $\Omega(\sqrt{n})$ [7]. They also showed that if the vertices of V_2 are bounded by degree d barycenter has a performance ratio of d-1 regardless of the graph's connectivity. However, despite said performance bounds, the one-sided barycenter heuristic was found to outperform many one-sided heuristics, including the median heuristic, on a variety of test graphs [8]. Poranen and Mäkinen suggest that this may be the due to the more frequent occurrence of ties when calculating medians versus averages [36] (i.e. it is much more likely for two nodes to have the same median than average when calculated over the x-coordinates of neighboring nodes). In the case of two sided crossing minimization (to be discussed next), they found through experiment that

www.manaraa.com

the average percentage of ties produced by iterated barycenter varied between 20.83 and 22.85 percent. This was smaller than the 38.87 to 42.65 percent average found for iterated median calculations.

One-sided median and barycenter heuristics can be extended to two sided crossing minimization through repeated and alternating application between the two layers of the bipartite graph. These two-sided variants form the so-called iterated barycenter and median heuristics. Unlike the one-sided cases no performance guarantees have been proven for iterated median and barycenter. However as with the one-sided case iterated barycenter was also shown through experiment to outperform other two-sided heuristics [8], and has indeed become a standard benchmark heuristic in testing the performance of two-sided crossing minimization methods.

In the related problem of weighted bipartite graph drawing, graphs with edges weights are also accounted for. In this generalized version of the problem the goal is to order the nodes of the graph on two parallel lines so as to minimize the sum of weighted crossings. Weighted crossings are calculated as the products of crossing edge weights. So for instance if two edges e_1 and e_2 with corresponding weights of w_1 and w_2 cross each other, their crossing weight of w_1*w will be added to the sum of weighted crossings. Counting crossings in an unweighted graph is thus equivalent to counting crossings in a weighted graph with uniform edge weights of 1.

www.manaraa.com

Besides being a generalization of unweighted bipartite graph drawing, the problem of weighted crossing minimization has seen application in problems specific to weighted bipartite graphs. Weighted bipartite crossing minimization has, for instance, seen application in recent research pertaining to VLSI layout design [41]. Weighted bipartite graph drawing has also arisen in the visualization of weighted layered dags (directed acyclic graphs). The edges in these scenarios have weights indicating some degree of 'importance', and so desirable layouts are the ones that avoid the crossing of edges with large weights. Applications in which this problem arises includes the visualization of metabolic pathways (subgraphs of the larger network of metabolic reactions) of different species for comparison [30], the layout of social networks [31], and the visualization of layered compound graphs in the context of biochemical pathways [32].

As with unweighted bipartite drawing, the edge weighted case yields two sub-problems, weighted one-sided and two-sided crossing minimization. Unlike the unweighted case the one and two sided permutation problems to weighted crossing minimization are much less studied. To the author's knowledge there are currently no published studies for two-sided weighted crossing minimization, and to date only one for one-sided crossing minimization [9]. As the authors of this study note, most studies that have used weighted bipartite graph drawing have chosen to apply the averaging heuristics or their weighted variants to minimize weighted crossings. This choice has not been based off of any experimental or theoretical studies, but rather on the intuition that

www.manaraa.com

heuristics that work well on unweighted bipartite graphs will also do so on weighted ones once they've been modified to account for edge weights.

2.2 Genetic Algorithmic Graph Drawing:

Genetic algorithms are stochastic search procedures that mimic the processes of natural selection. First introduced by John Holland in the mid-70s, a genetic algorithm generally proceeds as follows. Given some initial population of candidate solutions it applies modifying genetic operations to its members, creating a new population of viable solutions in the process. Each member of the new and previous populations is measured by some function to determine how fit it is in terms of solving the original problem. Following fitness value calculation, a selection rule/policy is applied to sift out fitter members for the next generation/iteration of the algorithm. The goal is to design the algorithm so that it evolves individuals that progressively get fitter, on the average. After some termination condition has been reached, the fittest individual with the best approximation to an optimal solution is selected. The specific components of a genetic algorithm are as follows:

- **chromosome representation:** a way of representing the "genetic" information of each candidate solution.
- **fitness evaluation function:** a function measuring how fit an individual is according to some predetermined measure of fitness.

www.manaraa.com

- **crossover operator:** given two "parent" individuals, combines segments of information from the parents' genomes to create child chromosomes.
- **mutation operator:** modifies the chromosome of a single individual with some predetermined probability.
- **selection policy:** given a population of individuals determines which ones will be considered in subsequent generations/iterations of the algorithm (e.g. elitist selection always selects the fittest member to be included in the subsequent generation)
- **stopping condition:** determines when the algorithm stops, at which point the final fittest population member is selected as output. For example, a genetic algorithm may stop after some predetermined number of iterations have passed or until a certain amount of time has passed with little to no improvement in the average fitness value.

The genetic algorithmic approach has been applied to several different graph drawing problems. They have been applied for drawing undirected graphs [12][13], for orthogonal graph drawing [14][15], for the 2 page book drawing problem [16], and for bipartite graph drawing, the focus of our current study. In all cases the chromosomes represent coordinate assignments for nodes, with the genetic operators designed to perform basic modifications to these coordinates (e.g. swapping of two nodes' x and/or y coordinates or movements of entire subgraph regions to other areas of the drawing area). The hope is

www.manaraa.com

that the genetic algorithm will eventually evolve a layout that satisfies certain aesthetic criteria. Examples include minimization of total edge length, even distribution of nodes, uniformity of edge length, minimization of drawing area and minimization of edge crossings, the latter being the most commonly elected criteria for fitness evaluation. Despite their successful application to graph drawing, the previously published genetic graph drawing algorithms (and genetic algorithms in general) suffer from a slow running time. In the case of graph drawing genetic algorithms take the most time when calculating edge crossings for fitness measurements. Naively the crossing count of a graph drawing can be calculated in time ($|E|^2$) which is quite slow. In the specialized case of a bipartite graph edge crossings can be calculated in time $O(\min\{n_1n_2, |E|\log(\min\{n_1, n_2\})\})$[17], which although is much faster than the naive approach, can still cumulatively add to the total costs of a genetic algorithm which will apply the algorithm to entire populations over the course of many generations. Comparative studies for bipartite drawing algorithms have thus far avoided evaluating previously published genetic algorithms due to their slow running times.

2.3 Linear Arrangements:

In the so-called linear arrangement problem, the goal is to plot a graph's vertices along a horizontal line so as to minimize the sum of edge lengths. The vertices are assigned unique x coordinates so as to avoid the trivial solution of placing all nodes on the same point. Like the two-sided bipartite crossing minimization problem, the linear arrangement problem is NP-Hard[1].

www.manaraa.com

Shahrokhi et. al. found an interesting connection between the linear arrangement problem and the two-sided bipartite crossing minimization problem [2]. As a result, bipartite drawings with lower edge length sums coincide with bipartite drawings with fewer crossings. Thus a good approximation to the linear arrangement problem should also give a good approximation to the two-sided bipartite drawing problem. One way of approximating the optimal linear arrangement of a graph is by utilizing its so-called Fiedler vector. This vector corresponds to the first nonzero eigenvalue of the graph's Laplacian matrix L = D - A. The matrix A corresponds to the graph's traditional adjacency matrix and D is a diagonal matrix containing the graph's degree sequence along its main diagonal.

Martin Juvan and Bojan Mohar were the first to establish a relation between a graph's Fiedler vector and linear arrangement cost [3]. Theoretically they showed that if there is not much difference in successive values of a graph's Fiedler vector then it will provide a good proxy to its optimal linear arrangement. They also showed through experiment on special classes of graphs (e.g. cycles, paths, complete bipartite graphs, complete graphs and their cartesian products) for which the minimum linear arrangement value is known, that the spectral Fiedler based ordering generally yielded close to optimal linear arrangement values. Newton et. al. followed with an experimental study comparing Juvan and Mohar's spectral algorithm for linear arrangements with the multi-scale linear arrangement algorithm of Yehuda and Koren and the iterated barycenter heuristic [20]. They also achieved favorable outcomes, finding that both linear arrangement techniques

www.manaraa.com

tended to outperform iterated barycenter for the two-sided crossing minimization problem.

Extensions of these ideas have been explored in the bipartite graph drawing literature. One such algorithm to be explored in section 3 combines spectral-based methods for linear arrangements with simulated annealing techniques.

2.4 Simulated Annealing

Simulated annealing is a randomized search process that, like the genetic algorithmic approach, aims to approximate an optimal solution to an objective function over a large search space. The approach has been applied to a wide range of combinatorial optimization problems, including the traveling salesman problem [33], the solving of sudoku puzzles [34], the matching of fingerprints [35], and bipartite graph drawing [18]. It models its behavior after the physical process of heating and then slowly cooling material to gradually minimize its defects and overall thermodynamic energy. The algorithm starts with an initial temperature T_0 and an approximate solution $s = s_0$ to an objective function f to be minimized. The current temperature T of the system (initially set to T_0) is gradually lowered at the end of each iteration, usually through multiplication with a positive constant less than one. The algorithm will proceed with this so-called annealing schedule until some stopping condition has been reached (e.g. T has dropped below some preset threshold temperature).

www.manaraa.com

During each iteration a randomly chosen and closely related neighbor s' to the current solution s is considered. If $f(s') \leq f(s)$, then the current solution s is set to s'. Otherwise s is set to s' with some probability that is a function of s, s', and the current temperature T. Often times the function $e^{(f(s)-f(s'))/T}$ is used. By allowing solutions to be explored that are worse than the current one, the process helps to avoid the problem of getting stuck in local optima, something that is often encountered in other stochastic methods such as genetic algorithms.

The annealing schedule and temperature help to regulate the acceptance of worse solutions throughout the course of the algorithm. At the very beginning when T is at its highest, the probability of selecting a worse solution is higher, resulting in a larger net of potential neighbors to explore. As the temperature cools down however, and T decreases, the same probability of acceptance also decreases. Until the stopping condition is reached, this gradually makes the search process of the algorithm more and more restrictive.

2.5 Weighted Crossing Minimization

Cakiroglu et. al. devised a 3 approximation algorithm for weighted one-sided crossing minimization, termed 3-WOLF (WOLF = weighted one layer free). The algorithm consists of a coarse phase that partitions nodes into disjoint sets/partitions P_r based on weighted edge sums (Algorithm 1), and a recursive refining phase that improves the edge crossings within partitions (Algorithm 2) and maintains a special invariant, PISP

www.manaraa.com

(partition invariant satisfying permutation), defined by the authors [9] that is associated with each of the partitions from Algorithm 1.

Algorithm 1: Coarse Ordering

/*decides which partition each node belongs to*/

/*1, 2, ...n_1 represent the nodes of V_1 ordered from left to right according to x-coordinate assignment*/

/*w(xy) = weight of edge between nodes x and y; zero if no such edge exists*/

for all x $\in V_2$ **do**:

leftsum = 0, rightsum = $\sum_{y=2}^{n1} w(xy)$

for r = 0 to n_1 - 1 **do**:

if (leftsum $\leq$ rightsum) **then** break;

leftsum = leftsum + $\sum_{y=r+1}^{r+1} w(xy)$

rightsum = rightsum - $\sum_{y=r+2}^{r+2} w(xy)$

P_r.append(x);

/*order the partitions*/

order the partitions as P_0, P_1,.....P_{n1-1}

Figure 2

www.manaraa.com

Algorithm 2: Fine-Grained Ordering

/*nodes in P_r have nonzero degrees*/

/*constructs a permutation (P_r) for P_r*/

/*initially $\Pi(P_r)$ is empty*/

if $|P_r| = 1$ **then**:

$\Pi(P_r)$.append(u) where $u \epsilon P_r$

else:

Divide P_r into equal sized partition P_{r1} and P_{r2}

/*solve for each subpartition*/

$\Pi(P_{r1})$ = Fine-Grained Ordering(P_{r1})

$\Pi(P_{r2})$ = Fine-Grained Ordering(P_{r2})

/*merge the resulting permutations*/

/*append a new node 'a' to the end of each permutation*/

/* $\sum_{y=1}^{r} w(ay) = -1$ and $\sum_{y=r+1}^{n1} w(ay) = 0$ */

$\Pi(P_{r1}) = \Pi(P_{r1})$.append('a')

$\Pi(P_{r2}) = \Pi(P_{r2})$.append('a')

www.manaraa.com

/*let u,v be the first nodes in $\Pi(P_{r1})$ and $\Pi(P_{r2})$ respectively*/

for i = 1 to $|P_r|$ **do**:

if $\sum_{y=1}^{r} w(vy) \times \sum_{y=r+1}^{n1} w(uy) \leq \sum_{y=1}^{r} w(uy) \times \sum_{y=r+1}^{n1} w(vy)$ **then**:

$\Pi(P_r)$.append(u)

u = u' where u' follows u in $\Pi(P_{r1})$

else:

$\Pi(P_r)$.append(v)

v = v' where v' follows v in $\Pi(P_{r2})$

Figure 3

The authors obtain the theoretic bound of $O(|E| + |V_1| + |V_2|\log(|V_2|))$ for the running time of 3-WOLF. To gauge its experimental performance they compared its time and output against simple weighted versions of common one-sided heuristics. The heuristics considered in these tests were weighted versions of the median heuristic, barycenter heuristic, GRE heuristic [10], and the penalty minimization scheme of Demestrescu and Finocchi [11].

As the authors note, applications that require minimization of weighted crossings usually employ weighted versions of the common averaging heuristics (i.e. median or barycenter-based methods) or of the penalty-graph based approach [11]. The intuitive reasoning behind this approach is that such simple edge-weighted extensions to these successful methods will perform well in the context of weighted crossing minimization.

www.manaraa.com

Interestingly, Cakiroglu et. al. found that this was not the case in regards to the average-based heuristics, with their deterministic 3-WOLF algorithm outperforming weighted versions of both the median and barycenter heuristics. This was particularly the case on random weighted bipartite graphs with edge density greater than or equal to 0.1. In fact, the variant of barycenter that accounted for edge weights in the averaging sums was found to perform worst in terms of crossings out of all of the algorithms. On graphs with densities between 10% and 50% edge density it ranged between 22 and 36 percent above the trivial lower bound for edge crossings (see section 4.1). This was considerably higher than the average results of the other algorithms with results ranging between 0.25 and 0.7 above the lower bound. The top 3 performing algorithms were 3-WOLF and weighted variants of the GRE heuristic [10] and the penalty minimization-based method [11], with 3-WOLF being the fastest and giving very similar results to the other two methods.

2.6 Bottleneck Crossing Minimization

The so-called 'bottleneck' crossing minimization problem presents another variant to bipartite and multi-partite graph drawing. Rather than minimize the total number of edge crossings, the goal is to minimize the maximum number of crossings occurring on any edge of the graph. If we let $c(E)_A$ represent the maximum number of crossings occurring on any single edge of a graph for a particular node arrangement A, the goal is then to find an A such that $c(E)_A$ is minimum. The optimal arrangements A can then be described as the following: $\{A \mid c(E)_A \text{ is minimum}\}$.

www.manaraa.com

Permuting the nodes by layer so as to achieve the optimal bottleneck value minimizes the deleterious effects of crosstalk in VLSI circuits, a problem discussed by Bhatt and Leighton [26]. Crosstalk in this context is a phenomenon by which a logic transmitted into a circuit creates undesired effects on neighboring circuits. Stallman and Gupta describe another application of bottleneck crossing minimization that involves being able to view subgraphs that have been zoomed in on and show relatively few edge crossings [27].

As with weighted bipartite graph drawing, bottleneck crossing minimization has seldom been studied in the graph drawing literature, with only one technical report and published article to date, both on multi-layered bottleneck crossing minimization [27]. According to Stallman, the bottleneck crossing minimization problem can easily be proven to be NP-Hard by modifying the unweighted two-sided proof of Garey and Johnson [4], using the bandwidth problem [1] in place of minimum linear arrangement. To our knowledge there has not been any published work thus far on one-sided bottleneck crossing minimization for bipartite graphs.

In his paper [29], Stallman presents a heuristic outlined previously by Stallman and Gupta [27] for multi-layered bottleneck crossing minimization. The input is a graph consisting of nodes that have been partitioned into layers. The components of the procedure are outlined in Algorithms 3 and 4 below.

www.manaraa.com

Termed MEC (Maximum Edge Crossings), the heuristic works by iteratively selecting 'unhandled' nodes incident to bottleneck edges and sifting them through their respective layer, at each step attempting to locally optimize the bottleneck crossings. An initial ordering on the nodes is assumed and each layer is free to permute. V(G) and E(G) stand for the vertex and edge sets of the graph respectively, c(e) equals the number of times edge e crosses with all of the other edges in the graph, and c*(E) represents the bottleneck crossing value for the edge set as a whole. If E' is a subset of E then c*(E') represents the bottleneck value over the subgraph induced by the edges in E' only. For a given vertex v, layer(v) refers to the layer of nodes that v belongs to, and p(v) to the position of node v within its layer.

Algorithm 3: MEC

initialize all nodes v ∈V(G) as unhandled.

while there is an edge with at least one unhandled endpoint **do**:

let e = vw be an edge with at least one unhandled endpoint s.t. c(e) = c*(E)

if v is not marked **then** EDGESIFT(v)

update c(f) for f ∈E(layer(v)) and mark v **endif**

if w is not marked *then* EDGESIFT(w)

update c(f) for f ∈E(layer(w)) and mark w **endif**

end do

Figure 4

www.manaraa.com

Algorithm 4: EDGESIFT(v)

let $y_1, y_2, \ldots, y_k$ be the nodes of layer(x) sorted by position

/*local variable *local_bottleneck* initialized to infinity*/

/*maintains the minimum local bottleneck crossing value found so far*/

local_bottleneck = ∞

/*local variable bottleneck_coord initialized to infinity*/

/*maintains the x-coordinate at which v is placed to minimize *local_bottleneck**/

bottleneck_coord = ∞

for i = p(v)-1 downto 1 **do**:

swap the x-coordinates of v and y_i

update c(e) for e ∈(E(v)∪E(y_i))

let c_i = c*(E(v)∪E(y_i))

if c_i < *local_bottleneck* then *let local_bottleneck* = c_i and *bottleneck_coord* = i

end do

for i = 2 to k **do**:

swap the x-coordinates of v and y_i

update c(e) for e∈ (E(v)∪E(y_i))

let c_i = c*(E(v)∪E(y_i))

if c_i < *local_bottleneck* then *let local_bottleneck* = c_i and *bottleneck_coord* = i

end do

if *bottleneck_coord* < p(v) **then** move v before y_o

else if *bottleneck_coord* > p(v) then move v after y_o

Figure 5

www.manaraa.com

During each pass of EDGESIFT, the starting node v (selected if the endpoint of a bottleneck edge) is swapped iteratively with neighboring nodes of the same layer, first swapping positions with nodes to its left until reaching the leftmost end of the layer. v is then moved back all the way to the right by swapping positions with rightmost neighbors of the same layer. All the while the current minimum local bottleneck value and associated position of v is maintained. After each swap the crossing values of the all edges in the graph are reevaluated. However, only the edges incident to the two nodes being swapped are evaluated for crossings. If the bottleneck value of the restricted edge set is smaller than the current minimum, then the local bottleneck value is reset to equal that of the current and the new and resulting placement of v is noted.

Since only the crossing values of the edges incident to v and the current neighbor y_i it is being swapped with are being evaluated for bottleneck crossing minimization, it is entirely possible that the position of p of v output by EDGESIFT may in fact result in a bottleneck value for the graph as a whole that is not globally optimal. Stallman and Gupta provide an example that illustrates this case (cite Stallman and Gupta report)

Figures 6 and 7 depict one swap movement during a pass of the sub-procedure. Here we assume that nodes 1 through 4 have x coordinates of 1, 2, 3 and 4 respectively. The nodes 5 through 8 of layer 2 also have corresponding x coordinates of 1, 2, 3, and 4. Assuming that node 7 has been input into EDGESIFT (since edge {2,7} has a bottleneck crossing value of 2), the local bottleneck value and position p are initially 2 and 3 respectively within the sub-procedure. Node 7 is then swapped with 6, the node to its left

www.manaraa.com

and the crossings are reevaluated for the affected edges. Out of the edges incident to nodes 6 and 7, the one with the largest crossing value is still {2,7} with a local bottleneck value of 1. The local bottleneck value and position are now reset to 1 and 2 for EDGESIFT.

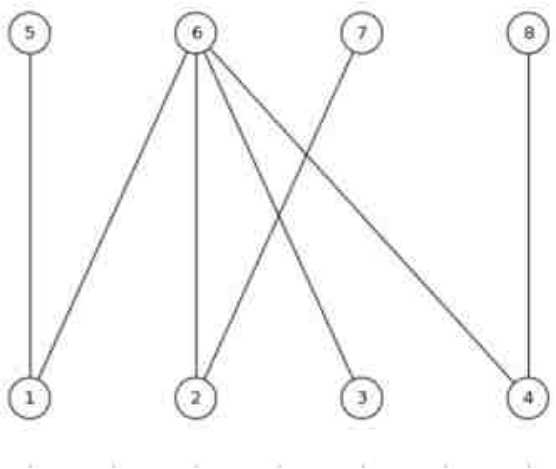

Figure 6: Initial local bottleneck value for EDGESIFT is 2 (the crossing number of edge {2,7}). The associated position of node 7 is p = 3.

www.manaraa.com

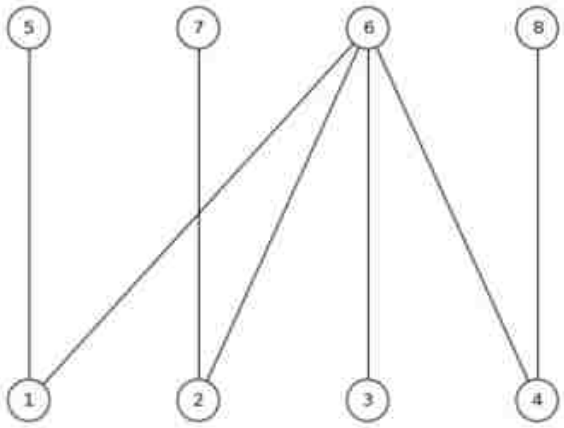

Figure 7: After swapping node 7 with 6, the bottleneck crossing value of the edges incident to nodes 7 and 6 has decreased to 1. The local bottleneck value and position are updated to 1 and 2 respectively.

Stallman and Gupta tested the performance of MEC against the barycenter heuristic for bottleneck crossing minimization in multi-layered graphs [27]. Their tests were performed on a variety of directed acyclic graphs, including random trees and random multipartite graphs (dummy nodes were added to the graphs to ensure that edges only ran between adjacent layers). They consistently found MEC to beat barycenter when it came to bottleneck crossings, with results for each graph class presented as the ratio of the minimum achieved by barycenter to the minimum achieved by MEC. For high density dags the ratio was consistently around 1.21, while for sparse dags it was around 2.00. When the densest and sparsest dags were ignored the ratio was roughly around 1.60.

www.manaraa.com

Chapter 3: Stochastic Methods for Bipartite Crossing Minimization and its Variants

3.1 Genetic Algorithms

Throughout the following discussion we will be making frequent use of the term "permutation". When we talk about the permutation of a layer we are referring to an array indexed from 1 in where the indices are x-coordinates and the values the correspondingly placed nodes. So for instance if for layer 1 we have the permutation [4,2,1,3], this means that node 4 has an x-coordinate of 1, node 2 an x-coordinate of 2, and so on.

Mäkinen and Sieranta were the first to devise a genetic algorithm for bipartite graphs [21], specifically for the one-sided crossing minimization problem. Their method employs both crossover and mutation operators along with an elitist selection policy. The elitist policy ensures that the permutation with the fewest edge crossings carries over to the next generation. Following this, the remaining population of the previous generation splits into two subpopulations A and B. These consist of the permutations that are more and less fit than the current population average. Remaining members of the next generation are generated by iteratively crossing over randomly selected members of A and B with each other.

Given two parent permutations $[x_0,...,x_{n-1}]$ and $[y_0,...,y_{n-1}]$ of the free layer, their crossover operator produces two child offspring $[x_0',...,x_{n-1}']$ and $[y_0',....,y_{n-1}']$ consisting of portions of each parent's permutation information. Starting with the random generation

www.manaraa.com

of indices i and j such that i < j and 1 i,j n, one child is formed by inheriting the segment $[x_i,...,x_j]$ from the first parent. The other child inherits $[y_i,...,y_j]$ from the second parent. Both segments will continue to occupy positions i through j in the respective children. Missing elements from each permutation are then added back to the children in the order they appear in the opposing parent (i.e. the parent that the child did not inherit from). As an example, suppose that two parents x = [4,3,1,2] and y = [3,1,2,4] are being crossed over with each other in this setting, with random indices of i = 1 and j = 2 generated at the beginning. The resulting children of the crossover will be [2,3,1,4] and [4,1,2,3]. The first child inherits the segment [3,1] from the parent x, and is then augmented by elements 2 and 4 according to their order in parent y. Similarly, child two inherits [1,2] from parent y, with elements 3 and 4 appended to it in the order that they appear in y.

As for their mutation operator, it essentially entails the randomized relocation of an element within its permutation followed by a right or leftward shifting of elements. For each number in the permutation, a random number p is generated between 0 and 1 inclusive. If p is smaller than some fixed and predetermined mutation rate r, then mutation takes place. Assuming that mutation is triggered at the number in position i, a random index j not equal to i is generated. If j > i, then x_i gets put in position j and element x_k k = i+1,...,j are shifted over one to the left. If j < i, then x_i is moved to position j and elements x_k k = j, j+1,...,i-1 are shifted over one to the right. Through experiment a Mäkinen nd Sieranta found a mutation rate of 0.03 worked best for this this type of operation.

www.manaraa.com

Mäkinen and Sieranta tested their algorithm on random bipartite graphs with equal sized vertex subsets and varying edge densities. On graphs of size 10 (i.e. 10 nodes per layer) and 15 the algorithm compared favorably to the barycenter heuristic. For size 15 graphs their GA to performed as much as 2% better than barycenter in terms of average crossings. On graphs of 50% density their GA performed similarly to barycenter. Although they did not publish running times, the authors noted that the primary drawback of their genetic algorithm was in its slow runtime due to expensive edge crossing calculations repeated over successive generations.

Another more recent application of genetic algorithms to bipartite graph drawings came with Zoheir Ezziane's GA for two-sided bipartite crossing minimization. [22]. Apart from the fact that it assumes both layers are free to permute, his algorithm also differs from that of Mäkinen and Sieranta in that it does not assume equal-sized vertex subsets, and also because it relies solely on a mutation operator. Ezziane chose this mutation-only strategy to avoid the significant computational work that often comes with incorporating a crossover operation.

The mutation operator he uses essentially swaps elements of the permutation array while keeping track of those that have already been considered. For each element x in the first layer (and similarly for each member of the second layer), a random number between 0 and 1 inclusive is generated. If said number is less than or equal to some preset mutation rate p (chosen to be 0.8 in the study) then x swaps permutation positions with another randomly chosen element y from its layer, but only if mutation hadn't previously

www.manaraa.com

been triggered at y. To see what this means, suppose that the permutation [3,2,1,4,5] is undergoing mutation under Zoheir's genetic algorithm. From the set {1,2,3,4,5} we start by selecting 1 and randomly generate a number between 0 and 1, let's say 0.7. Since that value is within range of the mutation rate, we select a random node to swap 1 with, let's say 3. The permutation then becomes [1,2,3,4,5]. We then pick the next element in our set, 2, and again randomly generate a number, say 0.85. Since 0.85 is greater than the mutation rate of 0.8, we leave 2 where it currently is in the permutation. As for the next element, 3, assume that mutation is again triggered and that it has been decided through random selection that it will trade places with 1. Since 1 already underwent a swap operation, it cannot trade places with 3. 3 could potentially switch positions with 2 however since 2 did not swap its position. If 3 (or any element for that matter) cannot find another tone to switch places with, then it will remain in the position assigned to it by the permutation.

Like Mäkinen and Sieranta, Ezziane also uses an elitist-based policy to ensure that the fittest individual is preserved through generations. It considers the fittest members from the current and previous generations, replacing the worst member of the current generation if the fittest member of the current generation has more edge crossings than the fittest member of the previous. Remaining members of the succeeding generation are selected through a so-called 'roulette wheel' selection policy that awards permutations with better fitness values a higher probability of survival. As Ezziane describes it, the relative and cumulative fitness values are calculated for each individual.

www.manaraa.com

Cumulative fitness is then subsequently used as a basis for selecting successive generation members. A summary of the procedure is outlined [28].

3.2 Simulated Annealing

Srivastava et. al. successfully applied simulated annealing to the two-sided bipartite drawing problem. [18] Following spectral initialization for linear arrangement approximation, their algorithm calculates linear arrangement costs of neighbors formed by swapping the positions of randomly selected pairs of nodes. Over time it also keeps track of neighbors with lowest crossing count. Every time that a neighbor solution is explored it's edge crossing number is calculated. If its crossing count value is equal to the best found so far it is added to a set of best crossing solutions. If it's lower than the best found value then the best set is emptied and the current neighbor is added to it.

Although the algorithm tries to achieve both low crossing numbers and linear arrangements, it avoids getting stuck in local linear-arrangement based optima by simultaneously keeping track of low crossing arrangements, even if they deviate from optimal linear arrangements. The authors found this dual strategy to outperform iterated barycenter, as well as the linear arrangement based methods of Newton et. al. [19] on a wide range of bipartite test graphs (the same categories of and some test data from [37]). Even more so, in about 30% of instances taken from their experiments, their algorithm was able to produce crossing numbers lower than those previously published. For cycles

www.manaraa.com

of even length and mesh graphs (graphs for which the minimum crossing number is known), the simulated annealing algorithm was able to achieve the exact minimum.

3.3 Stochastic Hill Climbing

Newton et. al. have successfully applied greedy randomized hill climbing to the one and two-sided crossing minimization problems [23][19]. In the one-sided case the basic sequential version of their heuristic works by randomly selecting nodes from the free layer and iteratively swapping them in a greedy fashion to reduce total crossing count. This differs from the 'stochastic' procedure tested by Jünger and Mutzel [8], and from the greedy switch and sifting heuristics of Eades and Kelly [24] and Matuszewski et. al. [25], the latter two of which greedily swap only neighboring nodes.

Newton et. al. compared several versions of their one-sided stochastic hill climbing procedure against the highly successful penalty minimization scheme of Demetrescu and Finocchi [11] and sifting method of Matsuzewski et. al. [25] on large sparse graphs. Specifically, they tested the performance of a standard sequential/iterative version of their heuristic and several parallelized versions of stochastic hill climbing for one-sided crossing minimization of random graphs with sizes n = 100, 200, …., 1000 nodes per layer and graph densities of 0.1%, 1%, or 10%. The parallelized versions of stochastic hill climbing involved interactions between master and slave entities where the master would send permutations to the slaves which would then modify them through application of stochastic hill climbing. A couple of the parallel methods that took on the

www.manaraa.com

form of genetic algorithms also introduced mutation and cooperation (in which the master maintains continuous communication with the slaves by resending the best or randomly chosen permutations to them).

The sequential and parallelized versions of stochastic hill climbing were all able to eventually surpass the sifting and penalty minimization results, showing the flexibility of randomized hill climbing for one-sided bipartite graph drawing. While the parallel versions of stochastic hill climbing were initially able to achieve better solutions than the sequential version faster, it took those and the sequential version about the same amount of time to reach the approximately optimal solution.

www.manaraa.com

Chapter 4: Methods

4.1 Weighted and Unweighted Bipartite Graph Drawing

6 algorithms in total were tested (the barycenter heuristic and 5 stochastic methods) against each other for one-sided bipartite crossing minimization performance. Except for barycenter each algorithm had a novel modification outlined below (e.g. different stopping condition, fitness evaluation, or parameter values) to improve performance and/or running times. All algorithms were implemented in Python with reference to the networkx library (version 1.9.1) for basic graph creation. 100 random weighted and unweighted bipartite graphs were generated for each combination of graph size (20, 25 and 30) and edge density (0.1, 0.2, and 0.3) using the author's own code. The term 'edge density' refers to the ratio of the number of edges in a graph to the maximum possible. The sets of graphs were then subsequently tested by each algorithm/heuristic. Weighted graphs were assigned random edge weights of 1, 2 3, 4 or 5. As most prior comparative studies only consider extremely sparse graphs for tests (e.g. graphs with density at most 0.1) it was thought beneficial to also consider graphs with higher densities of 0.2 and 0.3 so as to better gauge the scaling of algorithm/heuristic performance. Graphs with higher edge densities (in the range of 0.4 to 1.0) were not considered in the algorithm tests. Applications requiring low crossing minimization generally involve bipartite graphs that are on the sparser side. Also, the difference between algorithms when it comes to crossing minimization performance decreases with

www.manaraa.com

density, so the ranking of algorithms is more apparent when it comes to handling sparse graphs.

For both the weighted and unweighted scenarios, each algorithm/heuristic was run 5 times for every graph in a test set, with the average crossing number and standard deviation calculated for the 5 runs. The mean of all 100 average crossing numbers was then calculated for each test set and recorded. Similar calculations were done to determine the average deviation above the trivial lower bound for the number of crossings:

$$LB = \sum_{\{u,v\} \subseteq L_2} \min(c_{uv}, c_{vu})\{u, v\}$$

For each pair of distinct vertices u,v in layer 2, the lower bound calculates and sums up the weighted minimum of c_{uv} and c_{vu}. The symbol c_{ij} represents the number of crossings in the subgraph induced by the edges incident to nodes i and j, assuming that i is placed to the left of j. If the graph is weighted, then this term is equal to the total associated weighted crossings. Surprisingly, in the unweighted case, Jünger and Mutzel found this lower bound to be an extremely close approximation to the exact optimum as calculated by their branch and cut algorithm [8]. For random bipartite graphs of size 20 with densities 0.1 to 0.9 the lower bound was at most 0.3% lower than the optimum. For very sparse graphs (each node has degree 2 on average) with sizes ranging between 10 and 100 the lower bound was almost exactly equal to the exact minimum calculated by their branch and cut method.

www.manaraa.com

The same procedure for calculating averages crossings, times and deviations were done for the weighted graphs, the only difference being that edge weights were accounted for when calculating average weighted crossings in both the algorithms and the lower bound, and when determining total weighted edge lengths for the genetic algorithms. All algorithms/heuristics were implemented to make use of the divide and conquer procedure of Nagamochi and Yamada [17] when calculating total edge crossings.

A note should be made on initial x-coordinate assignments. All isolated nodes in the second layer (the layer being permuted) were ignored by the algorithms and appended at the end as the last nodes of the layer. The remaining nodes in the graph were given default x coordinates of 1,2,....etc. in each layer sorted according to the nodes' labels. So for example if the nodes in layers 1 and 2 were {0, 1, 2, 3, 4} and {5, 6, 7, 8, 9} respectively, then the x-coordinates for each layer would become {1, 2, 3, 4}. Similarly if by discounting isolated nodes in layer 2 we were left with nodes 5, 6 and 8, their default x-coordinates would become 1, 2 and 3 (1 assigned to node 5, 2 to node 6, and 3 to node 8). The isolated nodes would be appended to the end of the layer with x-coordinates of 4 and 5. With the exception of the hybrid simulated annealing algorithm which calculates an initial coordinate assignment, all graphs were assigned the default x-coordinates of the prior description when input into the bipartite graph drawing methods. Sections 4.1.1 and 4.1.2 provide details on the algorithms considered for unweighted and weighted bipartite graph drawing tests.

4.1.1 Unweighted Crossing Minimization:

www.manaraa.com

The algorithms/heuristics considered in the tests for unweighted bipartite graphs are as follows:

1. **BC:** the standard average-based barycenter heuristic
2. **Makinen_GA:** the GA of Mäkinen and Sieranta [21] for one-sided bipartite graph was implemented. Just as with two-sided drawings, the observation was made that one-sided drawings with fewer edge crossings tend to have lower linear arrangement values and vice versa. As such, a fitness function for measuring the linear arrangement value of a permutation was used to improve the genetic algorithm's running time, one of its primary drawbacks. A population size of 51 was used, but the mutation operation was taken out to improve runtimes. Cutting out mutation was found to significantly improve the overall runtimes of Makinen_GA. The same elitist selection policy, crossover, and mutation operators as described by the authors were used. As for the stopping condition, the algorithm would stop after 75 consecutive runs had passed with no improvement in the best found result. After reaching the stopping condition the algorithm would select and return the permutation with the lowest crossing count from the final generation.
3. **Zoheir_GA:** one-sided version of Zoheir Ezziane's [22] genetic algorithm for two-sided bipartite graph drawing [22] was implemented. As with Makinen_GA the genetic algorithm was evolved to minimize total linear arrangement value to obtain reasonable runtimes. A mutation rate of 0.8 and population size of 50 were

www.manaraa.com

used. These parameters appeared through experiment to give the best balanced tradeoff in terms of output vs. running time. The mutation step of Zoheir's GA was modified as well. In the original algorithm's description the mutation of a permutation involved the swapping of node positions while keeping track of previously handled nodes. The genetic algorithm implemented for the following tests was made to randomly swap nodes without keeping track of previously handled ones. This modification was made to improve overall runtimes. The algorithm would stop after 75 consecutive generations passed without improving the best found result. The same elitist roulette wheel selection policy (Algorithm 3) as described by Ezziane was used. Just as with Makinen_GA, Zoheir_GA would end by selecting and returning the permutation with the lowest crossing count from the final generation.

4. **Hybrid_SA:** A one-sided version of the two-sided hybrid simulated annealing algorithm of Srivastava and Sharma [18] was implemented. Unlike the original two-sided algorithm, no spectral calculations were done for initial node placement. Rather, all nodes kept their original default coordinates at the beginning of the algorithm. The nodes of layer 2 were permuted according to their ordering in the Fiedler vector, while the nodes of layer 1 were kept in their original fixed coordinate assignment. An initial temperature of $T_0 = 300$, final temperature $T_f = 1$, and cooling ratio = 0.97 were set as the main parameters for the algorithm. Also, 7 neighboring solutions were explored during each iteration

www.manaraa.com

of the algorithm. This number along with the previous parameters were chosen because they resulted in average crossing values that were closer in range to those computed by the other algorithms. Rather than returning a set of best solutions (as was described in [18]), the algorithm was modified to return just one permutation encountered over the course of Hybrid_SA that had achieved the lowest found crossing number.

5. **SHC:** the basic one-sided sequential stochastic hill climbing heuristic of Newton. et. al. [23] was implemented. The algorithm would stop after the best found solution had been encountered 100 times. To improve SHC running times an observation on edge crossings was made. To explain this, suppose that nodes x and y of layer 2 are randomly chosen by SHC for greedy swapping, with x initially lying to the left of y. Also consider [x:y], which denotes the nodes lying between x and y, sorted by x-coordinate. V' will denote the subset of nodes consisting of x, y and the nodes in [x:y], while E' will be the set of edges incident to the nodes of V'. If during an iteration of stochastic hill climbing we were to decide on whether to swap nodes x and y, the decision could be made based on edge crossings in the subgraph G' = (V', E'), rather than on the original graph G as a whole. That is, rather than having to calculate edge crossings for all edges of the graph (the most expensive operation per iteration of SHC), sometime could be saved by focusing only on those edges that may be affected by the node swap. These edges would be those defined by G'. As a concrete example, consider the

www.manaraa.com

bipartite graph of Figure 4. In this drawing, it is assumed that nodes 7 and 9 were randomly chosen from the second layer for greedy swapping. If said nodes are swapped, (i.e. 7 gets placed in the position previously occupied by 9 and 9 is assigned the initial x-coordinate of 7), then the only edges whose crossing counts will be affected are those lying in the subgraph induced by nodes 7, 8 and 9 and the associated incident edges. Focusing on this smaller subgraph helps to cut down on total running time by discounting the edges whose cross counts stay the same before and after a greedy swap.

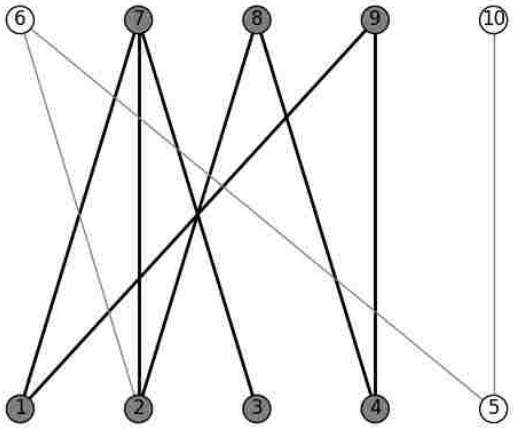

Figure 8: When swapping nodes 7 and 9 during stochastic hill climbing, examining the subgraph induced by nodes 7 through 9 and their incident edges suffices for edge crossing calculations.

6. **3-WOLF:** unweighted version of the 3-approximation algorithm by Cakiroglu et. al. [9] was implemented. In the unweighted case weights of 1 were uniformly assigned to all edges of the graph.

www.manaraa.com

4.1.2 Weighted Crossing Minimization:

Weighted bipartite graph testing was done under two different scenarios. In the first series of tests the algorithms/heuristics considered were the same as those used in the unweighted case, the only difference being that edge weights are now accounted for (e.g. when calculating weighted edge crossings, weighted linear arrangements, and weighted barycenter averages). In the case of linear arrangements, if an edge e has weight w_{ij} and endpoints with coordinates of (x_1, y_1) and (x_2, y_2), then it's total weighted linear arrangement value is equal to $w_{ij}*|x_1 - x_2|$.

In the second line of testing the traditional barycenter heuristic (BC) was tested against its own weighted variant (BC_W) on random weighted bipartite graphs. This was done to see how well the barycenter heuristic could approximate the weighted crossing minimization numbers by ignoring edge weights rather than factoring them into its average-based calculations.

4.2 Bottleneck Crossing Minimization

Two novel algorithms for one-sided bottleneck crossing minimization were tested against each other: a one-sided version of the maximum edge crossings heuristic (MEC) and a variation of stochastic hill climbing that greedily swaps vertex positions according to the same local bottleneck crossings principle of MEC. Two versions of each heuristic were considered, those with and those without initial barycenter preprocessing. As in Section 4.1, heuristics for one-sided bottleneck crossing minimization were implemented

www.manaraa.com

in python with reference to the networkx library (version 1.9.1). Random unweighted test graphs with sizes 20, 25 and 30 and densities 0.1, 0.2 and 0.3 were generated using the author's own code, with each graph size and density class consisting of 100 random bipartite graphs.

Each heuristic was run 5 times for every graph in a test set, with the average bottleneck crossing number calculated for the 5 runs. The mean of all 100 average crossing numbers was then calculated for each test set and recorded. The average times (recorded in seconds) were calculated similarly. Unlike the case of bipartite crossing minimization, there is no trivial lower bound to approximate the optimal bottleneck crossing number for the one-sided bottleneck problem. Comparisons of heuristic performance were thus made based on the average results of each heuristic's column for specified graph sizes.

All test graphs were initially assigned the same default x-coordinates of Section 4.1 when input into the bottleneck crossing minimization methods. It is assumed in both heuristics that node permutations are being executed only on the second layer, L_2, of the bipartite graph. Specific implementation details for the tested heuristics are now outlined as follows:

1. **OMEC (One-Sided Maximum Edge Crossings):** A one-sided version of the Maximum Edge Crossing heuristic originally described by Gupta and Stallman [27]. The EDGESIFT sub-procedure is the same as that of Section 2.5. Rather

www.manaraa.com

than using their 'swap' procedure for updating edge crossings, a simple edge counting sub-procedure, Edge_Count_Simple (Algorithm 7), was used. Assuming an edge e_1 has endpoints with x-coordinates x_1 (in layer 1) and x_2 in layer 2 and similarly that another edge e_2 has endpoint x-coordinates of y_1 and y_2 of layers 1 and 2 respectively, the algorithm makes use of the fact that e_1 crosses with e_2 if either 1: $x_1 < y_1$ and $y_2 < x_2$ or 2: $y_1 < x_1$ and $x_2 < y_2$. In the first case edge e_2 is said to cross "to the left of e_1" and in the second e_2 crosses "to the right of e_1". Although the implementation of Edge_Count_Simple assumes that both graph layers have the same number of nodes, it can easily be modified to handle unequal-sized vertex subsets.

Algorithm 6: OMEC

initialize all nodes v $\in L_2$ as 'unhandled'.

while there is an edge with an unhandled endpoint in L_2 **do**:

let e = vw be an edge with an unhandled endpoint v in layer 2 s.t. c(e) = c*(E)

EDGESIFT(v)

update c(e) for e $\in$ E

label node v as 'handled'

end do

Figure 9

www.manaraa.com

Algorithm 7: Edge_Count_Simple (coords)

/*input coords is an array of dictionary defining the (x,y) coordinates of nodes*/

/*initialize an nxn matrix where n = $|L_1|$ = $|L_2|$*/

create an empty nxn matrix M with rows and columns indexed by layer 1 and 2 x-coordinates respectively.

initialize all entries of M to be zero

/*assume $v_i \in L_1$ and $w_j \in L_2$*/

for each e = $\{v_i, w_j\}$ E **do**:

x_1 = x-coordinate of node v_i as defined in coords

x_2 = x-coordinate of node w_j as defined in coords

w_{ij} = weight of edge e = $\{v_i, w_j\}$

$M[x_1][x_2] = w_{ij}$

end do

/*initialize total number of edge crossings*/

edge_crossing_count = 0

/*sum the number of times each edge crosses with edges to its 'left'*/

www.manaraa.com

```
for each e = {v_i, w_j} ∈ E do:
        x_1 = x-coordinate of node v_i as defined in coords
        x_2 = x-coordinate of node w_j as defined in coords
        w_ij = weight of edge e = {v_i, w_j}

        for i = 1 to x_1 do:
            for j = x_2+1 to n do:
                edge_crossing_count = edge_crossing_count + w_ij*M[i][j]
end do
/*sum the number of times each edge crosses with edges to its 'right'"*/
    for each e = {v_i, w_j}∈ E do:
        x_1 = x-coordinate of node v_i as defined in coords
        x_2 = x-coordinate of node w_j as defined in coords
        w_ij = weight of edge e = {v_i, w_j}

        for i = x_1+1 to n do:
            for j = 1 to x_2 do:
                edge_crossing_count = edge_crossing_count + w_ij*M[i][j]
    end do
    return edge_crossing_count
```

Figure 10

2. **BSHC (Bottleneck Stochastic Hill Climbing):** A one-sided stochastic hill climbing method adapted to handle bottleneck crossings using the local bottleneck

www.manaraa.com

crossing principles of MEC's EDGESIFT. As with OMEC, Algorithm 7 was implicitly utilized by BSHC to handle the maintenance of edge cross counts. As reflected in its pseudocode (Algorithm 8), BSHC was made to keep running until 15 consecutive generations had passed with no improvement in the graph's bottleneck value.

Algorithm 8: BSHC

/*maintains number of consecutive iterations that have passed with no improvement in *bottleneck_value*/

consecutive_count = 0

/*maintains the local bottleneck value to be optimized*/

bottleneck_value = $c^*(E)$

while (*consecutive_count* < 25) **do**:

select a vertex $v_1 \in L_2$ incident to an edge $e = v_1w$ s.t. $c(e)$ = *bottleneck_value*

randomly select another vertex $v_2 \neq v_1$ from L_2

swap the x-coordinates of v_2 and v_1

let $c_i = c^*(E(v_1) \cup E(v_2))$

/*reswap the nodes if there was no improvement in the local bottleneck value*/

if $c_i \geq$ *local_bottleneck* then:

swap the x-coordinates of v_2 and v_1

/*recalculate the bottleneck value of the graph*/

bottleneck_value = $c^*(E)$

www.manaraa.com

Figure 11

3. **BC + OMEC:** OMEC with initial barycenter preprocessing. Applies OMEC to a bipartite graph that has had the barycenter heuristic initially applied to it.
4. **BC +BSHC:** BSHC with initial barycenter preprocessing. Applies BSHC to a bipartite graph that has had the barycenter heuristic initially applied to it.

www.manaraa.com

Chapter 5: Results

5.1: One-Sided Unweighted Crossing Minimization

Average test results on graphs of size 20 in terms of percentage deviation from the lower bound and time measured in seconds are given in Figures 12 and 13 below. For full results including average crossing values, standard deviations, and results for larger graphs, see the Appendix.

From Figure 12 one can clearly see that all crossing minimization methods gradually perform better on average with increasing graph density. For instance, Hybrid_SA achieved average percent deviations of 6.5, 3.7, and 2.5 percent above optimality for graph densities of 10, 20 and 30 percent respectively. Similarly, the barycenter heuristic resulted in corresponding percent deviations of 3.0, 1.2, and 0.6. This trend of improved performance with higher edge density is likely due to there being less room for improvement as the number of edges in a graph increases. As a graph's density grows, it naturally becomes harder to avoid the crossing of pairs of edges, making it "easier" for an algorithm to have closer to optimal output.

www.manaraa.com

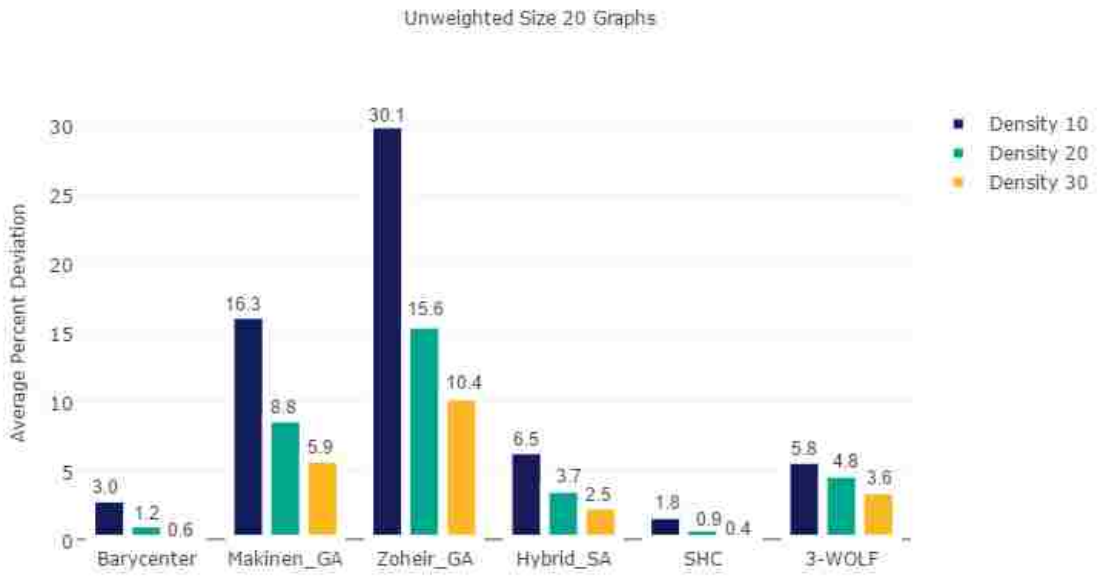

Figure 12: Average percent deviation of algorithms from the lower bound on random, unweighted, size 20 graphs.

www.manaraa.com

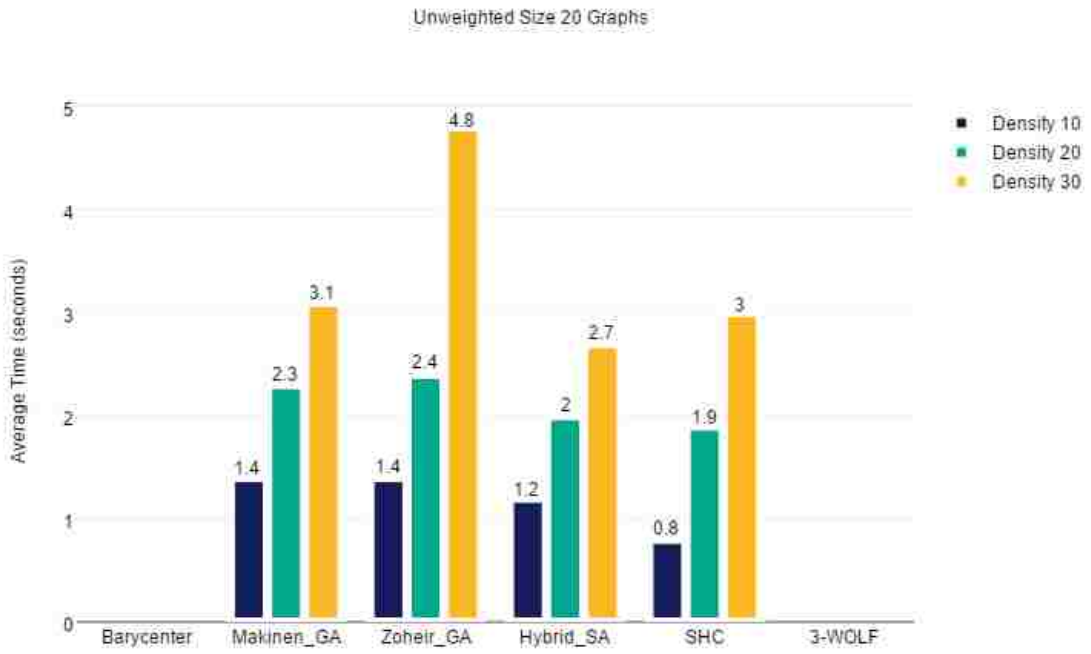

Figure 13: Average algorithm times (in seconds) when executed on random, unweighted, size 20 graphs.

Not surprisingly, barycenter ranks within the top two of the 6 methods when it comes to minimizing crossings. As previously noted, for size 20 graphs with densities 10, 20 and 30 percent the heuristic was able to achieve crossings 3, 1.2, and 0.6 percent higher than the associated lower bound on average. The only other method to outperform it was the one-sided stochastic hill climbing heuristic SHC, which produced corresponding deviations of 1.8, 0.9 and 0.4 percent above the lower bounds. However, despite its close to optimal crossings performance stochastic hill climbing was not the fastest method, with

www.manaraa.com

running times similar to those of one-sided hybrid simulated annealing (although with a higher standard deviation values). On graphs of size 20 and density 30%, for instance, SHC averaged a runtime of 3.0 seconds. This was clearly slower than the 0.0 second runtimes of barycenter and WOLF, the two fastest methods for one-sided crossing minimization. The high speed of these algorithms can be attributed to the fact that they are deterministic, reordering nodes according to certain calculated sums.

Although 3-WOLF and Hybrid_SA did not rank within the top 2, the two methods (ranked 3rd and 4th) still achieved crossing values that were close to optimal, being at most 6.5 percent above the average lower bound. Interestingly, although 3-WOLF was designed to optimize weighted edge crossings, it still performed very well on the unweighted graphs. For size 20 graphs it's average crossings ranged between 5.8 and 3.6 percent above the average lower bounds. Although the hybrid simulated annealing algorithm produced average results close to those of 3-WOLF, it did not achieve the same level of optimality as it did when applied to its original two-sided crossing minimization context [18]. This could be due to its hybrid design, which searches for permutations that give both low linear arrangement and low crossing count values. Although low linear arrangements tend to coincide with low cross counts in both the one and two-sided cases, they do not necessarily yield optimal results for the latter. The previously cited relations between linear arrangements and edge crossings in bipartite graphs was proven with the assumption that both layers (i.e. every node in the graph) could be permuted [2,3]. It could be then that part of the reason Hybrid_SA did not give results closer to stochastic hill climbing was because

www.manaraa.com

of the one-sided context of the drawing problem. When applied in its original two layer setting the linear arrangement values that it optimizes for likely give closer approximations to optimal graph crossings than in the current study's single layer case. Hybrid_SA may have also missed the cut in rankings because it was not given any spectral initialization in the one-sided crossing tests. The original two-sided version of the algorithm began with an initial Fiedler vector-induced placement of nodes. This likely helped the simulated annealing algorithm a lot both in terms of average crossings and in terms of time, something that the one-sided version, Hybrid_SA, did not benefit from.

In regards to the two genetic algorithms, it was the algorithm of Mäkinen and Sieranta that consistently performed better in terms of average crossings. For density 10, 20 and 30 percent graphs of size 20 Makinen_GA deviated 16.3, 8.8 and 5.9 percent, on average, above the lower bound. This was clearly lower than the average corresponding deviations of 30.1, 15.6 and 10.4 percent that resulted from application of Zoheir_GA. With the exception of Zoheir_GA on size 20 and density 30 percent graphs which had an average runtime of 4.8 seconds, the two genetic algorithms were able to achieve runtimes close to those of Hybrid_SA and SHC. This shows that at least in the unweighted context with graphs that are not too large, genetic algorithms for bipartite graph drawing can be time competitive if given a suitable proxy measurement for fitness evaluation. The increase in time efficiency does come at a cost though in terms of average crossings performance, with Makinen_GA and Zoheir_GA ranking 5th and 6th according to crossing minimization performance.

www.manaraa.com

As to why the genetic algorithm of Mäkinen and Sieranta performed better than that of Zoheir, it could be due to several things. For one, although the crossover operation of Makinen_GA was quite expensive, it did appear by design to introduce a lot of randomization in the resulting child permutations, for instance when each child would inherit its missing elements according to their order in the opposing parent. At the same time, the crossover operation was also made to preserve the ordering of nodes within segments from both (potentially fit) parents. In addition, the combination of more and less fit permutations during this phase helped to randomize the process even more. As a result, even though the mutation operator was removed to improve runtimes, the combined crossover and selection portions of Makinen_GA appeared to have enough randomness to avoid getting stuck in local optima while still evolving permutations with good fitness values. As it repeatedly swaps the positions of nodes at random, the genetic algorithm of Zoheir Ezziane does appear on the surface to be capable of searching a large space of permutations and to have a lot of inbuilt randomness, like Makinen_GA. However, unlike stochastic hill climbing, the genetic algorithm of Zoheir does not keep track of improvements in edge crossings with node swaps. Even if the reordering of a pair of nodes results in an increase in edge crossings, it will accept the permutation just the same. Even more importantly though, the iterative randomized swap procedure of Zoheir's GA is less likely to preserve good node arrangements from previous generations. While the genetic algorithm of Mäkinen and Sieranta preserves permutation segments during crossover, the heavily mutation based algorithm of Zoheir is constantly changing permutations at a high

www.manaraa.com

rate. As a result, it could be the case that Zoheir_GA was the lesser of the two genetic algorithms because, by its design, it is harder to maintain a consistently high average fitness value from generation to generation.

5.2: One-Sided Weighted Crossing Minimization

Following in Figures 14 and 15 are the average results of the weighted versions of the methods from Section 5.1 when applied to random weighted bipartite graphs. As in the unweighted case only the average percent deviations and times are reported for weighted size 20 graphs. Full results for graph sizes of 20, 25 and 30 can be found in the appendix.

Apart from the average weighted crossing values being higher (due to the addition of integer edge weights between 1 and 5) as well as the times, the percent deviations from average lower bounds are also larger when compared to the corresponding percents for unweighted graphs. This can in part be attributed to the rise in crossing minimization complexity that comes with adding edge weights to a graph. The top three methods, SHC, 3-WOLF, and Hybrid_SA, scale well with the added complexity and show only small increases in their deviations from optimality and average runtimes in comparison to their unweighted counterparts. For density 10 percent size 20 graphs 3-WOLF generally outperformed Hybrid_SA, while for higher 20 and 30 percent density graphs Hybrid_SA tended to give similar or slightly lower percentage deviations than 3-WOLF.

www.manaraa.com

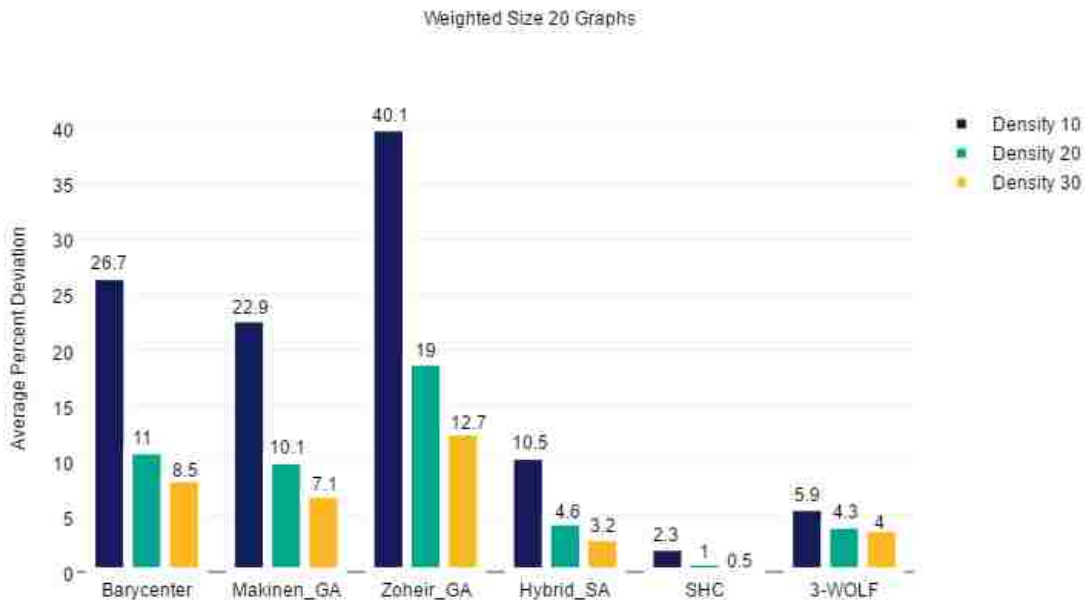

Figure 14: Average percent deviation of algorithms from the lower bound on random, unweighted, size 20 graphs.

www.manaraa.com

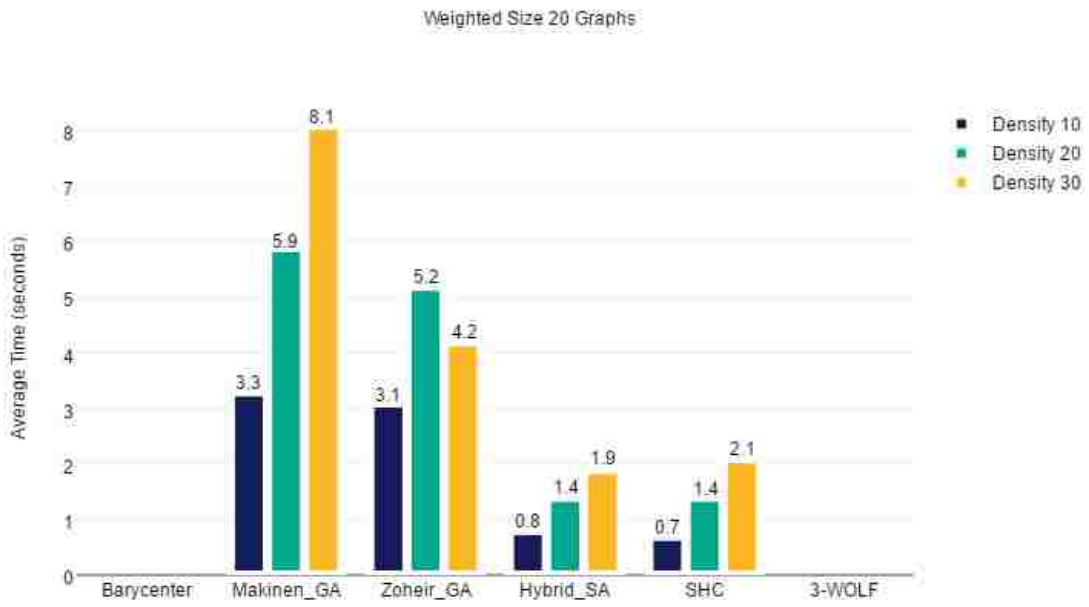

Figure 15: Average algorithm times (in seconds) when executed on random, weighted, size 20 graphs.

Interestingly, the weighted version of barycenter no longer achieves top ranking in the weighted drawing context. A similar observation was made by Cakiroglu et. al. when experimenting with weighted versions of common heuristics [9]. For unweighted bipartite graphs with size 20 and density 10, the barycenter heuristic was able to come within 3 percent of the estimated lower bound on crossings. On weighted graphs of the same size and density, however, it's percent deviation is over 26 percent, considerably higher than the near optimal 2.3 percent of the edge-weighted stochastic hill climbing and the second best 5.9 percent of 3-WOLF. Even on random weighted graphs with 30 percent density barycenter performed 8 percent worse than stochastic hill climbing and had more than

www.manaraa.com

double the percentage deviation value of 3-WOLF. The only method that weighted barycenter consistently performed better than was the weighted variant of Zoheir's GA. The genetic algorithm of Makinen and Sieranta was able to achieve slightly better weighted crossing values than barycenter on random weighted graphs of size 20.

In the weighted scenario Makinen_GA continued to outperform Zoheir_GA although with higher runtimes runtimes. The reasons for its superior crossings performance should be the same as in the unweighted case, namely that the former genetic algorithm appears more capable of evolving populations of permutations with higher average fitness than Zoheir's genetic algorithm. As for its runtimes, the genetic algorithm of Makinen and Sieranta clearly takes longer to run in the weighted scenario versus the prior unweighted case. This may be because the crossover operator that it is heavily dependent on may not scale well with the added complexity of weighted edges. What functions as a relatively efficient operator on unweighted bipartite graphs may not translate to efficiency or rapid convergence on their weighted counterparts.

That neither genetic algorithm was as competitive as the top 3 for weighted crossing minimization is likely due to their weighted linear arrangement based fitness functions which don't appear to approximate weighted edge crossings very well. Essentially, the reasoning would be similar to that made in the previous section for explaining why the hybrid simulated annealing algorithm wasn't able to achieve top ranking.

www.manaraa.com

As previously noted, stochastic hill climbing was the clear winner for minimizing weighted edge crossings. It was not as fast as the deterministic 3-WOLF though, which achieved the same rounded runtimes as barycenter. On size 20 graphs the runtimes for SHC ranged between 0.8 and 3.0 seconds on average, similar to Hybrid_SA. However, despite its slower runtimes, the fact that the weighted stochastic hill climbing heuristic, the simplest of all the crossing minimization procedures, was able to consistently beat 3-WOLF by at least more than 3 percent on size 20 graphs is impressive, particularly when one considers how much a percentage difference can account for in terms of total weighted crossings (see appendix results), and also because 3-WOLF has been shown to be a near optimal algorithm for weighted crossing minimization. The inherent flexibility of stochastic hill climbing and its ability to achieve crossings so close to the approximate lower bounds suggest that it is not only effective as a simple stand-alone method for weighted crossing minimization, but also as a potential post-processing procedure. If the graph data isn't too large and/or runtime is not an issue, then stochastic hill climbing alone can be used for approximating the minimum crossing number for a weighted bipartite graph. If, on the other hand, the graph data is quite large and/or runtime is more of an issue, then weighted stochastic hill climbing can be applied as a greedy post-processing method to improve the result of 3-WOLF for a weighted graph.

5.3: Weighted versus Unweighted Barycenter Results:

www.manaraa.com

Average results of weighted (BC_W) and unweighted (BC) variants of the barycenter heuristic for weighted size 20, 25 and 30 bipartite graphs are presented in figures 16, 17 and 18. Actual average weighted crossing numbers can be found in the appendix. Since both variants of the barycenter heuristic consistently averaged runtimes of 0.0 seconds, bar charts reflecting execution times were excluded.

The traditional barycenter heuristic was not originally designed to account for edge weights. If given a weighted graph as input, BC simply ignores the edge weights and orders nodes according to the averages of their neighbors' x-coordinates. BC_W, on the other hand, does consider edge weights in its calculations by computing averages as edge-weighted sums of neighbors' x-coordinates.

Neither BC nor BC_W was able to outperform the best methods of Section 5.2. However, interestingly, the simple non-edge-weighted version of barycenter performs better than its weighted variant when it comes to minimizing weighted edge crossings. This goes against the intuition that factoring in edge weights will help to improve the performance of a heuristic like barycenter when tested on weighted graphs. As one can see though, by ignoring edge weights the simpler version of barycenter was able to improve the average edge crossing performance by a significant amount over its weighted counterpart in some cases. For bipartite graphs with density 10 percent and sizes 20, 25 and 30 the corresponding improvements in average crossings were about 10, 7 and 5 percent respectively. For density 20 percent graphs with sizes 20, 25 and 30 the corresponding average improvements were 3.6, 2.9, and 2.6 percent. And for density 30

www.manaraa.com

percent graphs with sizes 20, 25 and 30 the average improvements were 2.8, 1.8, and 1.7 percent.

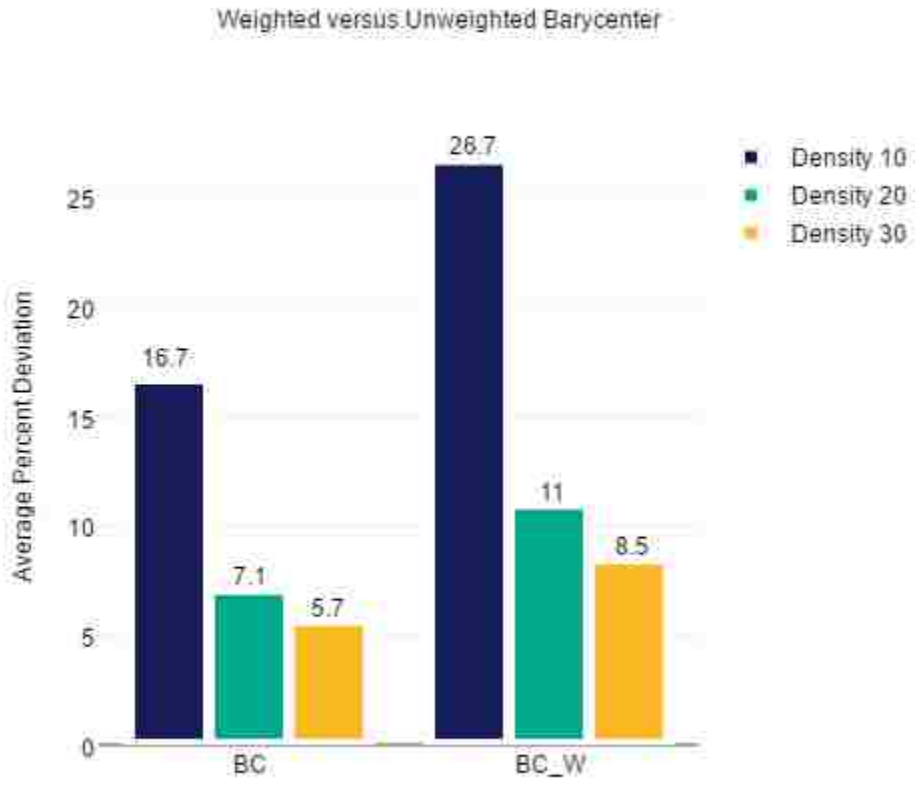

Figure 16: Average percent deviations of weighted (BC_W) versus unweighted (BC) barycenter on random weighted bipartite graphs of size 20.

www.manaraa.com

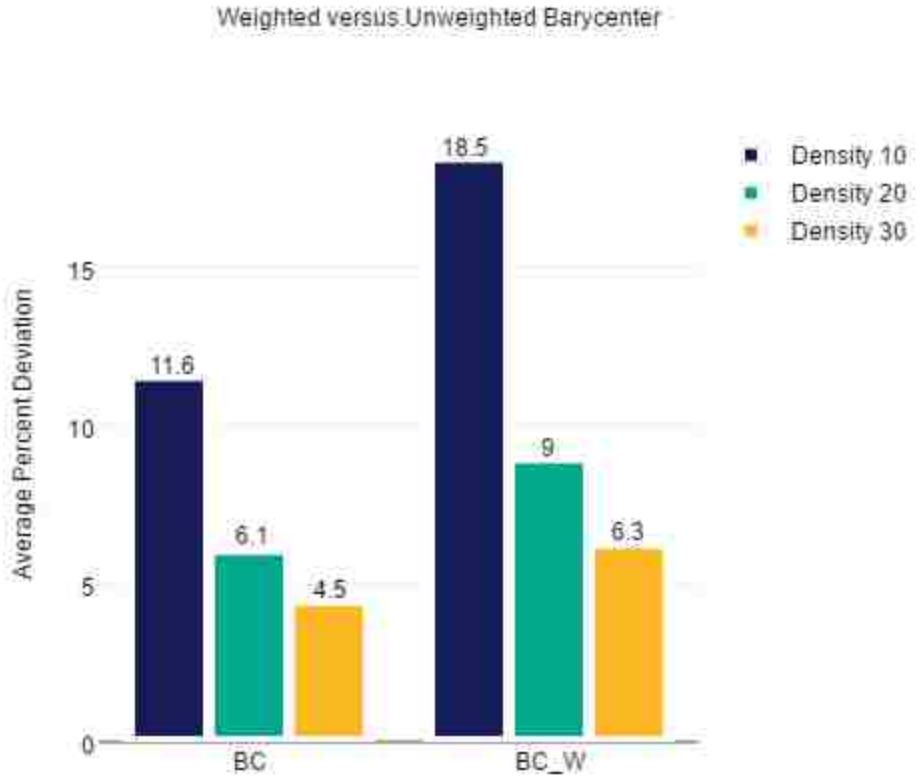

Figure 17: Average percent deviations of weighted (BC_W) versus unweighted (BC) barycenter on random weighted bipartite graphs of size 25.

www.manaraa.com

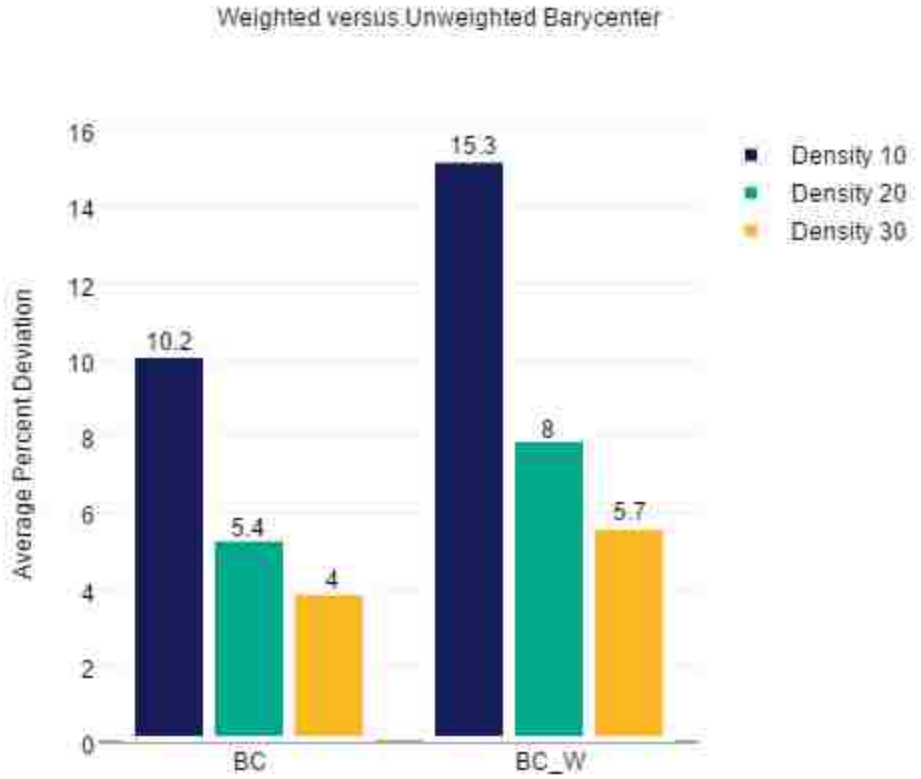

Figure 18: Average percent deviations of weighted (BC_W) versus unweighted (BC) barycenter on random weighted bipartite graphs of size 30.

Although the standard barycenter heuristic consistently performs better on the test graphs than its weighted variant, one may notice from the previously cited numbers that the percentage difference between the two decreases with graph size and density. It is possible that the two versions of barycenter perform more similarly with bigger graphs simply due to their higher complexity. It is especially the case that with higher density

www.manaraa.com

graphs, as the number of edges increases, the possibilities for reducing weighted crossings should go down as more pairs of edges come up that can't avoid crossing with each other. This is not to say, however, that the relative performance of the two methods should be judged purely based off of percentage differences. Even a percentage difference seemingly as small as 1.7 can amount to a weighted edge crossing reduction of almost 2,000 (e.g. when comparing the average 123808.2 of BC for size 30 density 30 graphs to the corresponding average of 125794.6 of BC_W as shown in the appendix). This could potentially be significant based on the application in which such results may arise, giving even more credence to the simpler fast heuristic that ignores edge weights.

As for the sparser, density 10 graphs, the unweighted barycenter heuristic clearly performs better than its weighted counterpart. As to why this may be the case, it is useful to think of an edge-weighted graph as a multigraph (a graph that allows for loops and multiple edges). Rather than picturing a weighted edge as a straight line running between layers with a number attached to it, one can also imagine it as several straight lines each with a weight of one and all connecting the same pair of vertices. So, rather than having a single edge between two nodes with a weight of 3, one could alternately have three parallel edges with weight one connecting the same pair of nodes. Minimizing crossings in a weighted bipartite graph then becomes equivalent to minimizing the crossings in its corresponding multigraph. If we now think in terms of arranging nodes to reduce the crossings on each single edge of unitary weight in the multigraph, the regular barycenter method that tends to straighten edges by calculating unweighted averages seems to be the

www.manaraa.com

natural choice. Introducing edge weights in this context could potentially throw off the average calculations and lead to a permutation that results in bad crossing values not just for one, but all parallel edges connecting the same pair of nodes. Since the placement of a node affects all parallel edges adjacent to it, good heuristic like the unweighted barycenter that gives low crossing values for single edges should benefit the remaining parallel edges. This could explain its superior performance on the crossing values of the weighted graph as a whole.

5.3: One-Sided Bottleneck Crossing Minimization

Average bottleneck crossing minimization results for OMEC, the one-sided variant of MEC, BSHC, the bottleneck crossings based stochastic hill climbing procedure, and each of the two methods with barycenter preprocessing (BC+ OMEC and BC+BSHC respectively) are presented in figures 19 and 20 below. Only size 20 graph results are shown, with results for other graph sizes and standard deviations given in the appendix.

As one can see, for all graph densities the bottleneck edge oriented variant of stochastic hill climbing was able to outperform OMEC by a significant amount in terms of average bottleneck crossing values. For size 20 density 10 graphs the average bottleneck value of BSHC was 22.5% better than that of OMEC (27.8/22.7). When the density increased to 20% the ratio of OMEC to BSHC was about 1.2. On size 20 graphs the two methods had similar runtimes. However, as the edge density and graph size increased so did the difference between the methods' times (see appendix), with the average runtimes

www.manaraa.com

of OMEC significantly outgrowing those of BSHC. **Make note on slower runtime here for OMEC.**

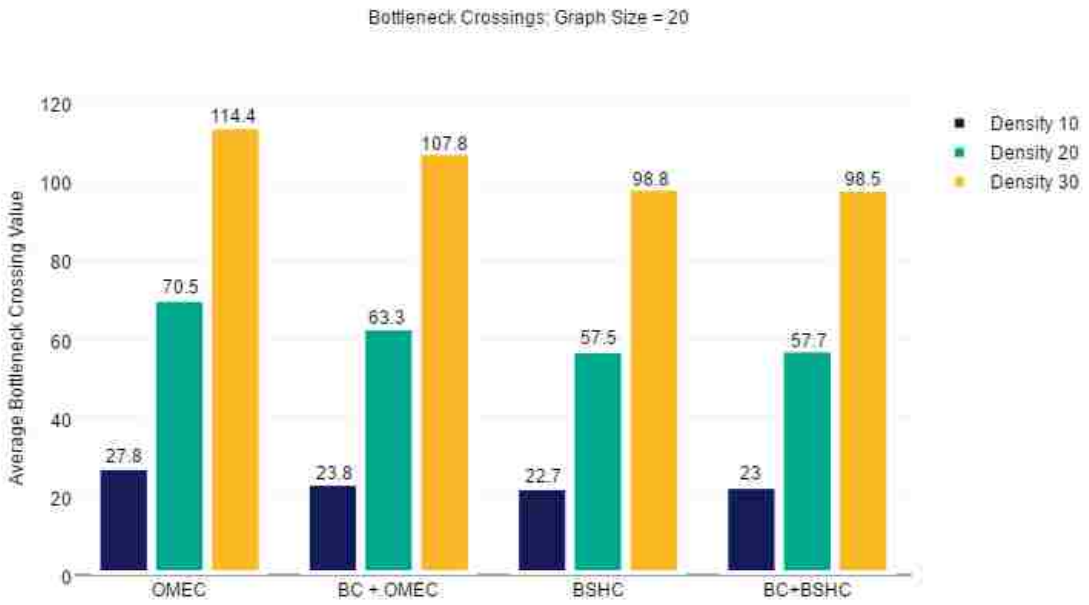

Figure 19: Average bottleneck crossing results of OMEC and BSHC (with and without barycenter preprocessing) on random unweighted size 20 bipartite graphs.

www.manaraa.com

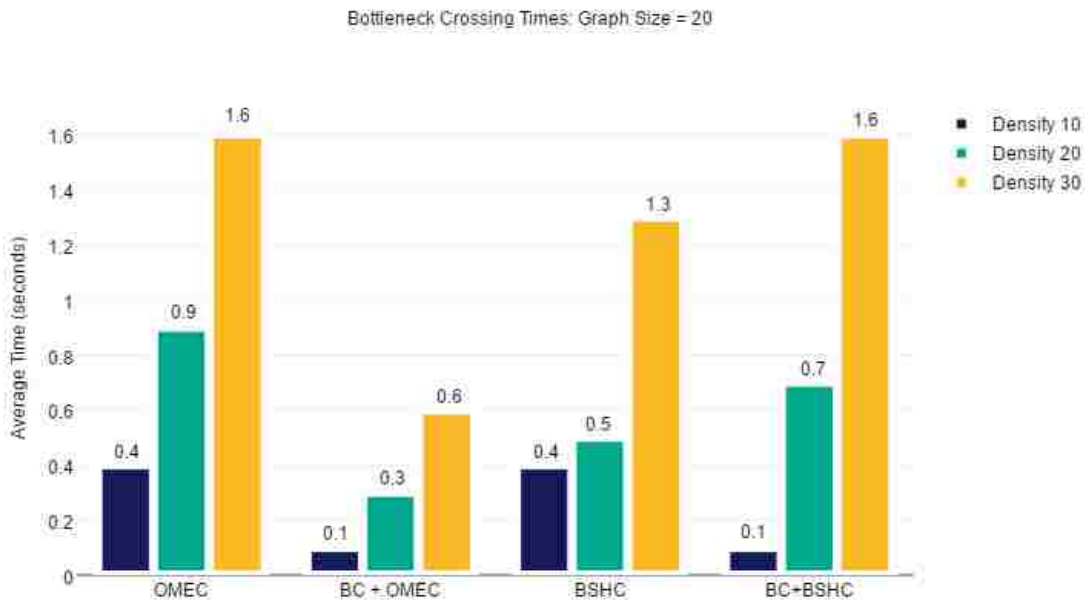

Figure 20: Average bottleneck crossing times of OMEC and BSHC (with and without barycenter preprocessing) on random unweighted size 20 bipartite graphs.

Apart from OMEC and BSHC Figure 19 also considers the results of the two methods when applied to random graphs that have been reordered by barycenter for initial preprocessing (i.e. BC+OMEC and BC+BSHC). The intuition behind this preprocessing step is that by applying barycenter first, the test graphs should get a quick reduction in the average crossings occurring on individual edges. With a smart initial layout like this that is closer to optimality, the heuristics shouldn't take as much time to further refine the ordering for bottleneck crossing minimization. According to the results, this intuition was correct. In most cases barycenter preprocessing helped to reduce the average bottleneck

www.manaraa.com

crossing values along with cuts in corresponding runtimes. For OMEC the initial barycenter-induced orderings helped reduce its average bottleneck crossings by significant amounts. Without preprocessing, the average OMEC bottleneck values on size 20 graphs with 10, 20 and 30 percent edge densities were 27.8, 70.5 and 114.4 respectively. With preprocessing these numbers decreased to 23.8, 63.3 and 107.8, values closer to those output by BSHC. Barycenter preprocessing was also able to reduce the runtimes of OMEC slightly for size 20 graphs, and in some cases significantly or size 25 and 30 graphs (see appendix). For bottleneck-based stochastic hill climbing the average bottleneck values obtained with preprocessing were similar to those without barycenter initialization. This is a likely indication that the values obtained through BSHC were already close to optimal. And as with OMEC, the runtimes of BSHC showed slight improvements with barycenter preprocessing that gradually became more apparent as the graph size and density grew (see appendix).

Despite the reduction in the crossings performance gap between BSHC and OMEC, BSHC was still able to achieve the lowest average bottleneck crossings after initial barycenter processing. Since OMEC ignores nodes after passing them to EDGESIFT, it's exploration of the search space is much more limited than bottleneck-based stochastic hill climbing, even after barycenter has improved the initial layout. As a result, the approximations for the bottleneck crossing minimization problem on the test graphs were better for BSHC both with and without barycenter preprocessing.

www.manaraa.com

Chapter 6: Conclusions and Future Research

Stochastic hill climbing was consistently found throughout all of the preceding tests to perform the best for one-sided unweighted, weighted, and bottleneck bipartite crossing minimization. These findings were particularly notable in the weighted and bottleneck crossing minimization contexts. In these cases, stochastic hill climbing was found to be flexible enough to surpass the results of 3-WOLF and the one-sided version of the MEC heuristic for handling bottleneck crossings. In their previous work Cakiroglu et. al. [9] had found 3-WOLF to be a top performing algorithm for one-sided weighted crossing minimization. Despite being the simplest of the crossing minimization methods, stochastic hill climbing had enough randomization to outperform the prior dominance of 3-WOLF. Also prior to this study, the one-sided bottleneck crossing minimization problem had never been considered, and in the multi-layer crossing minimization case MEC was found to be the best algorithm for the problem. In the tests on random unweighted bipartite graphs of varying sizes the bottleneck-based stochastic hill climbing procedure was found to consistently yield average bottleneck crossing values significantly lower than those output by the one-sided version of the maximum edge crossings heuristic. Even after applying the barycenter heuristic on test graphs for initial node placement, bottleneck-based stochastic hill climbing was still found to outperform the one-sided variant of MEC.

The main drawbacks for stochastic hill climbing came in terms of time, with gradual increases in runtime as the test graphs got larger and denser. If runtime is not

www.manaraa.com

critical then the stochastic hill climbing procedures could be used alone for solving the crossing minimization problems of the preceding sections. Being able to improve the average crossing values by even a small percentage gives stochastic hill climbing a competitive edge, as even a small percentage deviation from optimality can translate to thousands of corresponding edge crossings. If the time efficiency for calculating crossing minimization is important, then stochastic hill climbing could still be used as a post-processing procedure, say after barycenter or 3-WOLF, to reduce edge crossings. This was evidenced in the case of bottleneck crossing minimization, where initial barycenter placement was found to improve the runtimes of stochastic hill climbing by significant amounts while still achieving superior crossing values. And if simplicity is desired, then stochastic hill climbing is particularly attractive in that it is, besides barycenter, the simplest algorithm to implement. That one of the simplest methods was able to achieve the best approximations for the hard crossing minimization problems is particularly impressive.

With these conclusions in mind and in relation to the overall work of this thesis the following extensions and problems could form promising lines of future research:

1. Although Stallman and Gupta [27] claim that it is simple to prove the NP-Hardness of the multi-layer bottleneck crossing minimization problem by adapting techniques from previous work [1][4], a formal proof is not supplied in their technical report or in the subsequent paper by Stallman [29]. A formal proof of this claim would be desirable, as well as one for the one-sided variant of the problem.

www.manaraa.com

2. The majority of studies that concern bipartite graph drawing assume that the minimization of crossings is not only beneficial for circuit-based applications, but is also the basis of the easiest to read and aesthetically pleasing of bipartite graph layouts. It would be interesting to conduct a study by which users view and rate bipartite graph drawings that have been optimized according to varying criteria: crossing minimization, minimal edge length, and bottleneck crossing minimization. Although graph drawings are primarily judged by edge crossings, it may be that in the case of bipartite graphs, drawings that are optimized according to other aesthetic measures appear more useful and are easier to understand.
3. In the case of traditional bipartite graph drawing, exact methods exist for one-sided [8], two-sided [39] and multilayer crossing minimization [38]. To the author's knowledge there are no published algorithms for calculating the minimum bottleneck crossing number for bipartite graphs. It would be very useful for future research to have such a method, not only to solve the problem exactly and hopefully efficiently but also to provide a minimum baseline to gauge the crossing performance of various approximation algorithms and heuristics.
4. The problem of two-sided weighted crossing minimization has never been studied. It would be worthwhile to compare the performance of an iterative 3-WOLF against a two-sided weighted stochastic hill climbing in this context. It would also be interesting to see how iterated barycenter methods (both ones including and excluding edge weights in their calculations) would perform, especially since the

www.manaraa.com

iterated barycenter heuristic is one of the primarily dominant methods for unweighted two-sided bipartite graph drawing.

5. It was observed that the traditional barycenter heuristic yielded lower weighted crossings when ignoring edge weights than the weighted variant that has been used previously by researchers for weighted graphs. The difference between the two in terms of percentage deviation from optimality does appear to close though with increasing graph sizes and densities (see appendix). Further experimental studies should be done to compare the performance of the two versions of barycenter on larger-sized graphs and with more variation in edge weights to see how the results may change with bigger data.
6. It would be worthwhile to repeat the weighted and unweighted one-sided drawing experiments with barycenter heuristic methods that have been augmented with various tie breaking heuristics (i.e. heuristics that will determine the relative order of nodes that have the same average computed by barycenter). Poranen and Mäkinen explored the utility of such tie-breaking methods for two-sided bipartite graph drawing, finding that such heuristics did indeed improve the average results on various sets of bipartite test graphs [36]. It would be particularly interesting to see how well these heuristics apply to both the weighted and unweighted cases for one-sided bipartite graph drawing.
7. It would also be interesting to modify the population generation and selection phases of our genetic algorithms to discount repeated permutations. This could, as

www.manaraa.com

noted by Khan [41] potentially lead to a wider more varied searching of the solution space. In turn the average crossing results may improve, and possibly even the running times (for instance if it results in the GA's taking fewer iterations to converge) despite the extra computational effort in checking for repeat permutations.

8. It would be worthwhile to repeat the bottleneck crossing tests with a version of BSHC that is more optimized. Specifically, a modified version of the sub-procedure for counting subgraph edge crossings in stochastic hill climbing could cut down the runtime of BSHC. Rather than recalculating the edge crossings of the graph as a whole during each iteration, bottleneck stochastic hill climbing could limit its calculations to a smaller subgraph and thus save a considerable amount of runtime. The runtime of OMEC could also be improved by making use of the updating swap procedure of MEC, rather than the simpler Edge_Count_Simple method for keeping track of individual edge crossings.

www.manaraa.com

Appendix

The tables that follow fully summarize the data obtained through the experiments outlined in Chapter 4. Each table presents the average performance results for algorithms in specific graph drawing contexts and for varying graph sizes (20, 25, or 30). In the case of weighted and unweighted crossing minimization, data are given in the same tabulated form with a column set for each algorithm. Column data is given in rows (corresponding to specific graph densities and separated by horizontal lines) for both average results and standard deviations in results. Columns for the top 2 or 3 performing algorithms in unweighted and weighted crossing minimization are bolded to emphasize their rankings.

Within each row of a column, the average number of edge crossings, time (in seconds) and percentage deviation of an algorithm from the lower bound are given in said order and separated by vertical lines. So for instance, on random unweighted bipartite graphs of size 20 and density 10% the genetic algorithm of Mäkinen and Sieranta produced layouts, with 209.4 crossings in 1.4 seconds on average. Its average percentage deviation above the lower bound was 16.3%. Its average associated standard deviations were 9.5, 0.4, and 5.5 respectively.

Following weighted and unweighted crossing minimization result tables are those comparing the results of weighted (BC_W) versus the traditional unweighted (BC) barycenter heuristic. Since both heuristics are essentially deterministic their average standard deviations in results (which are all 0.0) are not given. The average weighted

www.manaraa.com

crossings, time, and percent from optimality are presented in the same format as in the preceding tables. Results for graphs of size 20, 25 and 30 are all given in the same table.

The final set of tables at the end of the appendix give the full results for one-sided bottleneck crossing minimization with and without barycenter preprocessing. Average bottleneck crossings and time in seconds of each algorithm are given for each graph size and density with standard deviations in parentheses. So for instance on graphs of size 20 and density 10 percent OMEC resulted in an average bottleneck crossing value of 27.8 with a standard deviation of 0.3. The corresponding average time was 0.4 seconds with an average standard deviation of 0.0 seconds.

Unweighted One-Sided Crossing Minimization Test Results:

Average Results: Unweighted Graphs, Size = 20						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**186.1|0.0|3.0**	209.4|1.4|16.3	233.1|1.4|30.1	192.2|1.2|6.5	**184.1|0.8|1.8**	191.3|0.0|5.8
20%	**968.5|0.0|1.2**	1052.3|2.3|8.8	1117.2|2.4|15.6	1003.5|2.0|3.7	**976.8|1.9|0.9**	1015.3|0.0|4.8
30%	**2449.6|0.0|0.6**	2584.4|3.1|5.9	2692.4|4.8|10.4	2501.3|2.7|2.5	**2450.2|3.0|0.4**	2529.7|0.0|3.6
Average Standard Deviations						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**0.0|0.0|0.0**	9.5|0.4|5.5	10.2|0.4|5.8	2.8|0.0|1.6	**2.1|0.2|1.2**	0.0|0.0|0.0
20%	**0.0|0.0|0.0**	24.5|0.7|2.6	29.4|0.7|3.1	7.3|0.0|0.8	**4.0|0.4|0.4**	0.0|0.0|0.0
30%	**0.0|0.0|0.0**	45.2|0.9|1.9	47.2|1.4|1.9	13.0|0.0|0.5	**5.0|0.6|0.2**	0.0|0.0|0.0

www.manaraa.com

Average Results: Unweighted Graphs, Size = 25						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**501.5|0.0|2.0**	564.3|2.7|15.8	631.6|2.1|29.9	523.3|1.8|7.2	**496.8|1.2|1.7**	516.1|0.0|5.6
20%	**2576.2|0.0|0.9**	2765.1|6.4|8.2	2946.9|3.4|15.4	2654.7|3.3|3.8	**2575.1|2.8|0.7**	2667.0|0.0|4.3
30%	**6361.9|0.0|0.5**	6669.4|9.1|5.3	6981.0|7.2|10.3	6499.9|2.9|2.6	**6356.4|4.3|0.4**	6544.4|0.0|3.3
Average Standard Deviations						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**0.0|0.0|0.0**	22.2|0.9|4.6	21.8|0.6|4.5	6.9|0.0|0.5	**4.0|0.3|0.8**	0.0|0.0|0.0
20%	**0.0|0.0|0.0**	57.5|2.0|2.2	65.7|1.0|2.6	16.2|0.0|0.6	**6.9|0.6|0.3**	0.0|0.0|0.0
30%	**0.0|0.0|0.0**	86.4|2.8|1.4	108.1|1.9|1.7	29.4|0.2|0.5	**10.4|0.9|0.2**	0.0|0.0|0.0

Average Results: Unweighted Graphs, Size = 30						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**1177.4|0.0|1.9**	1301.0|3.9|14.2	1451.7|4.3|27.7	1216.8|1.8|6.7	**1158.5|2.4|1.0**	1202.4|0.0|5.4
20%	**5610.5|0.0|0.7**	5984.5|6.9|7.6	6403.6|7.0|15.1	5779.8|4.9|3.9	**5600.1|5.3|0.6**	5779.4|0.0|3.8
30%	**13776.7|0.0|0.3**	14455.5|10.1|5.3	15102.0|10.1|10.0	14080.4|4.3|2.6	**13771.7|8.6|0.3**	14115.0|0.0|2.8
Average Standard Deviations						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**0.0|0.0|0.0**	43.0|1.3|3.8	41.2|1.2|3.6	13.2|0.2|1.2	**6.2|0.6|0.5**	0.0|0.0|0.0
20%	**0.0|0.0|0.0**	114.7|2.3|2.1	111.4|1.9|2.0	34.3|0.0|0.6	**13.4|1.2|0.2**	0.0|0.0|0.0
30%	**0.0|0.0|0.0**	196.0|3.5|1.4	203.5|2.8|1.5	58.3|0.2|0.4	**18.6|1.8|0.1**	0.0|0.0|0.0

Weighted One-Sided Crossing Minimization Test Results:

www.manaraa.com

Average Results: Weighted Graphs, Size = 20						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	1737.3|0.0|26.7	1708.1|3.3|22.9	1930.3|3.1|40.1	**1537.3|0.8|10.5**	**1429.2|0.7|2.3**	**1478.4|0.0|5.9**
20%	9057.8|0.0|11.0	8988.9|5.9|10.1	9696.8|5.2|19.0	**8551.5|1.4|4.6**	**8254.5|1.4|1.0**	**8613.6|0.0|5.3**
30%	22410.5|0.0|8.5	22199.3|8.1|7.1	23342.2|4.2|12.7	**21385.9|1.9|3.2**	**20840.1|2.1|0.5**	**21545.0|0.0|4.0**
Average Standard Deviations						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	0.0|0.0|0.0	92.3|1.0|6.8	104.9|0.9|7.9	**28.3|0.1|2.1**	**15.0|0.2|1.1**	**0.0|0.0|0.0**
20%	0.0|0.0|0.0	256.3|1.8|3.2	277.0|1.5|3.4	**78.3|0.1|1.0**	**32.6|0.3|0.4**	**0.0|0.0|0.0**
30%	0.0|0.0|0.0	441.6|2.6|2.2	525.1|1.2|2.5	**131.1|0.1|0.6**	**50.9|0.4|0.2**	**0.0|0.0|0.0**

Average Results: Weighted Graphs, Size = 25						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	5129.0|0.0|18.5	4941.9|5.5|18.4	5589.9|2.5|34.4	**4563.6|1.3|9.2**	**4285.2|1.6|2.3**	**4439.3|0.0|5.9**
20%	24055.0|0.0|9.0	24005.7|10.6|9.5	25825.1|7.8|17.9	**22926.6|2.6|4.5**	**22136.9|3.3|0.9**	**22920.39|0.0|4.5**
30%	58084.5|0.0|6.3	57717.9|14.7|6.5	60825.8|6.3|12.3	**55922.1|3.5|3.1**	**54473.0|4.8|0.5**	**56334.43|0.0|3.9**
Average Standard Deviations						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	0.0|0.0|0.0	220.0|.8|5.4	219.3|0.7|5.4	**65.4|0.1|1.6**	**33.1|0.4|0.8**	**0.0|0.0|0.0**
20%	0.0|0.0|0.0	584.6|3.4|2.7	603.2|2.2|2.8	**172.4|0.3|0.8**	**70.3|0.7|0.3**	**0.0|0.0|0.0**
30%	0.0|0.0|0.0	931.5|4.7|1.7	1064.8|1.8|2.0	**298.3|0.4|0.5**	**99.6|1.1|0.2**	**0.0|0.0|0.0**

www.manaraa.com

Average Results: Weighted Graphs, Size = 30						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	10975.3|0.0|15.3	11271.3|8.5|16.8	12765.0|6.4|32.6	**10472.3|2.1|8.4**	**9867.8|2.9|2.0**	**10248.8|0.0|5.9**
20%	52250.9|0.0|8.0	52521.0|16.4|8.8	56703.0|6.4|17.5	**50475.5|5.4|4.5**	**48676.8|9.2|0.8**	**50339.5|0.0|4.2**
30%	125794.6|0.0|5.7	126650.9|23.2|6.3	133654.5|8.9|12.2	**122904.1|7.4|3.2**	**119655.2|14.1|0.4**	**123098.4|0.0|3.3**
Average Standard Deviations						
Density	**Barycenter**	**Makinen_GA**	**Zoheir_GA**	**Hybrid_SA**	**SHC**	**3-WOLF**
10%	**0.0|0.0|0.0**	414.4|2.8|4.4	431.0|1.8|4.5	**123.8|0.2|1.3**	**60.6|0.7|0.6**	**0.0|0.0|0.0**
20%	**0.0|0.0|0.0**	1126.3|5.9|2.3	1185.7|1.9|2.5	**322.4|0.0|0.7**	**132.6|2.0|0.3**	**0.0|0.0|0.0**
30%	**0.0|0.0|0.**	1975.4|8.2|1.7	2058.7|2.5|1.7	**559.9|0.0|0.5**	**192.8|3.0|0.2**	**0.0|0.0|0.0**

Weighted vs. Unweighted Barycenter Results:

	Weighted Size 20 Graphs		**Weighted Size 25 Graphs**		**Weighted Size 30 Graphs**	
Density	**BC**	**BC_W**	**BC**	**BC_W**	**BC**	**BC_W**
10%	1613.6|0.0|16.7	1737.3|0.0|26.7	4827.7|0.0|11.6	5129.0|0.0|18.5	10482.5|0.0|10.2	10975.3|0.0|15.3
20%	8758.9|0.0|7.4	9057.8|0.0|11.0	23437.3|0.0|6.1	24055.0|0.0|9.0	51058.0|0.0|5.4	52250.9|0.0|8.0
30%	21847.0|0.0|5.7	22410.5|0.0|8.5	57092.6|0.0|4.5	58084.5|0.0|6.3	123808.2|0.0|4.0	125794.6|0.0|5.7

One-Sided Bottleneck Crossing Minimization Results:

www.manaraa.com

	Size 20 Graphs		Size 25 Graphs		Size 30 Graphs	
Density	**OMEC**	**BSHC**	**OMEC**	**BSHC**	**OMEC**	**BSHC**
10%	27.8(0.3)|0.4 (0.0)	22.7(1.0)|0.4 (0.10	47.2(0.6)|1.3 (0.3)	38.2(1.5)|1.1 (0.3)	73.8(0.6)|3.9(0.6)	59.4(2.0)|2.9(0.7)
20%	70.5(0.4)|0.9 (0.1)	57.5(1.8)|0.5 (0.2)	116.4(0.7)|3.5 (0.5)	97.4(2.6)|2.5 (0.7)	172.2(0.8)|10.6 (1.4)	149.2(3.7)|5.2(1.4)
30%	114.4(0.4)|1.6 (0.2)	98.8(2.3)|1.3 (0.4)	186.8(0.7)|5.7 (0.8)	164.9(3.1)|3.2 (0.9)	276.5(0.7)|17.1 (2.0)	250.9(4.8)|3.4(1.1)

	Size 20 Graphs		Size 25 Graphs		Size 30 Graphs	
Density	**BC+OMEC**	**BC+BSHC**	**BC+OMEC**	**BC+BSHC**	**BC+OMEC**	**BC+BSHC**
10%	23.8(0.2)|0.1 (0.0)	23.0(1.0)|0.1 (0.1)	40.9(0.2)|0.3 (0.1)	37.7(1.3)|0.3 (0.1)	64.0(0.3)|1.0(0.3)	58.5(1.8)|0.8(0.3)
20%	63.3(0.5)|0.3 (0.1)	57.7(1.5)|0.7 (0.5)	105.8(0.5)|1.2 (0.2)	95.7(2.0)|0.9 (0.3)	160.5(0.6)|3.4(0.6)	147.5(3.3)|1.8(0.7)
30%	107.8(0.5)|0.6 (0.1)	98.5(2.2)|1.6 (1.1)	177.6(0.6)|2.3 (0.4)	163.1(2.6)|1.3 (0.4)	264.9(0.9)|6.5(0.9)	248.2(4.0)|2.6(0.8)

www.manaraa.com

References

[1] M.R. Garey, D.S. Johnson. *Computers and Intractability - A Guide to the Theory and Practice of NP-Completeness*. W.H. Freeman, San Francisco, 1979

[2] F. Shahrokhi et. al. "On Bipartite Drawings and the Linear Arrangement Problem," *SIAM J. Comput.*, vol. 30, 2001, pp. 1773-1789.

[3] M. Juvan and B. Mohar. "Optimal Linear Labelings and Eigenvalues of Graphs," *Discrete Appl. Math*, vol. 36, 1992, pp. 153-168.

[4] M.R. Garey and D.S. Johnson. "Crossing Number is NP-Complete," *SIAM J. Algebraic and Discrete Methods*, vol. 4, pp. 312-316, (give date).

[5] P. Eades and N.C. Wormald. "Edge Crossings in Drawings of Bipartite Graphs," *Algorithmica*, vol. 11, 1994, pp. 379-403.

[6] K. Sugiyama et. al. "Methods for Visual Understanding of Hierarchical System Structures," *IEEE Transaction on systems, Man and Cybernetics*, vol. 11, No. 2, 1981, pp. 109-125.

[7] Xiao Yu Li and Matthias F. Stallman. "New Bounds on the Barycenter Heuristic for Bipartite Graph Drawing," *Information Processing Letters*, vol. 82, June 30 2002, pp. 293-298.

[8] M. Jűnger and P. Mutzel. "2-Layer Straight Line Crossing Minimization: Performance of Exact and Heuristic Algorithms," *J. Graph Algorithms Appl.*, vol. 1, 1997, pp. 1-25.

[9] Olca A. Cakiroglu et. al. "Crossing Minimization in Weighted Bipartite Graphs," *Experimental Algorithms*, vol. 4525, Springer Berlin Heidelberg, 2007, pp. 122-135.

www.manaraa.com

[10] A. Yamaguchi and A. Sugimoto. "An Approximation Algorithm for the Two-Layered Graph Drawing Problem," *in Proceedings of 5th Annual Intl. Conf. on Computing and Combinatorics (COCOON '99)*, LNCS, Springer, 1999, pp. 81-91.

[11] C. Demestrescu and I. Finocchi. "Breaking Cycles for Minimizing Crossings," *J. Exp. Algorithms*, vol. 6, no. 2, 2001.

[12] T. Eloranta and E. Makinen. "TimGA: A Genetic Algorithm for Drawing Undirected Graphs," *Divulgaciones Matematicas*, vol. 9, no. 2, 2001, pp. 155-170.

[13] Q.G. Zhang et. al. "Drawing Undirected Graphs with Genetic Algorithms," *Advances in Natural Computation*, vol. 3612, 2005, pp. 28-36.

[14] B. M. M. Neta et. al. "A multiobjective genetic algorithm for automatic orthogonal graph drawing," *in Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011*, Dublin, Ireland, ACM, July 12-16, 2011, pp. 925-932.

[15] B. M. M. Neta et. al. "A fuzzy genetic algorithm for automatic orthogonal graph drawing," *Applied Soft Computing*, vol. 12, no. 4, April 2012, pp. 1379-1389.

[16] H. He et. al. "Genetic algorithms for the 2-page book drawing problem of graphs," *Journal of Heuristics*, vol. 13, no. 1, Feb 2007, pp. 77-93.

[17] Hiroshi Nagamochi and Nobuyasa Yamada. "Counting edge crossings in a 2-layered drawing," *Information Processing Letters*, vol. 91, no. 5, September 15, 2004, pp. 221-225.

[18] K. Srivastava et. al. "A hybrid simulated annealing algorithm for the Bipartite Crossing Number Minimization Problem," in Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, Hong Kong, 2008, pp. 2948-2954.

www.manaraa.com

[19] Matthew Newton et. al. “Two New Heuristics for Two-Sided Bipartite Graph Drawing,” in *Graph Drawing Lecture Notes in Computer Science*, vol. 2528, Springer Berlin Heidelberg, 2002, pp. 312-319.

[20] Y. Koren and D. Harel. “A multi-scale algorithm for the linear arrangement problem,” in *Proceeding WG ‘02 Revised Papers from the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, London, UK: Springer-Verlag, 2002, pp. 296-309.

[21] E. Mäkinen and M. Sieranta. “Genetic Algorithms for Drawing Bipartite Graphs,” *International Journal of Computer Mathematics*, vol. 53, 1994, pp. 157-166.

[22] Z. Ezziane. “Experimental Comparison between Evolutionary Algorithm and Barycenter Heuristic for the Bipartite Drawing Problem,” *J. Computer Science*, vol. 3, no. 9, 2007, pp. 717-722.

[23] M. Newton et. al. “A Parallel Approach to Row-Based VLSI Layout using Stochastic Hill Climbing,” *Developments in Applied Artificial Intelligence*, Springer Berlin-Heidelberg, vol. 2718, 2003, pp. 750-758.

[24] P. Eades and D. Kelly. “Heuristics for reducing crossings in 2-layered networks,” *Ars Combinatoria*, vol. 21, 1986, pp. 89-98.

[25] C. Matuszewski et. al. “Using sifting for k-layer straightline crossing minimization,” in *7th International Symposium on Graph Drawing (GD’99)*, LNCS 1731, Springer, 1999, pp. 217-224.

[26] S. Bhatt and F. Leighton. “A framework for solving VLSI graph layout problems,” *Journal of Computer and System Sciences*, vol. 28, 1984, pp. 300-343

www.manaraa.com

[27] S. Gupta and M. Stallman. “Bottleneck crossing minimization in layered graphs,” Dept. Computer Science, NC State University, North Carolina, Rep. 13, 2010.

[28] Henry S.H. Chung et. al. “An Optimized Fuzzy Logic Controller for Active Power Factor Corrector Using Genetic Algorithm” in *The Practical Handbook of Genetic Algorithms: Applications*, 2nd ed. Boca Raton, FL, CRC Press, 2000, ch. 11, sec. 11.3.4, pp. 376.

[29] M. Stallman. “A heuristic for bottleneck crossing minimization and its performance on general crossing minimization: Hypothesis and experimental study,” *Journal of Experimental Algorithmics*, vol. 17, 2012.

[30] U. Brandes et. al. “Visualizing related metabolic pathways in two and a half dimensions” (long paper), in *Proceedings of Graph Drawing (GD ’03)*, LNCS, Springer, 2004, pp. 111-122.

[31] U. Brandes and D. Wagner. “Analysis and Visualization of Social Networks,” in *Graph Drawing Software*, Springer, 2003, pp. 321–340.

[32] M. Forster. “Applying crossing reduction strategies to layered compound graphs,” in *Proceedings of Graph Drawing (GD ’02)*, LNCS, Springer, 2002, pp. 276-284.

[33] Brian P. Flemming et. al., “Minimization or Maximization of Functions,” in *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, ch. 10, sec. 9, pp. 445-447.

[34] Zahra Karimi-Dehkordi et. al., “Sudoku Using Parallel Simulated Annealing”, in *Advances in Swarm Intelligence*, Springer Berlin Heidelberg, 2010, ch. 60, pp. 461-467.

www.manaraa.com

[35] Xiaofang Pei et. al., "Application of Simulated Annealing Algorithm in Fingerprint Matching," in *Applied Informatics and Communication*, Springer Berlin Heidelberg, 2011, ch. 5, pp. 33-40.

[36] Timo Poranen and Erkki Makinen. "Tie Breaking Heuristics for the Barycenter and Median Algorithms," unpublished.

[37] Web site for test graphs: Available: http://math.nist.gov/MatrixMarket/

[38] M. Jűnger et. al. "A polyhedral approach to the multi-layer crossing minimization problem," in *Graph Drawing*: *5th International Symposium, GD '97, Rome, Italy, September 18-20, 1997. Proceedings*, Springer, 1997, ch. 2, pp. 13-24.

[39] V. Valls et. al. "A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs," *European Journal of Operational Research*, vol. 90, 1996, pp. 303-319.

[40] B. Smith and S. K. Lim, "QCA channel routing with wire crossing minimization," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI, GLSVLSI '05*, 2005, pp. 217-220.

[41] Salabat Khan et. al. "A Solution to Bipartite Drawing Problem Using Genetic Algorithm," in *Advances in Swarm Intelligence*, Springer-Verlag Berlin Heidelberg, 2011, ch. 63, pp. 530-538.

www.manaraa.com