Book Embeddings:
K-Page Crossing Number Minimization Problem

Heuristic Optimization Techniques, 2015WS

October 21, 2015

1 Problem description

The *Book Embedding Problem (BEP)* is a problem from the field of graph theory. It represents a generalization of the planar embedding problem i.e. "Is a given graph $G = (V, E)$ with vertex set V and edge set E planar?". Short recap: A graph is planar iff it can be drawn in a plane s.t. no edges cross each other. It is well known that planarity testing for graphs can be solved in $\mathcal{O}(|V|)$.

The BEP generalizes this problem by asking "how many pages of a book do you need to embed (draw) the edges of a graph without crossings if all vertices are arranged on its spine?". The resulting number of pages is called the *book thickness*, *page number*, or *stack number*. Calculating the page number of a graph is known to be $\mathcal{NP}$-complete. Be aware that even a simple planar graph might have page number larger than 1. One simple example is the K_4 graph. For planar graphs, it is known that graphs with page number up to 3 exist, though it is an open problem if planar graphs with page number ≥ 4 exist. Applications for the BEP exist e.g. in VLSI design and graph drawing.

For the programming exercises we consider a variant of the BEP which is also $\mathcal{NP}$-complete—the *Book Embedding Problem with minimal number of crossings*, also called *K-page crossing number minimization problem (KPMP)*. It is defined as follows:

Definition 1 *Given an undirected graph $G = (V, E)$ and page number of K, find a book embedding of G with K pages s.t. the number of crossings is minimized. Formally a valid K-page book embedding is described by*

- *a spine ordering $v_{i_1} < v_{i_2} < \ldots < v_{i_{|V|}}$ of V determining the sequence of the vertices on the spine, and*
- *a edge partitioning $P_E = \{S_1, \ldots, S_K\}$ of E.*

A crossing is defined as

$$\{i_1, i_2\} \in S_k, \{j_1, j_2\} \in S_k, v_{i_1} < v_{i_2}, v_{j_1} < v_{j_2} : \; v_{i_1} < v_{j_1} < v_{i_2} < v_{j_2}$$

2 Programming framework

To help you with the implementation of your programming exercises we provide you with a small programming framework for C++ and Java. The first part of

the framework is an instance reader which reads BEP problem instances in the following format. The input format is also the expected solution format.

```
#Comment line a lines are skipped during parsing
5 #Number of vertices |V|; comments are ignored
3 #K - number of pages
#Next |V| lines will name the vertex ids (integers)
#in a random spine order
#Counting always starts at zero
0
1
2
3
4
#The remaining lines list the edges of the graph:
#VERTEX_ID_A VERTEX_ID_B [PAGE]
#The initial page assignments are can be ignored in input files.
0 1 [0]
0 3 [1]
2 4 [0]
1 2 [0]
...
```

We provide with a small programming framework for C++ and Java 1.8. The framework implements instance reading and solution writing for the KPMP. Use:

- `KPMPInstance` and the method `readInstance` to read an instance specification from a file You can access the graph information via the provided methods. The instance graph is available both as adjacency lists and adjacency matrix.
- `KPMPSolutionWriter` to write a solution either to a file (via `write`) or `stdout` (via `print`).
  - `KPMPSolutionWriter::setSpineOrder` expects a list of vertices which define the spine order.
  - `KPMPSolutionWriter::addEdgeOnPage` expects two vertices forming an edge and a page number where the edge resides.

Be aware that those classes are not intended to perform any validation of your solutions. Further `KPMPInstance` is not intended as solution representation. You should design your actual representations based on your implemented algorithms.

3 Reports

For each programming exercise we require you to hand in a short report (∼2 pages) via TUWEL containing (if not otherwise specified) at least:

- A description of the implemented algorithms

- Experimental setup (Machine, tested algorithm configurations)
- Best objective values and runtimes (+ Mean/Std. dev. for randomized algorithms over ∼30 runs) for all published instances.
- Do not use excessive runtimes for your algorithms, limit the maximum cpu time (not wall clock time!) to ∼15 minutes per instance.

4 AC cluster

To provide a standardized experiment environment it is possible to use the AC computing cluster. Login using ssh on `USERNAME@eowyn.ac.tuwien.ac.at` or `USERNAME@behemoth.ac.tuwien.ac.at`. Both machines run `Ubuntu 14.04 LTS` and provide you with a `gcc 4.8` toolchain and `Java 1.8`. External libraries required by your program should be installed in your home directory and the environment variables should be set accordingly.

Before submitting a command to the computing cluster create an executable e.g. a bash script setting up your environment and invoking your program. It is possible to supply additional command line arguments to your program. To submit a command to the cluster use:

```
qsub -l h_rt=00:15:00 [QSUB_ARGS] COMMAND [CMD_ARGS]
```

The `qsub` commmand is a command for the Sun Grid Engine and the command above will submit your script with a maximum runtime of 15 minutes (hard) to the correct cluster nodes. Information about your running/pending jobs can be queried via `qstat`. Sometimes you might want to delete (possible) wrongly submitted jobs. This can be easily done by typing `qdel <job_id>`. You can find additional information under:
`https://www.ac.tuwien.ac.at/students/grid-engine/`

Once you found a new good solution you can submit a solution file use the following command:

```
hot2015submit ASSIGNMENT INSTANCE_NAME SOLUTION_FILE
```

`ASSIGNMENT` needs to be of the form "assignmentX" where X is the number of the assignment. `INSTANCE_NAME` is the name of the instance for submitted solution e.g. "automatic-1". `SOLUTION_FILE` is the path to your solution file in same format as the input files. If you encounter a problem with the submission please send a mail to `heuopt-ws15@ac.tuwien.ac.at`.

The submission system will check your solution for correctness and update the table of the best solutions. The best solutions can be queried by executing

```
hot2015results
```

which will print the currently best known solutions to the command line (make sure that your window is large enough).