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.

There will be four tracks for each variant:

- The Pareto Challenge (PC) assesses each submission in regards to the running time and solution quality. For each test instance, an algorithm will be evaluated according to the primal integral[1] for the entire execution. As reference value for the primal integral, we may take the optimal solution (if known) or the best solution obtained by any of the solvers during the competition.
- The Quality Challenge (QC) will assess each of the submissions in regards to the solution quality, regardless of the actual running time of the solver. However, a time limit is provided, which will be set after the first round and will be instance-dependent. As a guide, the maximum runtimes will be at least 3600 seconds.
- The Gap-based Exact Challenge (GXC) considers both *lower* and *upper* bounds produced within a given time limit. For each instance, an algorithm is evaluated according to the primal integral of its *gap* (difference between the upper bound and the lower bound).
- The Time-based Exact Challenge (TXC) takes into account the time until the optimality is proven; algorithms that fail to prove optimality are given the time limit.

For each track, we have two subtracks, which differ on how results of individual instances are aggregated into a single score.

- *Points-based method* (P): For each instance tested, points are awarded according to the scoring system used by Formula 1 between 1991 and 2002. For each instance, all competing methods are ranked according to their individual value. The best method gets 10 points, the second 6, then 4, 3, 2, 1. Methods that do not produce results or are not among the top 6 do not receive any points. In case of ties, the points at play are evenly split among the methods involved. For example, if nine methods in the quality challenge find solutions $101, 102, 102, 102, 103, 103, 103, 104, 105$, the

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

scores are 10, 13/3, 13/3, 13/3, 1, 1, 1, 0, 0. Note that there is a triple tie for second, so each method gets a third of the points reserved for positions 2 to 4 ($6 + 4 + 3 = 13$). There is also a triple tie for fifth, so each method involved gets a third of the points reserved for positions 5 to 7 ($2 + 1 + 0 = 3$). The total score of a method is then the sum of its points over all test instances. The method with the highest total score is the official winner.

- *Mean value* (M): Results for different instances are aggregated by computing the geometric mean of individual values.

2.1 Competition timeline

The competition is split into three phases. At the end of each phase, an evaluation of the results will be performed and feedback provided to the participants. The phases are:

Round 1: 10th to 16th November 2014

Round 2: 17th to 23rd November 2014

Round 3: 24th to 30th 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 the competitions for the different variants and related challenges will be determined solely on the solver performance in this round.

Each of the rounds requires different submissions from the competitors (see the following points). Submissions should be sent to `gamrath@zib.de`.

Round 1: 10th November

At the beginning of the round, the competitors need to announce the variants for which they want to join the competition. Since a competition will be held for each Steiner tree problem variant, this will identify the competing participants. Additionally, the participants will perform experiments on the provided instances at their home institution. No particular time limit is imposed for this round of the competition. A *test script* will be provided to run and evaluate the experiments. The interface requirements are described in Section 4, the required format of the output file in Section 3.

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

Round 2: 17th November

The experiments will once again be performed at the competitors home institution. Additionally, the code or a compiled binary must be submitted at the beginning of the phase, so that it can 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.

The details regarding the code submission are give in Section 5.

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. On November 24, the competitors must submit the final version of their implementation developed for the challenge. The code or binary, in the form detailed in Section 5, 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 and announced during the workshop.

3 Output File

The output file format consists of four sections. The first provides general information about the problem and the program used to solve the problem. The second section shows the development of the best primal solution over the solving time. The third section lists information about the optimisation run. This section includes the number of threads used for the computations, the total solving time and the final primal and dual bounds (if applicable). The final section outputs the graph of the best found solution.

The first three 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 improving primal solutions;
Solution 10.2 81.71   # the first entry is the time when the solution was found,
Solution 32.7 79.21   # the second entry is the objective value of the solution.
...
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
```

It must be noted that the time is measured as wallclock time and includes all time spent by the program, in particular also reading and presolving time. The "Problem Type" in the `Comment` section must be an abbreviated name of the Steiner tree problem variant that is being solved. The possible problem types are listed in Appendix A.

The format of the final section, the output of the graph for the best found solution, has a different structure depending on the instance type. There are two main instance types considered in this challenge, namely the Steiner tree problem on graphs and the geometric Steiner tree problem. Each of these require different data to describe the final solution.

For the Steiner tree problems on graphs, the final solution information must be of the form:

```
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 geometric Steiner tree problems 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.
E 5 3
End
```

4 Binary interface requirements

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 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 much 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.

6 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 provide 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.