Graph Drawing Contest 2020
Crossing Minimization with Randomness

Sebastian Benner

No Institute Given

Abstract. Stuff

1 Introduction

The annual Graph Drawing Contest[1] is an open challenge to design an algorithm for optimized graph drawing. The exact criteria for such a drawing are changed every couple of years, the current ones remain the same as last years challenge. 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.
The main criteria this time around are crossing which ought to be minimal in the resulting drawing. In itself this already poses a NP-hard problem. Additional constraints placed on the drawing are:

- 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.

After all 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 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.
During the last couple of years most if not all of the top scoring contestants based their algorithm at least to some part on randomness which will be the basis of this work. The goal of is to find a balance between lightweight calculations for random steps and a more directed approach to randomness.

2 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

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

therefore E is a subset of $V^{(2)}$. 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 unordered 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.

3 Prior Work

A Heuristic Approach towards Drawings of Graphs with High Crossing Resolution [1] While not aiming at the same goal, this algorithm served as basis for the last years winner. The aim was to create a drawing with 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.

A Greedy Heuristic for Crossing-Angle Maximization [3] Another winning contribution to a different year with the same aim as before, an algorithm to create a drawing with the largest possible minimum angle. Starting from a force directed base drawing the algorithm greedily selects the minimal crossing in the current drawing. A random vertex is chosen and displaced within a square around the original position to achieve the maximal local crossing angle. The size of the square is decreased in each iteration.

4 Sugiyama Framework

5 Crossing Minimization

Following the base drawing, the graph drawing algorithm is heavily based on 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 improvement 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 iterations I. Each iteration only tests a single new position 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. Iff the node is within the bounding box of an edge, it is tested against a crossing.
   (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 crossing afterwards.
2. Iff all metrics equal 0 all other steps are skipped.
3. To assign a new position (x, y) to the node, its x value is randomized on the entire width of the grid. The y value is only randomized between the highest lower neighbour and the lowest upper neighbour. This assures that each edge will remain upwards facing.
4. Afterwards each metric is calculated again.
   (a) Iff 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) Iff 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 does not violate any restrictions.
   (d) Otherwise the displacement is reversed.

The algorithm was implemented in C++ using *Open Graph Drawing Framework*[2]. In order to increase the performance of the edge-crossings algorithm all edges were run in parallel using *OpenMP*[3]. 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.

The results from the base drawing, the algorithm with both $100,000$ and 1.5 million iterations and the winning entry $Tbingen - Bus$ 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. Both runs of the presented algorithm finished within the one hour mark on a

[2] https://ogdf.uos.de/
[3] https://www.openmp.org/
[4] http://mozart.diei.unipg.it/gdcontest/contest2019/results.html

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Sugiyama	14	46*	17	15*	37*	412*	526	911	134	3,327*	3,039*	339,951*
100k	0	4	6	7	32	103	13	328	49	1,885	1,736	157,391
1m	0	4	5	4	32	76	8	310	45	1,475	1,721	139,542
Tübigen-Bus	0	4	5	4	32	81	0	307	38	1,568	1,721	147,628

Fig. 1. Performance of base layout and algorithm compared to last years winner

*Lenovo Thinkpad T*460, 100, 000 is included nonetheless as most changes presented later on were only tested with this smaller number of iterations. On most instances last years winner is either equivalent or better performing, especially on larger graphs. There are only two instances were this is not the case, one within each of the runs. Since the algorithm is mostly based upon randomness the run with less iterations is actually better than the longer run twice for the smaller graphs.

6 Different approaches

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

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
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

Fig. 2. Performance of different node selection methods

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. 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. $CrossWeight$ 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 are much closer to the purely random approach. As can be seen increasing the weight further improves the results slightly. The problem is how well it improves. 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 smaller graphs still perform worse than before. Increasing the weight further would just imitate the random selection 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. Looking closely the weighted selected outperformed always outperformed random selection on the final and largest graph, but was underperforming in graphs with only a few crossings left. This data called for a mixed approach. 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.

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.

As no better selection algorithm could be found the next thing to search for was a better way to select the new node position in each step. Once again several different methods were tested with $100,000$ steps on top of the Sugiyama base drawing.

– What first comes to mind would be limiting the x dimension just like the y dimension is limited. By reducing the maximum distance a node could be moved at once the chance for attempts

7 Conclusion

References

[1] Michael A. Bekos et al. "A Heuristic Approach towards Drawings of Graphs with High Crossing Resolution." In: *CoRR* abs/1808.10519 (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] 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.