Reference manual

Kemény Tamás

Jan-Jul 2018

1 Introduction

This repository contains an implementation of a Parameterized Approximation Scheme (PAS) for the Steiner tree problem for weighted undirected graphs.

The Steiner tree problem:
Given:

- a graph $G = (V, E)$
- positive edge weights $w : E \to \mathbb{R}^+$
- a set $R \subseteq V$ of *terminals* ($V \backslash R$ are called *Steiner vertices*)

Find: *Steiner tree* $T \subseteq G$ connecting all terminals of R with minimum weight

2 Classes

Edge An instance of the edge class represents an edge of the graph. Each edge contains integers representing its two endpoints and its weight. Each edge of the input graph is assigned one unique id, however this id is stored as a singleton list, since after the metric completion of the graph, some edges will represent a path in the input graph rather than a single edge. In this case the id of the edge corresponds to the ids of each edge along the path.

Vertex A vertex has information about itself and its neighbouring vertices. Each vertex contains a boolean isTerminal, a list of neighbouring vertices sorted increasingly by the weight of the incident edge, a set of neighbouring terminals and a list of sums of weights of edges to neighbouring terminals up to an index. The latter list is precomputed once reading of the input graph is finished and recomputed when the list of neighbouring terminals changes. This way during calls to the bestStar method, the appropriate sum maybe be fetched and divided by the number of terminals in the star minus one for a fast computation of the minimum ratio.

Graph The Graph class stores all data necessary to represent an instance of the Steiner problem i.e.: a set of vertices, edges, terminals and a weight function. Furthermore the class implements methods to modify the graph, such as adding and removing edges or modifying Steiner vertices into Terminals.

Kernel The Kernel class contains methods implementing the graph operations the reduction method relies on, such as bestStar, contractStar and computeClosure:

1. Pair< Double, List <Edge> > **bestStar**(Integer center)
   The bestStar method computes the star with the best possible ratio centered at a given vertex and returns a Pair whose first element is a double representing the minimum ratio and second element is a list of the corresponding edges of the star.

2. void **contractStar**(List<Edge> STAREDGES, Integer CENTER)
   The contractStar method is equivalent to successively performing edge contractions for every edge in STAREDGES, however to prevent loops and to make things faster the routine does the following:
   (a) introduce a new vertex, mark it as terminal
   (b) for each vertex in STAREDGES gather its neighbourhood (excluding vertices in STAREDGES) in a set
   (c) add a new edge between the new vertex and each vertex in the set
   (d) remove STAREDGES from the graph.
3. void **computeClosure**(Integer UPPERBOUND)
   The computeClosure method finds the metric closure of the graph up to some upperbound L, that is for every pair of non-adjacent vertices a connecting edge is added to the graph with weight equal to the minimum distance between the vertices. This is done by running Dijkstra's algorithm starting from each vertex of the graph. During a Dijkstra procedure if the top of the heap exceeds the upper bound, the computation is stopped and the state of the routine is saved (min-heap, list of visited nodes) so that we may continue once the upper bound is incremented.

The Kernel class however does not specify what order star contractions and the metric completion should take place, instead it serves as a helper class for **FastKernel** and **SlowKernel**.

FastKernel The FastKernel class inherits from the Kernel class, performs star contractions until no stars are available for contraction, then computes the metric completion of the graph up to a small threshold L. The threshold L is increased incrementally until there is a star we can contract, which is then contracted. This technique significantly reduces the time spent on computing the metric completion of the graph.

SlowKernel The SlowKernel class also inherits from the Kernel class, instead of the the incremental completion of the graph however, this method computes the whole completion of the graph and then contracts a single star.

DreyfusWagner This class provides an implementation of the classical Dreyfus, Wagner '71 algorithm. The algorithm solves the problem exactly using dynamic programming, the idea is to compute a tree spanning a larger and larger subset of the set of terminals, until we have the optimum Steiner tree. The table can be computed by the following recurrence relation:
For every $D \subseteq R$ of size at least 2, and every $v \in V(G) \backslash R$

$$T[D,v] = \min_{\substack{u \in V(G) \backslash R \\ \emptyset \neq D' \subsetneq D}} \{T[D',u] + T[D \backslash D', u] + dist(v,u)\}$$

Observe that entries $T[D,v]$ where $D = \{u\}$ is a singleton set can be filled in trivially by $dist(v,u)$. The weight of the optimum Steiner tree can be read from the bottom row $T[K,v]$ and the Steiner tree can be reconstructed by backtracking from the optimum entry in the table through a tree of parent pointers. Note that this algorithm has exponential time and space complexity.

ReadInput The ReadInput class reads a given file containing the description of an instance of the steiner problem and returns an instance of the Graph class. The specified file must be in the .gr format.

For documentation on more classes, see the Javadocs (via doc/index.html).

The description of a file in the .gr format is the following:

The file starts with a line ‘SECTION Graph’. The next two lines are of the form ‘Nodes #nodes’ and ‘Edges #edges’, always in that order, where #nodes is the number of vertices and #edges is the number of edges of the graph. The #edges next lines are of the form ‘E u v w’ where u and v are integers between 1 and #nodes representing an edge between u and v of weight the positive integer w. The following line reads ‘END’ and finishes the list of edges.

There is then a section Terminals announced by two lines ‘SECTION Terminals’ and ‘Terminals #terminals’ where #terminals is the number of terminals. The next #terminals lines are of the form ‘T u’ where u is an integer from 1 to #nodes, which means that u is a terminal. Again, the section ends with the line ‘END’.

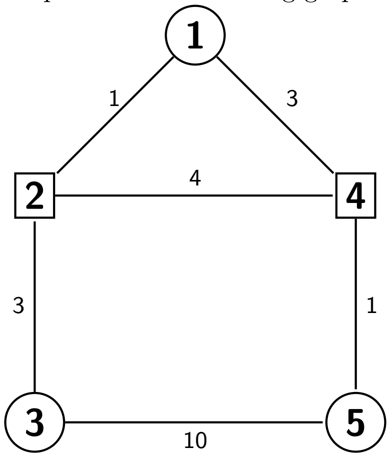
Example: instance000.gr

```
SECTION Graph
 Nodes 5
 Edges 6
E 1 2 1
E 1 4 3
E 3 2 3
E 2 4 4
E 3 5 10
E 4 5 1
END

SECTION Terminals
 Terminals 2
T 2
T 4

END
```

Represents the following graph:



3 Usage

Run the executable jar with the following java command:
`java -jar steiner-tree-pas.jar -in <input filename>.gr`

List of parameters:

`-ui [input|output]` - display input or output file in interactive ui

`-silent` - mute standard output message

`-long` - more detailed standard output

`-s p` - run FPT algorithm on at least `p` Steiner vertices

`-t r` - run FPT algorithm on at least `r` Terminals

`-a [0..10]` - instead of using options `-s` or `-t` we may specify a number between 0 and 10 as a rough indication of the accuracy with which we want to run the approximation scheme (0 being the least and 10 the most accurate)

`-c` - continuously find better approximations, until the optimum is found

`-out` - write computed Steiner tree into **`<input filename>-tree.gr`**

`-slow` - use slower, but possibly more accurate reduction method

Examples:

- **`java -jar steiner-tree-pas.jar -in instances\instance001.gr -ui input`**
  Display instance001.gr in the ui
- **`java -jar steiner-tree-pas.jar -in instances\instance001.gr -ui output -a 10`**
  Compute optimum and display Steiner tree in the ui
- **`java -jar steiner-tree-pas.jar -in instances\instance019.gr -c -long -out`**
  Compute exhaustively the best feasible solution for instance019.gr with detailed standard output and keep writing the Steiner tree into **`instance019-tree.gr`**

4 Dependencies

For development the following dependencies are needed:

JDK 8+	- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Maven	- https://maven.apache.org/install.html