Graph Drawing Contest 2020
Crossing Minimization with Randomness

Sebastian Benner

Universität Kassel FB 16

Abstract. Drawing graphs with the minimal amount of edge-crossings is a NP complete problem. Additional constraints placed upon the drawing, namely being a straight-line upwards drawing, do not reduce this complexity. To approximate a solution for this problem within a certain time frame and with the least amount of edge-crossings possible, a multi-step algorithm based mostly on random displacements is proposed. While some limitations are necessary, the best results were achieved by only preventing erroneous positions and only the most bare bone limitations. In addition the work contains some pointers for further improvements and adaptation to different constraints placed upon the drawing.

1 Introduction

Calculating the minimal number of crossings present in all possible drawings of a graph is a hard problem. In [5] it was shown to be NP-complete. Likewise, it is equally as hard to find such a drawing.
To circumvent this problem, algorithms are designed to find drawings with close to if not equal to the minimal amount of crossings. The annual Graph Drawing Contest[1] is an open challenge to design such an algorithm for optimized graph drawings. The exact criteria for such a drawing are changed every couple of years, but both the current and last contest aimed for minimal crossings. The Live Challenge will contain between five to ten acyclic directed graphs with up to a few thousands nodes each. All resulting layouts must be submitted within one hour of the graphs being handed out.
Besides the main criterion additional constraints are placed on the drawings:

- Each edge must be a straight upward facing line, meaning the source of each directed edge must be lower than the target.
- Each node must be placed upon a grid of given size.
- Crossings between a node and an edge are not permitted, as well as overlapping nodes.

The constraints do not change the complexity of the problem. In [3] it was shown that determining whether or not a directed acyclic graph even allowed a upwards drawing without any crossings is already NP-complete - a problem that can be solved more easily for undirected graphs.

[1] http://mozart.diei.unipg.it/gdcontest/contest2020/challenge.html

After all resulting graphs are collected, for each of the original graphs a best drawing is determined with all the other graphs receiving a weighted score based on the difference in edge-crossings. The highest overall score wins the contest. Each team has to bring its own hardware to run their respective algorithm, meaning there is no limitation in terms of tools used and the given time to solve the task can be counterbalanced by more powerful hardware.
For at least three consecutive years the winning algorithm was based on randomness. The aim of this work is to iterate on these winning entries and to provide a lightweight algorithm much more focused on the random aspect than before.
The rest is structured as follows: Section 2 and 3 will go through the notation needed and related work respectively. Afterwards, section 4 will propose the algorithm with an evaluation in the next chapter. What follows is a section containing both failed approaches as well as pointers for follow-up work not bound to the constraints of the contest. Finally, section 6 will give a conclusion.

2 Mathematical Foundation

A *graph* G is defined as an ordered pair (V, E) of *vertices* V and *edges* E. Edges are unordered pairs of two vertices $\{x, y\}$, said to *join* them, and therefore E is a subset of $\binom{V}{2}$. In the special case where edges are ordered pairs (x, y) with x as *source* and y as *target* the graph D is called *directed* graph or *digraph*. We call the number of vertices the *order* of G and the number of edges the *size* of G. $V(G)$ and $E(G)$ are the sets of vertices and edges of G respectively, $x \in V(G)$ with vertex x can be written as $x \in G$ while $\{x, y\} \in E(G)$ with undirected edge $\{x, y\}$ is written as $\{x, y\} \in G$. For a more comprehensive explanation I refer to the book *Modern Graph Theory* [2] upon which this notation is based on.
A *drawing* is a visual representation of a graph through a *layout*, which itself is a mapping $V \longrightarrow X_1 \times X_2$ with $X_1, X_2 \subset \mathbb{N}$ as two-dimensional coordinates (x_1, x_2). Directed edges are drawn as straight arrows between their respective nodes pointing from the source to the target. A drawing is called *upwards* if $x_{2_{target}} > x_{2_{source}} \forall (target, source) \in E$. A *node-overlap* is defined by two nodes having the same coordinates. *Edge-crossings* and *node-crossings* occur when a edge is drawn through another edge or a node respectively.

3 Related Work

There have been many works on minimizing the crossings in a drawing. This section will go through the most significant contributions to the algorithm presented here, many of them bear a connection to the contest, which this work is aimed at.
While not aiming at the same goal, the algorithm in [1] served as a basis for the last years winner. The aim was to create a drawing with the largest minimum angle, called the resolution of G, possible. To achieve this the

algorithm build and maintained two sets: all nodes and only such nodes that are deemed critical for the current resolution of G. Critical are all nodes connected with edges involved in minimal angles. In each iteration either a node from the set of all critical nodes is chosen uniformly or inverse proportionally to its proximity to a critical nodes from the set of all nodes. To determine the new position of a node rays are used which are cast out uniformly distributed in all directions with the best result becoming the new position. Combined with a energy-based base drawing and some tweaks to avoid local minima, the algorithm proved itself and also won its respective year.
Another winning contribution to a different year with the same aim as before is found in [4]. An algorithm to create a drawing with the largest possible minimum angle. Starting from a force directed base drawing the algorithm is greedily selecting the minimal crossing from the current drawing. A random vertex is chosen and displaced to the optimal position within a square around the original position to achieve the maximal local crossing angle. The size of the square is decreased with each iteration.

4 Proposed Algorithm

The following algorithm is a multi-step procedure. The first step is to apply a base drawing, which is chosen to achieve the best results in the least amount of time. While it is not needed, starting from a local minima reduces the iterations for the second step. Said step consists of random displacements to produce improvements or no changes at all for a single randomly chosen node each iteration. The second step was designed to run as efficient as possible for the maximum amount of displacements within the time frame.

4.1 Base Drawing

The base drawing used in this algorithm is the *Sugiyama Framework* [6]. The algorithm works as follows:

1. All cycles are removed. As the graphs in this particular contest are directed and guaranteed to have an upwards drawing, this step can be skipped.
2. The nodes are sorted into different layers and dummy nodes are placed wherever an edge crosses a layer. Again using the upwards facing edges it is rather simple so solve this step for the graphs at hand.
3. The nodes are moved around to minimize the crossings between two layers. This is repeated several times for the whole graph until a local maxima is found or the maximal number of iterations is completed.
4. Finally the dummy nodes are removed again.
5. As an extra step the resulting drawing is either stretched or compressed to match the width and height of the grid.

While other base drawings were tested, for this specific set of requirements Sugiyama yielded the best base drawing within the minimal amount of time.

4.2 Crossing Minimization

Following the base drawing, the iterative graph drawing algorithm is heavily based on a repeated completely random displacement of nodes. In order to achieve the best results there are three things to balance against each other: Being as efficient as possible to run as many displacements as possible, directing the algorithm enough to minimize steps without any improvements and allowing enough randomness to minimize the risk of local minima. What follows now is a step by step explanation of the final algorithm, afterwards I will go through possible changes and their impact on the overall results.

0. The displacement process is started with a fix number of iterations I. Each iteration represents only a single displacement for a single node.
1. A node is chosen completely random and the following values are determined:
   (a) The initial number of node-overlaps is calculated by comparing the node position with every other node.
   (b) To arrive at the number of node-crossings the node position is tested against every edge. If the node is within the bounding box of an edge, the more complex crossing test is performed.
   (c) Finally edge-crossings are calculated by testing each edge connected with the node against every other edge, again only by testing if their bounding boxes intersect and testing for an edge-crossing afterwards.
2. If all metrics equal 0 the iteration is stopped prematurely.
3. The position of the node is randomized within a box around the original position. In each direction the node can move a randomized distance of up to 10% of the grid size in the respective dimension. Afterwards the new position is corrected to be within the in the box an between the all upper and lower nodes to assure the upward facing attribute of all edges. The maximal distance is introduced to prevent nodes from jumping across the entire width which would most likely never decrease the number edge-crossings coming from an initial Sugiyama drawing. Although the chosen box may still seem too large, it performed the best on average between all possible grid sizes.
4. Afterwards each metric is calculated again.
   (a) If both the number of node-crossings and node-overlaps remain the same or improve compared to before, the new position is kept if the same applies to the number of edge-crossings.
   (b) If both the number of node-crossings and node-overlaps remain the same or improve compared to before, the new position is kept with an increase in edge-crossings up to $n = 4$ with a decreasing chance between 100% and 0% in the first $I/(n + 1)$ steps. This means an increase of 1 can be kept in the first half, a increase of 2 in the first third and so on.
   (c) Regardless of the edge-crossings the new position is kept iff the number of node-overlaps is reduced or the number of node-crossings decreases while the node-overlaps stay the same. This

priority assures that the resulting drawing almost certainly does not violate any restrictions.

(d) Otherwise the displacement is reversed.

5 Evaluation

The algorithm was implemented in C++ using the *Open Graph Drawing Framework*[2] for the base drawing. In order to increase the performance of the edge-crossings algorithm, it was run in parallel using *OpenMP*[3]. Each edge connected to the current node was evaluated separately and the resulting values were summed up. Across all tested configurations OpenMP only cut the runtime in half as all displacements still run in sequence and the average node has not enough edges to further improve runtime.

Table 1: Performance of base layout and algorithm compared to last years winner

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	time
Sugiyama	14	46*	17	15*	37*	412*	526	911	134	3,327*	3,039*	339,951*	2m1s
100k	**0**	**4**	6	7	**32**	103	13	328	49	1,885	1,736	157,391	4m1s
1m	**0**	**4**	5	4	**32**	**76**	8	310	45	1,475	**1,721**	139,542	22m17s
2m	**0**	7	5	4	**32**	**76**	6	312	52	**1,414**	**1,721**	135,253	45m7s
3m	**0**	**4**	**4**	**3**	**32**	81	**0**	**303**	40	1,506	**1,721**	**134,113**	66m13s
Tübigen-Bus	**0**	**4**	5	4	**32**	81	**0**	307	**38**	1,568	**1,721**	147,628	< 60m

The results from the base drawing, the algorithm with $100,000$, 1, 2 and 3 million iterations, as well as the winning entry *Tübingen-Bus* from last years contest[4], can be seen in Table 1. Results marked with an asterisk do not fulfil all requirements placed upon the drawing. Most runs of the presented algorithm finished under 45 minutes on a *Lenovo Thinkpad T*460, 3 million iterations took about 66 minutes. As the algorithm is based upon randomness, each row represents the best result from 5 different runs. The results seem very promising as there is at least one better result compared to last years for all graphs but one. Looking at the differences between 1, 2 and 3 million iterations is becomes clear that rather than having one single run over the course of one hour, it might be beneficial to have multiple smaller runs with different random seeds and keeping only the best result for each graph. While the possibility or iterations introducing new edge-crossings improves the final results, it is possible to introduce new edge-crossings which cannot be removed later on no matter the amount of iterations.

[2] https://ogdf.uos.de/

[3] https://www.openmp.org/

[4] http://mozart.diei.unipg.it/gdcontest/contest2019/results.html

6 Different Approaches

This section will go through different approaches not included in the final algorithm to possibly further reduce the number of crossings and some of their results.

6.1 Base Drawing

While Sugiyama proved itself as the best algorithm for strictly upwards drawings, other base drawings were tested and should be used depending on the circumstances.
Most prominently, a base drawing to simply space out the nodes as much as possible while maintaining the fulfilment of all restrictions was tested. Expectedly, it did not reduce the number of crossings at all. The increased range of movement for each node combined with the almost instantaneous runtime for the given graphs did not make up for local minima created by Sugiyama.
The other algorithm considered was energy based. While it certainly would yield comparable if not better results than Suigyama, the used libraries did not support energy based upwards drawings. Planning, writing and testing the method myself did not fit into the scope of this project but is advised for future projects.

6.2 Node Selection

Another major part of optimization seemed to be the selection of nodes, which happens completely at random. Table 2 shows the result from different selection methods, each tested with $100,000$ displacement steps on top of the Sugiyama base drawing. $100k$ represents the results for random selection at the time. As the algorithm is based on randomness each row is represents the best result from five different runs. In order of the table the following methods were tested:

1. The basis to the algorithms used was to initially calculate all crossings and select the next node weighted by their crossing number. A fully greedy algorithm would be to prone to local minima. $Cross\ Weight$ simply used the number of crossings as-is and therefore would not select nodes without any crossings. It is included as a baseline but was never expected to outperform the purely random approach. Its major flaw is that recalculating edge-crossings for every node in each step would not be feasible without massively increasing the runtime, so only the current node was updated. This commonly lead to local minima as most nodes would not be selectable despite being involved in crossings introduced by moving other nodes but never being updated themselves.
2. $1w$ and all other entries with this naming scheme introduced a base chance for each node to be selected on top of their crossing number. Already with a base weight of 1 the results for smaller graphs seem much better. As can be seen increasing the weight further improves the results for some graphs and decreases it for others. A weight

of 100 for most graphs is well beyond the number of crossings a single node could be involved with. Yet despite being almost random, the largest graphs still performed worse than before. Increasing the weight further would just imitate the random selection completely but with a large chunk of initial runtime to get the edge-crossings of each node. The results could be improved by updating all nodes in each step, but this would almost multiply the runtime of each step by the number of nodes.

3. As the weighted selection always outperformed random selection on the first few graph, but was underperforming in graphs with large number of nodes a mixed approach was tested. The best result was achieved with a base weight of 10 and a linearly decreasing chance to use the weighted selected from 100% in the first step down to 0% in the last step. However none of the results warranted the extra runtime for a weighted selection since the smaller graphs could be fixed with more iterations and the larger graphs always performed better with a random selection.

Other tested weighted selection methods included decreasing weights over time - both prioritising nodes with more crossings and nodes not selected for the most steps, periodically updating all edge-crossings every x steps as well as both a changing and static chance to for a purely random and greedy step. Compared to the more in-depth explained ones above they performed worse in either runtime or resulting edge-crossings.

6.3 Parallelization

The first thing considered was to change the way the algorithm is parallelized in order to improve performance as the algorithm could be described as randomized brute-forcing within certain boundaries.

- The algorithm could simply work on all input graphs in parallel. However this cannot yield a better performance than being roughly twice as fast since only two graphs of the dataset occupy most of the runtime. However beyond the scope of the contest this would be the best solution for equally large graphs.
- Another approach is to run the displacement steps in parallel. In theory this could scale the same way as running the graphs in parallel would but without the requirement of equally sized graphs. The downside would be the possibility of introducing erroneous steps. Surprisingly only the upward-facing attribute of the edges posed a problem when moving multiple nodes at once. Fixing this afterwards would both ruin the speed-up and introduce new edge-crossings.
- A third way to parallelize the algorithm would be to test x many new positions for the same node in each iteration, but the loss of iterations due to the sequential calculation of edge-crossings was not worth it on the hardware used as there were not enough cores to make significant difference with the number of positions tested. On more powerful hardware this approach should be considered again.

Table 2: Performance of different node selection methods

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
100k	0	11	6	6	32	130	21	576	82	2,826	1,728	212,714
CrossWeight	2	5	5	8	32	120	23	362	46	2,153	2,429	231,582
1w	0	4	5	6	32	133	21	372	44	2,144	2,514	229,974
10w	0	4	5	4	32	134	7	392	39	2,136	2,540	232,182
100w	0	4	5	6	32	126	14	393	37	2,142	2,622	228,550
10w mix	0	4	6	6	32	122	24	419	39	2,124	2,625	230,613

6.4 Stopping Criterion

Looking through the graphs provided by the contest, it may occur that a stopping criterion would be from benefit. Most of the graphs are so small that running the displacement steps a few million times seems counter-intuitive. Such a criterion could e.g. work with the metrics already calculated and stop the iterations when nothing improved for a certain amount of prior iterations. In Figure **??** the runtime of 2 million

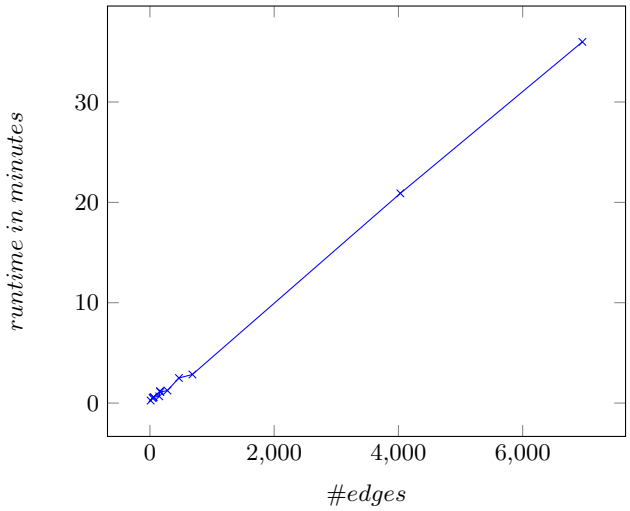

Fig. 1: Runtime in minutes for different amounts of edges, all with 2 million displacement steps

displacement steps broken down to the individual graphs can be found. Somewhat expectedly the runtime is linear on the number of edges. Going back to the actual results provided in Table 1 it becomes clear that most time is spend the last two graphs. While a stopping criterion may improve the runtime for the first few minutes, these large graphs likely will not achieve their minimal crossing number within the given time frame.

Adding any other operation on top of the displacement step would provide no benefit for these graphs and the seconds saved in the beginning would be counterweighted by the minutes lost later on. Given an environment without a time limit a stopping criterion would improve the algorithm as no specific iteration number would be needed.

7 Conclusion

In this work a simple algorithm mostly based on randomness is presented. While its performance seems promising, there are still many things to be experimented with which would simply not fit into the scope of this project. The major take-away should be that random algorithms can be both easy to understand and yield competitive results. Throughout the development it proved time and time again that to much restrictions only seem to worsen the result and the simplest approach is the best one: Leaving the randomness as much room as possible by only directing it in such a way that nothing outside of the confinements of the task can be produced. A final word on the contest itself cannot be given at this time as the contest is still to be held. Future algorithms building upon this work should keep in mind that many decisions made are purely based on the restrictions of the contest and should be changed accordingly as necessary. The choice of base drawing algorithm is especially important to rethink based on new requirements. As overthinking the algorithm itself would most likely not improve the results by much it is suggested that more time is spend with the actual implementation and its efficiency as well as the approach to parallelization. Nothing improved the results as much as drastically increasing the number of random displacements or simply testing multiple different random seeds in the given time frame.

References

[1] Michael A. Bekos et al. "A Heuristic Approach towards Drawings of Graphs with High Crossing Resolution." In: *CoRR* (2018). URL: http://dblp.uni-trier.de/db/journals/corr/corr1808.html#abs-1808-10519.

[2] Béla Bollobás. *Modern Graph Theory*. 1998. DOI: 10.1007/978-1-4612-0619-4.

[3] Markus Chimani and Robert Zeranski. "An Exact Approach to Upward Crossing Minimization." In: *ALENEX*. Ed. by Catherine C. McGeoch and Ulrich Meyer. SIAM, 2014, pp. 73–85. ISBN: 978-1-61197-319-8. URL: http://dblp.uni-trier.de/db/conf/alenex/alenex2014.html#ChimaniZ14.

[4] Almut Demel et al. "A Greedy Heuristic for Crossing-Angle Maximization." In: *Graph Drawing*. Ed. by Therese C. Biedl and Andreas Kerren. Vol. 11282. Lecture Notes in Computer Science. Springer, 2018, pp. 286–299. ISBN: 978-3-030-04414-5. URL: http://dblp.uni-trier.de/db/conf/gd/gd2018.html#DemelDMRW18.

[5] D. S. Garey M.R.;Johnson. *Crossing number is NP-complete.* 1983.

[6] Nikola S. Nikolov. "Sugiyama Algorithm." In: *Encyclopedia of Algorithms.* 2016, pp. 2162–2166. URL: http://dblp.uni-trier.de/db/reference/algo/algo2016.html#Nikolov16a.