TECHNISCHE
UNIVERSITÄT
WIEN

Graph Drawing Algorithms

Upward Drawing Exercise

by Eugen Havasi & Peter Rjabcsenko

The Approach

2 Part Algorithm:

- Custom deterministic approach for initial solution
- Simulated Annealing metaheuristic for optimization

Manual parameter selection for metaheuristic

Solution Data Structure

Additionally to the input data we also made use of the following:

- Each node keeps track of its **predecessors** and **successors** (for convenience)
- Node relations are stored in an **adjacency matrix**
- Early on a **layer list** is created for ease of access of nodes on a particular layer
- Consequently the **maximum layer number** is calculated
- **[m.l.n.** x **width]** sized **occupation matirx**
- **Cost** represents the total number of edge crossings

Initial Solution Generation

A deterministic approach to calculate a feasible initial solution:

- **Feasibility criteria:** upward directed, within grid boundaries, no overlapping edges / nodes positioned on edges
- **Initial idea:** Sugiyama's Framework with layering and dummy nodes failed, no solution for bent edges
- **Approach:** recursively calculate layer of each node, then assign X coordinates to nodes on each layer bottom-up

Initial Solution Generation

- **Grid points:** the cells of the **occupation matrix** keep track of whether a grid point is free, occupied by a node or an edge goes directly through it
- **Feasibility violation:** when assigning a new node to an empty grid point, all edges to its predecessor are tested for violating feasibility (node/ edge overlaps)
- **Initial Cost:** total crossings are calculated after graph is positioned on grid

Simulated Annealing Metaheuristic

- **Escape Local Optima:** S.A. has a higher probability to accept worse solutions at the beginning, facilitating the escape from local optima
- **Neighbourhood structure:** random node assigned to random (feasible) X coordinate on the same layer, solution was always kept feasible
- **Stopping criteria:** either the temperature reaches a minimum value or a given time period elapses
- **Incremental Evaluation:** recompute crossings only in an area affected by the moving of selected node. Unfortunately was not implemented.

Simulated Annealing (Parameters)

- **Temperature:** number of vertices in the graph (arbitrary selection), usually max cost – min cost
- **Cooling Rate:** we used a static cooling rate of 0.95
- **Equilibrium Coefficient:** when multiplied with node number, results in number of moves attempted before the temperature is adjusted
- **Stopping Criterion:** additionally to the time limit (3min), if the temperature dropped below 0.01 we also terminated the search (no meaningful improvements reached after this threshold)

“Test” Results (over 10 runs)

#	Init. cost	Best cost	Mean cost	Std. dev.	CPU time (sec)
01	0	0	0	0	0.0015
02	17	12	12	0	0.0187
03	16	2	2	0	0.0178
04	73	27	27.3	0.9	0.064

“Auto” Results (over 10 runs)

#	Init. cost	Best cost	Mean cost	Std. dev.	CPU time (sec)
01	21	14	14	0	0.0062
02	111	4	7.6	3.382	0.1281
03	270	6	7.7	3.796	0.5875
04	172	11	14.1	1.758	0.7125
05	733	32	32	0	10.30
06	2279	332	358.2	10.21	34.40
07	5734	260	315.2	60.64	38.10
09	6362	6223	6246.3	17.74	173.3
10	18182	18158	18174	11.31	172
12	1269418	1268500	1268925	364.9	177

Conclusions / Remarks

- We observed that we can improve (even if sometimes only marginally) the solution quality in a really short time

  Combination of deterministic and stochastic methods seem to work well

- **Room for improvement:** We failed to implement **incremental evaluation** and could have definitely benefited from a longer time limit for S.A.

- We were not able to compute even an initial solution for instances “auto8” and “auto11”

Instructions

The algorithm can be run as a .jar from the command line, e.g.:

```
java -jar upwarddrawing.jar auto1 true 180 0.95 0.01 15
```

where:

auto1: instance name without .json prefix

true: whether execution is verbose (optional, default: true)

180: time limit for S.A. in seconds (optional, default: 180)

0.95: cooling rate (optional, default: 0.95)

0.01: termination condition (optional, default is 0.01)

15: equilibrium coefficient (optional, default is square root of node number)

Instructions contd.

- Instances should be placed in an “instances” folder on the same level as the .jar file.
- Additionally a “solutions” folder needs to be created and placed on the same level as the .jar file
- The source itself is a Java project using Maven

TECHNISCHE UNIVERSITÄT WIEN

THANK YOU

FOR YOUR ATTENTION!