Graph Drawing Heuristics for Path Finding in Large Dimensionless Graphs

Tim Weninger, Rodney R. Howell, William H. Hsu
Department of Computing and Information Sciences
234 Nichols Hall
Kansas State University
Manhattan, KS, 66506
{weninger, rhowell, bhsu}@ksu.edu

Abstract— *This paper presents a heuristic for guiding A*-search for approximating the shortest path between two vertices in arbitrarily-sized dimensionless graphs. First we discuss methods by which these dimensionless graphs are laid out into Euclidean drawings. Next, two heuristics are computed based on drawings of the graphs. We compare the performance of an A*-search using these heuristics with breadth-first search on graphs with various topological properties. The results show a large savings in the number of vertices expanded for large graphs.*

Keywords: search, heuristics, large graphs, graph drawing

1. Introduction

The structure of interconnections among a set of entities is an increasingly interesting topic because of the important information that can be used to infer latent information about the entities themselves, relationships among entities or the network as a whole. Examples of such graph structures include the World-Wide Web, the Internet, social networks, traffic systems, etc. Recently, advances in this area have led to the identification of web sites that form a knowledge base on a common subject [1], [2], [3]. The famous PageRank [4] and HITS [5] algorithms similarly use network structure to determine the relative importance of graph entities (web pages).

We address the problem of finding relationships between two vertices in an explicitly presented arbitrary graph with unweighted edges. Given such a graph G with a start vertex s and a goal vertex γ the *shortest path problem* is to find the path from s to γ where the number of edges is minimized.

The time complexity of the uninformed general search strategy (e.g. breadth-first search) is in $\mathcal{O}(|E| + |V|)$ where $|E|$ is the number of edges and $|V|$ is the number of vertices in G, whereas the space complexity is in $\mathcal{O}(|V|)$. Since $|V|$ is potentially much larger than the diameter of the graph, breadth-first search is often impractical for large graphs.

To solve this problem, latent information can be leveraged in order to reduce the search frontier and therefore reduce the search time. These informed search strategies use heuristic information to appraise the merit of every candidate search avenue that is encountered during the search. The informed search then continues along the path of highest merit. This strategy exists within many combinatorial problems arising in network analysis, scheduling, packet routing, game playing, etc.

By far, the most studied adaptation of informed search is the A* algorithm, which defines the cost of a path as the sum of the costs of its arcs. Formally, A* considers the function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the current path from s to n and $h(n)$ is the heuristic estimate of the cost of the path from n to a γ. In addition, because A* requires $h(n)$ to be admissible (i.e. a lower bound to the actual cost of any path from n to γ), $f(n)$ is an optimistic estimate of all possible solutions stemming from the current path. Therefore an optimal path can be found by terminating the search immediately upon reaching a goal node.

Common heuristics (e.g. straight-line distance, manhattan distance) can only be computed in graphs where its vertices correspond to points in a Euclidean space. The novel contribution of our work is the application of these heuristics in *dimensionless* graphs.

Definition 1. *A graph is dimensionless if its vertices do not correspond to points in a Euclidean space.*

We will show that our heuristics that the admissibility of a heuristic depends on a graph's topology, and therefore, technically, our algorithm should be called simply A if the heuristic is not an underestimate of the actual cost. However, we will continue to call it A* because the implementation is the same and similar papers do not distinguish A from A*.

The specific motivation for this research is for the analysis of social networks wherein links between users denote friendship [6], [7], [8]. Such networks are easily represented as dimensionless graphs because social networks have no need to assign (x, y) coordinates to each user. Because of the large number of social network users, breadth-first search is impractical. As an illustration, the relatively small social network *LiveJournal* contains 18.8 million users with an average of 76.69 directed links (mutual friendship counts as 2 links) per user (cf. [7]) resulting in 1.4 billion links. To find the shortest path between users in a candidate pair $\langle u, \gamma \rangle$ breadth-first search would therefore potentially examine 1.4 billion edges and use about 150 MB of memory per query. Alternatively, to calculate all pairs shortest paths (APSP) with the Floyd-Warshall algorithm would use $\mathcal{O}(|V|^2)$ space and $\mathcal{O}(|V|^3)$ time, or over a petabyte of memory and 6.6×10^{21} computations, not to mention the disk space needed to store the results. We can see from these simple estimations that traditional graph-search approaches are impractical even on the most advanced computing platforms.

To solve the impracticality of traditional search techniques, we will first generate a two-dimensional Euclidian drawing of the graph. From that drawing, (x, y) coordinates can be found and potential search heuristics emerge. This work studies two such graph drawing algorithms: the Spring Embedder (SE) method, and the High Dimensional Embedding (HDE) method. Using these two graph drawing methods, we then consider two heuristics: straight-line distance, and angle deviation (described in later sections). Next, we describe the experiments used to evaluate the different graph drawing methods in concert with the two heuristics. We then present the results and compare them with traditional methods where applicable. Finally, we offer conclusions and recommendations for future work.

2. Related Work

General search, including pathfinding, is arguably the most basic concept in field of artificial intelligence [9]. Uninformed search techniques (e.g. Dijkstra's algorithm, the Bellman-Ford algorithm, Floyd-Warshall algorithm, breadth-first, depth-first, depth-limited) have been rigorously studied, and their implementations appear in a wide-range of domains. [10]

Unfortunately, most uninformed search strategies are inefficient. A more practical search strategy uses problem-specific knowledge beyond the graph itself in order to find solutions. The key component to these informed strategies is the heuristic function. In some domains an admissible (or otherwise descriptive) heuristic is readily obtained. For example, when finding paths on street or railway networks the distance from the current point to the destination can be used as a heuristic [11].

There are several approaches offered by researchers in path finding and related fields that attempt to shorten the time to compute paths within their specific domains. Wagner and Willhalm [12] were able to reduce the search space to nearly 10% of the original. Hershberger and Suri [13] found an optimal $\mathcal{O}(n \log n)$ algorithm for finding Euclidean shortest paths in the presence of obstacles, where n is the number of vertices in all of the obstacle-polygons, using the shortest path map approach [14] and the continuous Dijkstra method [15], [16].

More closely related is the work by Sedgewick and Vitter [17] on shortest paths in Euclidean graphs, which starts with a Euclidean graph in 2 dimensions and extends it to d dimensions (where each vertex corresponds to a point in d-dimensional Euclidean space $\mathbb{R}^d$). They set the weight of each edge $u \to v$ to be proportional to the straight-line distance between u and v. As a result they show an improvement over Dijkstra's running time of $\mathcal{O}(|E| + |V| \lg |V|)$ to $\mathcal{O}(|V|)$.

The overarching goal of the related research has been to use latent information within graphs to reason about the paths within it in order to make informed decisions. In all cases the heuristics providing the information were based on a Euclidean drawing of the graph. Most often the straight-line distance heuristic (distance from current vertex n to the goal vertex γ) was considered. However, this distance heuristic was gathered from a reliable source (e.g. map data, latitute/longitude). Our goal is to operate with a similar efficiency in lieu of this data (*i.e.* in dimensionless graphs). To accomplish this, we must generate the heuristic data by creating 2-D Euclidean drawings from the dimensionless graphs. The following section on graph drawing describes the two approaches used in this work.

3. Graph Drawing

A graph $G(V, E)$ is an abstract structure used to model relationships among entities; the entities are represented by a set of vertices V and the relationships are represented by the edges $E \subseteq V \times V$. Most graphs are intended to be comprehended by humans; therefore, the usefulness of a graph can be determined by the clarity of its layout.

Definition 2. *A layout of a graph $G(V, E)$ is a mapping of the graph's vertices to the two dimensional Euclidean space: $X : V \to \mathbb{R}^2$.*

Typically, the clarity of a graph drawing is usually defined by how many edge crossings there are. Of course, not all graphs are planar graphs, so it is impractical to ask for 0 edge crossings in all cases. However, most graph drawing algorithms operate to minimize the number of edge crossings, and maintain uniform edge lengths.

In this work we test two graph drawing algorithms. They are described below.

3.1 Spring Embedder Method

The most common approach from drawing graphs is the spring embedder (SE) method. Algorithms based on this approach are made up of two parts: (1) the heuristic force model, and (2) the optimization algorithm. The resulting layout brings the system to equilibrium in which the total force on each vertex is within ϵ of 0. SE methods always draw straight line segments so the crux of the method is reduced to positioning the vertices.

The most straightforward method is to assign forces as if the edges were springs that act in accordance with Hooke's Law: every two vertices are connected by a spring, whose rest length is proportional to the graph-theoretic distance between its two endpoints. Furthermore the vertices are assigned forces in accordance with Coulomb's Law: the force between two points is inversely proportional to the square of the total distance between the two charges. With the forces set, the graph is simulated with the forces pushing and pulling on one another. This is repeated until the system comes into an equilibrium state at which point location of the vertices are recorded (i.e. the graph is drawn). An example graph drawing via the Spring-embedder method is shown in Figure 1. This graph, as well as all graphs presented in this paper, was drawn with Tulip 3.1.1 [18].

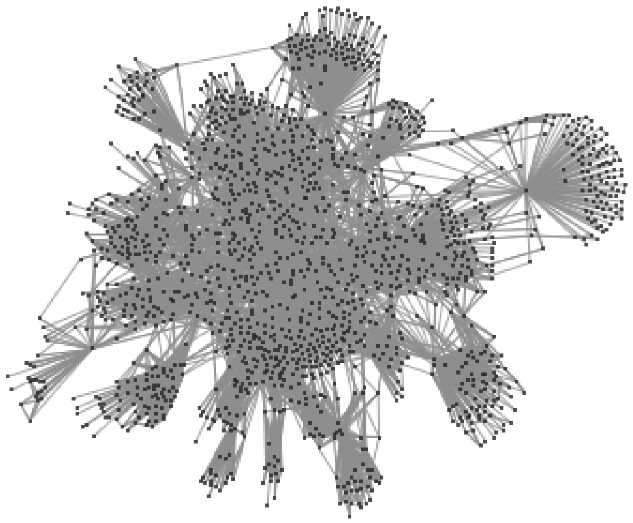

Fig. 1: 2000-vertex subset of *LJGraph* drawn using spring embedding (SE).

There are several disadvantages with the SE method. Most important is the time complexity: in the general case, force-directed algorithms are known to have a running time of $\mathcal{O}(|V|^3)$. The SE method is also susceptible to poor local maxima because a state with very low energy does not necessarily correspond to the the optimal energy state. Moreover, the drawings are strongly influenced by the initial layout, which is usually random. [19], [20], [21], [22]

3.2 High Dimensional Embedding

Drawing a graph so as to minimize the number of edge crossings while maintaining other aesthetic goals cannot be practically achieved in a low dimension, "due to the fact that several aesthetic goals have to compete on a shared limited space" [23]. Therefore, it is important to perform the initial drawing work in many dimensions in order to conserve space and make the entire task easier.

The high dimensional embedding (HDE) approach suggested by Harel and Koren consists of two parts: (1) the graph is embedded in a very high dimension (e.g. in 50 or 100 dimensions) and then (2) projected into a 2-D plane using principle components analysis. An example graph drawing via the HDE method is shown in Figure 2.

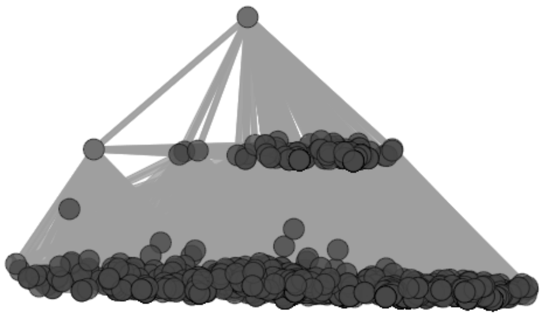

Fig. 2: 2000-vertex subset of *LJGraph* drawn using High Dimensional Embedding (HDE).

The algorithm is very fast, and is able to compute on very large graphs efficiently. For example, a social network graph with 770,595 vertices and 2,992,607 directed edges was rendered on a standard desktop in about 5 seconds. [23]

At this point, judging by the drawn examples shown in Figures 1 and 2, we expect our heuristic search results to be faster and more accurate with the SE method because the graph drawing in Figure 1 better represents the graph drawing principles outlined at the beginning of this section.

4. Methodology

In order to empirically test our path finding approach, we executed several experimental searches with different heuristics on three large real-world graphs and one synthetic graph. This section describes the design of these experiments, while the next section presents the results.

4.1 Datasets

Our experimental dataset contains four graphs. The first, *LJGraph*, is a recent, partial snapshot of the online social network service *LiveJournal*; *LJGraph* contains vertices corresponding to *LiveJournal* users and directed edges corresponding to friendships among the crawled users. The second graph, *WikiGraph* is a subset Wikipedia; *WikiGraph* contains vertices corresponding to Wikipedia entries and directed edges corresponding to the wiki-links between pages. The third real-world graph, *DBLPGraph* is the full list of authors and papers listed at the DBLP Computer Science Bibliography; *DBLPGraph* contains vertices corresponding to authors and undirected edges corresponding to scholarly articles. *GridGraph* is the fourth graph; this synthetic graph was auto-generated to represent a grid (i.e. checkerboard). Table 1 lists the sizes of each graph.

Table 1: Sizes for all graphs in the experimental dataset, and the clustering coefficient ($\overline{C}$)

Graph	Vertices	Edges	$\overline{C}$
LJ	770,595	2,992,607	0.11
Grid	1,000,000	1,998,000	0.00
DBLP	654,628	3,573,312	0.48
Wiki	400,000	3,241,997	0.18

4.2 Layout and Heuristics

Given a graph G, with a set of coordinates $(x_i, y_i) \in C$ (e.g. latitute/longitute, Cartesian points) corresponding to all points $v_i \in V$, we define two heuristics: (1) straight-line distance, and (2) angle deviation. These heuristics are described in the following subsections.

4.2.1 Straight-line distance $h(d)$

With the straight-line distance heuristic $h(d)$, the open list of the search agent is sorted by the distance from the current vertex n to the goal vertex γ. Calculation of this heuristic is performed by the Euclidean distance formula, shown in Equation 1.

$$d = \sqrt{(x_\gamma - x_n)^2 + (y_\gamma - y_n)^2} \tag{1}$$

4.2.2 Angle deviation $h(\Theta)$

An alternative to the straight-line distance is the the angle deviation $h(\Theta)$. The calculation of this heuristic requires three vertices to be known: (1) the current vertex n, (2) the goal vertex γ, and (3) a vertex which is a candidate for relaxation m. From these three vertices two vectors $\langle m, n\rangle$, $\langle n, \gamma\rangle$ are represented as A and B respectively. The angle in question, $\angle mn\gamma$, is computed by Equation 2.

$$\tan(\theta) = \frac{\|A \times B\|}{A \cdot B} \tag{2}$$

From Equation 2, we see that the angle deviation is equal to the arc tangent of the cross product length divided by the dot product.

The θ closest to 0 is the best candidate for relaxation.

These two heuristics are only two among many possible heuristics (cf. [24]). These, and many other, heuristics are admissible in grid-style graphs while they are not necessarily admissible in non-grid-style graphs.

4.3 Experiment Setup

Two sets of tests were executed to determine the performance of our approach.

The first set of tests used only *LJGraph*. The purpose of this set of tests is to demonstrate how the search performance changes as graphs become larger. To that end, we clipped *LJGraph* into 11 subsets with sizes 100, 200, 400 ... 102,400. Three graph sizes were explicitly singled out for further investigation: $A \subseteq G$ with 100 vertices and 277 edges; $B \subseteq G$ with 1,600 vertices and 10,168 edges; $C \subseteq G$ with 102,400 vertices and 1,325,668 edges.

Each subgraph was embedded into a 2-dimensional drawing with the spring embedder (SE) and high dimensional embedding (HDE) methods creating $A \rightarrow A_{SE}$, $A \rightarrow A_{HDE}$, $B \rightarrow B_{SE}$, etc. Note: Subgraph C could not be drawn with the SE method because of space/time complexity limitations.

From each subgraph drawing a random u and γ were chosen and three searches were executed: (1) Breadth-First Search, (2) A*-search on graphs drawn with the SE method, (3) A*-search on graphs drawn with the HDE method. Only the straight-line distance heuristic is used in the first set of experiments. This test was repeated 1,000 times each with random u and γ. This setup makes sure to keep the randomly generated u and γ consistent across the three search methods in order to compare results.

The second set of tests used the full size of all graphs. The purpose of this set of tests is to demonstrate how the search performance changes with different large graphs.

Because the full-size graphs are large the SE method could not be used to draw the graphs; therefore, only the HDE method was used to create drawings of the full-sized graphs.

From each drawing a random u and γ were chosen and 3 searches were executed: (1) Breadth-First Search, (2) A*-Search with distance heuristic, (3) A*-Search with angle deviation heuristic. This test was repeated 1,000 times each with random u and γ. This set up made sure to keep the randomly generated u and γ consistent across the three search methods in order to compare results.

Outcomes of these experiments are described by the metrics described in the next subsection and are presented in the results section.

4.4 Metrics

There are several metrics used to determine the performance of each search method: (1) Mean Number of Relaxations, (2) Percent Correct, (3) Mean Shortest Path Length (SPL), (4) Mean Path Length Error (MPL-ϵ).

The mean number of relaxations is the number of relaxations performed during the search. This metric is used instead of CPU-time because this the number of relaxations could more appropriately be applied to disk-seeks or database lookups in graphs too large to fit into main memory.

The percent correct is the number of times the heuristic approach returns the actual shortest path over the total number of executions.

The mean SPL is the mean path length returned by BFS. This is the correct/acutal SPL.

Finally, the MPL-ϵ is the mean difference between the path length returned by the heuristic approaches and the BFS-SPL (metric #3).

5. Results

This section presents the results obtained by executing the experiments from subsection 4.3 using the metrics described in subsection 4.4. Two sets of experiments were performed. The first tested the scalability of this approach by comparing results of various subsets of the *LJGraph*. The second set of experiments examined the accuracy and savings ratios of the 4 full-sized graphs.

5.1 Scalability

Table 2 contains the results of breadth-first searches on 3 subsets of *LJGraph*. These results are considered the control results for the first set of experiments. The Mean Shortest Path Length (SPL) in Table 2 is the actual shortest path length. The SPL increases as the size of the graph increases because the u and γ are chosen randomly, and the distance between random vertices typically increases as $|V|$ increases.

Table 2: Results from breadth-first searches performed on subgraphs of LJGraph.

Size	SPL	Relaxations
100	1.92	98.66
1600	3.41	818.701
102400	4.31	9886.14

Figure 3 illustrates the relationship between the size of the graph and the mean and median number of relaxations taken to find a path when using the A*-search with the straight-line distance heuristic. Some results for searching on spring embeddeder (SE) method could not be calculated due to time/space constraints. We see that our approach consistently requires less than half the number of relaxations, on average.

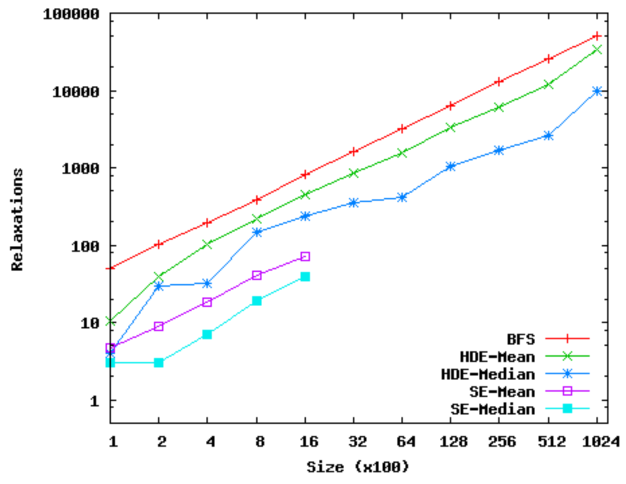

Fig. 3: Comparison of graph size to the number of relaxations in BFS and A*-search with the straight-line distance heuristic.

Table 3 contains the results of the A*-search for subgraphs drawn with both layout methods. As expected, the heuristic search performed reasonably well compared to the control in terms of accuracy. There are significant savings in the number of relaxations. As the graph size increases the savings increase and the accuracy decreases dramatically. As expected, the searches with a spring embedder (SE) method perform better than searches with the high dimensional embedding (HDE) method.

Table 3: Results from searches performed on subgraphs of LJGraph using straight-line and angle deviation heuristics.

		Size	SPL	Relax	MPL-ϵ	Correct
SE	$h(d)$	100	1.94	4.75	.02	98.6%
		1600	4.62	70.85	1.21	40.4%
		102400	-	-	-	-
	$h(\Theta)$	100	1.93	4.08	.01	99.1%
		1600	4.08	39.93	.67	58.1%
		102400	-	-	-	-
HDE	$h(d)$	100	1.99	5.35	.07	97.1%
		1600	6.18	453.13	2.767	26.5%
		102400	9.698	34429	5.62	3.3%
	$h(\Theta)$	200	2.00	7.09	.09	93.4%
		1600	6.38	388.51	2.97	27.6%
		102400	11.34	33979	7.23	5.6%

5.2 Graph Comparisons

The second set of experiments focuses on the differences between the 4 full size graphs. Table 4 contains the results of a breadth-first search on the full graphs. These results are considered the control results.

Table 4: Results from breadth-first searches performed on full graphs

	SPL	Relaxations
LJ	5.27	20026.13
Grid	661.28	500459.94
DBLP	9.55	172107.69
Wiki	9.45	112987.68

Table 5 contains the results of the A*-search for graphs drawn with the HDE method. The SE method is not able to draw these graphs because of time/space complexity limitations.

One result of particular interest is the MPL-ϵ for the *GRIDGraph* with the distance heuristic $f(d)$ in Table 5. The error rate is equal to or almost equal to 0. We believe that this is because the HDE layout of *GRIDGraph* very closely resembles a checkerboard or a grid. Therefore, distance and angle heuristics can accurately guide the A*-search algorithm to the goal.

As in the previous set of experiments, the heuristic searches found paths from u to γ much faster than breadth-first search. However, this speed up is often at the expense of accuracy which fluctuates among graph-types.

Table 5: Results from heuristic searches performed on full-size graphs drawn by the high dimensional embedding (HDE) method using straight-line and angle deviation heuristics.

		SPL	Relaxations	MPL-ϵ	Correct
$h(d)$	*LJ*	7.65	5866.41	2.39	12.3%
	Grid	661.28	236732.77	0.0	100.0%
	DBLP	11.11	56637.80	1.57	40.1%
	Wiki	20.12	97747.48	10.67	14.3%
$h(\Theta)$	*LJ*	13.60	4833.80	8.34	4.4%
	Grid	661.37	253017.65	0.09	96.7%
	DBLP	10.96	59749.61	1.42	49.1%
	Wiki	20.64	132307.91	11.19	17.5%

6. Conclusions

In conclusion, we have described a pathfinding approach that uses heuristics gleaned from Euclidean drawings of otherwise dimensionless graphs. This approach readily operates on graphs with millions of vertices and edges and makes no assumptions about the type or density of the graph. The strength of this approach is shown by time and space savings in terms of the number of relaxations as shown in Table 6.

Table 6: Comparison of the error on the SPL and Savings (in terms of Relaxations) from Tables 4-5 using straight-line and angle deviation heuristics.

		SPL-Error	Savings
$h(d)$	*LJ*	45.16%	70.71%
	Grid	0.00%	52.69%
	DBLP	16.34%	67.09%
	Wiki	112.91%	13.49%
$h(\Theta)$	*LJ*	158.06%	75.86%
	Grid	0.01%	49.44%
	DBLP	14.76%	65.28%
	Wiki	118.41%	-17.10%

Table 6 also shows the weakness of this approach; that is, for some graphs, the error rate may outweigh the savings benefits.

These results suggest that the use of graph drawing techniques may be useful in approximating shortest paths in large explicitly-presented graphs, particularly in applications requiring many different paths from the same graph. However, the goals of graph drawing and path finding are not identical. For example, a requirement of graph drawing is that the graph be embedded in two, or sometimes three, dimensions. While using fewer dimensions reduces the overhead of computing Euclidean distance or angle deviation, the requirement that the graph be embedded in two or three dimensions is not essential for these heuristics.

This observation suggests that it might be worthwhile to try applying our heuristics directly to the high dimensionsal embedding of Harel and Koren. They have observed that 50 dimensions tends to work well for the ultimate purpose of drawing a graph in two dimensions. However, it would seem to be worth trying our heuristics on graphs embedded in fewer dimensions, but more than two. It is not clear at this point where the best balance between the accuracy of the heuristics and their computational costs might occur. One theoretically attractive feature of this this approach is that Harel and Koren have shown bounds relating the distances between nodes in the embedding and their graph-theoretic distances [23]. These bounds might be used to show theoretical bounds on the accuracy of the approximations and the running times of the algorithms.

Another alternative worth considering is the use of other measures of distance. For example, Manhattan distance is a bit easier to compute than Euclidean distance, and it might yield results that are just as accurate.

7. Acknowledgements

This research was partially funded by a grant from the Defense Intelligence Agency. In addition, we would like to thank Jiawei Han, Yizhou Sun and Yintao Yu for their help with the DBLP graph, Henry Haselgrove for publishing the Wikipedia graph, and the Tulip software team.

References

[1] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, “Extracting large-scale knowledge bases from the web,” in *Proceedings of the 25th VLDB Conference*, 1999, pp. 639–650.

[2] M. Kuramochi and G. Karypis, “An efficient algorithm for discovering frequent subgraphs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, pp. 1038–1051, 2002.

[3] J. Abello, M. G. C. Resende, and R. Sudarsky, “Massive quasi-clique detection,” in *In Latin American Theoretical Informatics (LATIN*, 2002, pp. 598–612.

[4] S. Brin, “The anatomy of a large-scale hypertextual web search engine,” in *Computer Networks and ISDN Systems*, 1998, pp. 107–117.

[5] J. M. Kleinberg, “Authoritative sources in a hyperlinked environment,” *Journal of the ACM*, vol. 46, pp. 668–677, 1999.

[6] W. H. Hsu, A. King, M. S. Paradesi, T. Pydimarri, and T. Weninger, “Structural link analysis from user profiles and friends networks: A feature construction approach,” in *ICWSM*, Boulder, CO, 2007, pp. 75–80.

[7] T. Weninger, “Link discovery in very large graphs by constructive induction using genetic programing,” Master’s thesis, Kansas State University, Manhattan, KS, USA, December 2008.

[8] L. Getoor, “Link mining: A new data mining challenge,” *SIGKDD Explorations*, vol. 4, no. 2, pp. 1–6, 2003.

[9] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[10] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, “Shortest paths algorithms: Theory and experimental evaluation,” *Mathematical Programming*, vol. 73, pp. 129–174, 1996.

[11] M. Marathe, N. I. Ona, L. L. Aboratory, R. Jacob, R. Jacob, M. V. Marathe, K. Nagel, and K. Nagel, “A computational study of routing algorithms for realistic transportation networks,” *ACM Journal of Experimental Algorithms*, vol. 6, 1999.

[12] D. Wagner and T. Willhalm, “Geometric speed-up techniques for finding shortest paths in large sparse graphs,” in *ESA*, 2003, pp. 776–787.

[13] J. Hershberger and S. Suri, “Finding a shortest diagonal of a simple polygon in linear time,” *Comput. Geom.*, vol. 7, pp. 149–160, 1997.

[14] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, “Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons,” *Algorithmica*, vol. 2, pp. 209–233, 1987.

[15] J. S. B. Mitchell, “A new algorithm for shortest paths among obstacles in the plane,” *Ann. Math. Artif. Intell.*, vol. 3, no. 1, pp. 83–105, 1991.

[16] ——, “Shortest paths among obstacles in the plane,” *Int. J. Comput. Geometry Appl.*, vol. 6, no. 3, pp. 309–332, 1996.

[17] R. Sedgewick and J. S. Vitter, “Shortest paths in euclidean graphs,” *Algorithmica*, vol. 1, no. 1, pp. 31–48, 1986.

[18] D. Auber, “Tulip : A huge graph visualisation framework,” in *Graph Drawing Softwares*, ser. Mathematics and Visualization, P. Mutzel and M. Jünger, Eds. Springer-Verlag, 2003, pp. 105–126.

[19] T. Kamada and S. Kawai, “An algorithm for drawing general undirected graphs,” *Inf. Process. Lett.*, vol. 31, no. 1, pp. 7–15, 1989.

[20] ——, “Automatic display of network structures for human understanding,” Department of Information Science, Tokyo University, Technical Report 88-007, February 1988.

[21] P. Eades, “A heuristic for graph drawing,” *Congressus Nutnerantiunt*, vol. 42, pp. 149–160, 1984.

[22] T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” *Softw., Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, 1991.

[23] D. Harel and Y. Koren, “Graph drawing by high-dimensional embedding,” in *In GD02, LNCS*. Springer-Verlag, 2002, pp. 207–219.

[24] A. Patel, “Heuristics,” 2008, http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html.