11th DIMACS Implementation Challenge

in Collaboration with ICERM:

Steiner Tree Problems

Competition Rules

The DIMACS Implementation Challenges address questions of determining realistic algorithm performance where worst case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails. Experimentation also brings algorithmic questions closer to the original problems that motivated theoretical work. It also tests many assumptions about implementation methods and data structures. It provides an opportunity to develop and test problem instances, instance generators, and other methods of testing and comparing performance of algorithms. And it is a step in technology transfer by providing leading edge implementations of algorithms for others to adapt.

The 11th Implementation Challenge is dedicated to the study of Steiner Tree problems (broadly defined), bringing together research in both theory and practice. This edition of the challenge is co-organized by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) and ICERM (Institute for Computational and Experimental Research in Mathematics). The challenge is part of the DIMACS Special Focus on Information Sharing and Dynamic Data Analysis and will be capped by a workshop hosted by ICERM at Brown University in Providence, Rhode Island, in December 2014.

1 Challenge description

The classical Steiner Tree problem has the goal of connecting a given set of objects with the smallest possible cost. There are many variants on this classical setting that involve the introduction of additional complexities to the problem. This challenge aims to develop state-of-the-art implementations to solve the classical Steiner tree problem and its many variants.

The variants considered for this challenge include: classical Steiner problem in graphs, classical geometric Steiner problem (including rectilinear, Euclidean, and higher dimensions), prize-collecting

Steiner trees, dynamic Steiner trees, node-weighted Steiner trees, generalized Steiner trees (Steiner forest), obstacle-avoiding Steiner trees, directed Steiner trees (arborescences), rooted Steiner trees with height (or hop) constraints, group Steiner trees, stochastic variants. While it is not necessary for submissions to this challenge to address every variant, such submissions are welcome.

2 Competition

Given the variety and diversity of the challenge instances, an individual competition for each Steiner tree variant is held. Specifically, a competition will be formed for each variant that has at least two submissions to the challenge. Since it is possible for an individual submission to address multiple variants, the submitter must state which competitions they wish to enter.

For each of the competitions, two different challenges exist. These are the *Pareto Challenge* and *Quality Challenge*. The *Pareto Challenge* assesses each submission in regards to the running time and solution quality. The implementations will be evaluated for the Pareto Challenge using a shifted geometric mean of the primal integral[1] across the instance set. The *Quality Challenge* will assess each of the submissions in regards to the solution quality, regardless of the running time. The best implementation for the Quality Challenge will be decided by the shifted geometric mean of the objective values from the instance set achieved within the given time limit. The total runtimes for the challenge will be set after the first round and will be instance dependent. As a guide the maximum runtimes will be at least 600 seconds and 3600 seconds for the Pareto and Quality challenges respectively.

2.1 Submission timeline

There will be three different submission dates for this challenge. After each submission, an evaluation of the results will be performed and feedback provided to the participants. The submission dates are:

Round 1: 10th November 2014

Round 2: 17th November 2014

Round 3: 24th November 2014

The first two rounds of this competition are designed as testing phases. After these rounds feedback will be provided in regards to the current performance compared to all other submissions and whether there are any other issues. The third round is the actual competition. The winners of each variant competition and related challenges will be determined solely on the solver performance in this round.

Each of the rounds requires different submissions from the competitors. The required submissions for each of the rounds are:

---

[1]Measuring the impact of primal heuristics (T. Berthold), *Operations Research Letters*, 41(6):611–614, 2013

Round 1: 10th November

The competitors will perform experiments on the provided instances at their home institution. No time limit imposed for this round of the competition. The output from the experiments and the output from the *test script* must be submitted. The required format of the output file is detailed in Section 3. Since a competition will be held for each Steiner tree problem variant, the submissions in this round will identify the competing participants.

The output submitted by the participants will be reviewed by the organisers and the competition rankings will be published on the website.

Round 2: 17th November

The experiments will once again be performed at the competitors home institution. In addition to the experiment and test script output, the code or a compiled binary must also be submitted in this round. The submitted code or binary will be tested on the competition machines provided by ZIB. The results submitted by the participants in this round will be compared against the results provided from the test runs performed on the competition machines. Both sets of results will be published to provide feedback and assurance that everything is set up correctly and running well on the competition machines. Again, the competition rankings at the end of this round will be published.

The details regarding the code submission are give in Section 4.

Round 3: 24th November

The final round will involve the evaluation of the submitted code or compiled binary solely on the competition machines provided by ZIB. In this round the competitors must submit the final version of their implementation developed for the challenge. The code or binary, in the form detailed in Section 4, is required for submission in this round. All experiments will be performed on the competition machines.

In this round, the submitted implementations for each problem variant will be evaluated against a set of instances selected by the organisers.

The winners of the Pareto and Quality challenges for each of the problem variants will be decided based upon the results from this round.

3   Output File

The result output file format consists of four sections. The first one provides general information about the problem and the program used to solve the problem. The second section shows the development of the best solution over the solving time. The third section lists information about the optimization run, e.g., the total solving time and final primal and dual bounds (if applicable). These sections must be of the following form:

```
SECTION Comment
Name "Instance Name"   # instance name like stated in the .stp file
Problem "Problem Type" # abbreviation of problem type
Program "Progam Name"
Version "Program Version"
End


SECTION Solutions
Solution 1.4  89.36    # list of all improving primal solution found; each one with
Solution 10.2 81.71    # the time after which it was found and its objective value
...
Solution 3492 77.23    # final solution
End


SECTION Run
Threads 1              # If this is omitted, 1 will be assumed.
Time 3536              # The total running time in seconds.
Dual 45.34             # The computed dual bound. Can be omitted if heuristic.
Primal 77.23           # The objective value of the best found solution.
End
```

Note that the time always refers to the solving time up to that point, including presolving time, but without reading time. The possible problem types are listed in Appendix A.

The fourth section prints the final solution computed by the solver. The format of this section differs, depending on the type of instance. Two main types of problem instances are considered in this challenge, namely Steiner tree on graphs and geometric Steiner tree, which need different data to identify a solution.

For the Steiner tree problems on graphs, the final solution information must look like this:

```
SECTION Finalsolution
Vertices 6            # The number of vertices in the best found solution.
V 1                   # The vertex numbers within the best found solution.
V 2
V 4
V 7
V 9
V 11
Edges 5              # Number of edges in the best found solution.
```

```
E 1 2                   # Directed edges between two vertices. For undirected
E 2 4                   # variants the vertex order is not important.
E 4 7
E 2 9
E 7 11
End
```

The final solution output for the geometric Steiner tree must be provided in the following format:

```
SECTION Finalsolution
Points 5                # Number of points in the best found solution.
Edges 3                 # Number of edges in the best found solution.
PPP x1 y1 z1            # The coordinates of the points.
PPP x2 y2 z2            # NOTE: PPP for 3 dimensions, PPPP for 4 dimensions, ...
PPP x3 y3 z3
PPP x4 y4 z4
PPP x5 y5 z5
E 1 2                   # The edges connecting the previously listed points
E 2 4                   # Directed edges between two vertices. For undirected
E 4 5                   # variants the vertex order is not important.
End
```

4   Code submission

The participants are asked to submit the their code and compiled binaries as either a `.tgz` or `.zip` file. The submitted archives should contain:

- The source code. This is not necessary, but strongly preferred.

- A short README file with a few (short) instructions to aid the evaluation.

  These instructions should include details about required libraries and how to select external software packages, such as CPLEX or Gurobi.

- Any "exotic" extra libraries in source that are necessary for the submitted code. Also, it is important to submit libraries where the version is important.

- A binary of the participants implementation.

- A binary of the challenge test program.

  Download: http://dimacs11.cs.princeton.edu/benchmark/dimbench.cpp

If the source code is submitted, upon unpacking the archive it should be possible to call

`make`	to build a new binary, and
`make check`	to execute the code with a few of the test instances so the evaluators can easily verify that the build was successful.

Additionally, calling `make check`, without first calling `make`, should perform the verification test on the provided binary.

4.1 Submitted binary

The submitted binary should be able to take the following arguments (in order):

```
filename time threads outputfile
```

`filename`	is the instance to be solved.
`time`	is the maximum runtime in seconds. Once this time is reached, the execution of the program will be terminated.
`threads`	is the number of cores that the program may use. This argument should be 1 for single threaded programs.
`outputfile`	the name of the file to write the solution output.

Alternatively, a wrapper script can be provided taking these arguments and calling the binary in an appropriate way.

5 Competition machine details

The competition machines will be provided by ZIB and these will be used to evaluate the submitted code in Rounds 2 and 3. These machines are running the Ubuntu operating system and the gcc and Intel compilers are installed. Additionally, the provided software includes CPLEX, Gurobi, XPress and Mosek. Finally, the libraries lemon, boost and MPI are available. If there is anything in addition to the listed machine features that you require, please contact the organisers.

The evaluation of the submitted implementations will be performed on a cluster located at ZIB. This will provided shared memory for parallel codes up to a maximum of 10 cores. If requested, tests for parallel codes can be performed using more cores. For distributed memory parallel implementations a maximum of a few hundred cores will be used. For single threaded codes, tests will be performed on a single dedicated node of the cluster. The single threaded code tests will be performed multiple times to check for stability and ensure a more accurate evaluation.

A   Problem types

SPG	classical Steiner problem in graphs
CRTP	capacitated ring tree problem
DCST	degree-constrained Steiner trees in graphs
HCDST	hop-constrained directed Steiner tree problem in graphs
ESMT	geometric Steiner minimum trees, euclidian case
RSMT	geometric Steiner minimum trees, rectilinear case
MWCS	maximum-weight connected subgraph
MVDST	minimum vertex-disjoint Steiner trees
NWSPG	node-weighted Steiner problem in graphs
OARSMT	obstacle-avoiding rectilinear Steiner minimum trees
PCSPG	prize-collecting Steiner problem in graphs
RPCST	rooted prize-collecting Steiner tree problem in graphs
STPRBH	Steiner tree problem with revenues, budget, and hop constraints
SSTP	stochastic Steiner tree problem

Table 1: Problem types and abbreviations.