Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

DANIEL REHFELDT, THORSTEN KOCH

SCIP-Jack—a solver for STP and variants with parallelization extensions: An update

ZIB Report 18-05 (January 2018)

Zuse Institute Berlin
Takustr. 7
D-14195 Berlin

Telefon: +49 30-84185-0
Telefax: +49 30-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

SCIP-Jack – A solver for STP and variants with parallelization extensions: An update*

Daniel Rehfeldt[†‡], Thorsten Koch

Abstract

The Steiner tree problem in graphs is a classical problem that commonly arises in practical applications as one of many variants. Although the different Steiner tree problem variants are usually strongly related, solution approaches employed so far have been prevalently problem-specific. Against this backdrop, the solver SCIP-Jack was created as a general-purpose framework that can be used to solve the classical Steiner tree problem and 11 of its variants. This versatility is achieved by transforming various problem variants into a general form and solving them by using a state-of-the-art MIP-framework. Furthermore, SCIP-Jack includes various newly developed algorithmic components such as preprocessing routines and heuristics. The result is a high-performance solver that can be employed in massively parallel environments and is capable of solving previously unsolved instances. After the introduction of SCIP-Jack at the 2014 DIMACS Challenge on Steiner problems, the overall performance of the solver has considerably improved. This article provides an overview on the current state.

1 Introduction

The Steiner tree problem in graphs (STP) is a classical $\mathcal{NP}$-hard problem [1] entailing a wealth of research articles. Given an undirected, connected graph $G = (V, E)$, costs $c : E \to \mathbb{Q}_{\geq 0}$ and a set $T \subseteq V$ of *terminals*, the problem is to find a tree $S \subseteq G$ of minimum cost that includes T.

While Steiner tree problems can be found in various applications, these problems are usually one of the many variants of the STP, such as for instance the rectilinear Steiner tree problem [2] or the prize-collecting Steiner tree problem [3]. The 2014 DIMACS Challenge, dedicated to Steiner tree problems, marked a revival of research on the STP and related problems: Both at and in the wake of the Challenge several new Steiner problem solvers were introduced and many articles were published. One of these new solver is *SCIP-Jack*, which was by far the most versatile solver participating in the DIMACS Challenge, being able to solve the STP and 10 of its variants (note that in the current version one more variant can be handled). Moreover, SCIP-Jack was able to win two categories of the Challenge.

SCIP-Jack is described in detail in the article [4], but already in an updated version that vastly outperforms its predecessor participating in the DIMACS Challenge. However, the development of SCIP-Jack did not stop with [4]. In the following we will report on recent improvements and provide current results that again demonstrate a significant speed-up of SCIP-Jack.

*The work done for this article was supported by the BMBF Research Campus Modal SynLab and by the BEAM-ME project.

[†]Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, {rehfeldt, koch}@zib.de

[‡]Technical University Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany

Table 1: SCIP-Jack can solve the STP and 11 related problems

Abbreviation	Problem Name
STP	*S*teiner tree *p*roblem in *g*raphs
SAP	*S*teiner *a*rborescence *p*roblem
RSMT	*R*ectilinear *S*teiner *m*inimum *t*ree problem
OARSMT	*O*bstacle-*a*voiding *r*ectilinear *S*teiner *m*inimum *t*ree problem
NWSTP	Node-weighted *S*teiner *t*ree problem
PCSTP	*P*rize-*c*ollecting *S*teiner *t*ree *p*roblem
RPCSTP	*R*ooted *p*rize-*c*ollecting *S*teiner *t*ree *p*roblem
MWCSP	*M*aximum-*w*eight *c*onnected *s*ubgraph *p*roblem
RMWCSP	*R*ooted *m*aximum-*w*eight *c*onnected *s*ubgraph *p*roblem
DCSTP	*D*egree-*c*onstrained *S*teiner *t*ree *p*roblem
GSTP	*G*roup *S*teiner *t*ree *p*roblem
HCDSTP	*H*op-*c*onstrained *d*irected *S*teiner *t*ree *p*roblem

Internally, all problems are transformed into the Steiner arborescense problem (SAP), the directed version of the STP [4]. In only two cases it is necessary to add specific constraints. The transformations are a distinct feature of SCIP-Jack and allow for a generic solving approach with a single branch-and-cut algorithm. Descriptions on some of the transformations can be found in [5]. While in principle it would be possible to only employ solving routines for the SAP (since each problem variant can be transformed to it), this approach falls far short of being competitive as it fails to utilize special properties of particular problem types. Therefore, SCIP-Jack includes a plethora of specialized heuristics and preprocessing routines. Table 2 indicates for which problem variants specialized algorithms are used. Detail on the heuristics is given in [4] and [5] (and in Section 2), while detail on the preprocessing can be found in [6].

2 Recent improvements

Apart from general improvements, particular progress has been made with the MWCSP and the PCSTP. Therefore, these two variants will be discussed separately in Section 2.2.

2.1 General improvements

The general improvements of SCIP-Jack include a change in the default propagator, see [4], which now additionally employs reduction techniques to fix variables (of the underlying IP formulation) to zero. These variables correspond to arcs in the SAP—to which all Steiner tree variants including the STP are transformed. Whenever ten percent of all arcs have been newly fixed during the branch-and-cut procedure, the underlying, directed, SAP graph D is (re-) transformed into a graph G for the respective Steiner tree problem variant. All edges (or arcs) in G that correspond to arcs that have been fixed to 0 in D are removed. Thereupon, the default reduction techniques of SCIP-Jack are used to further reduce G and the changes are retranslated into arc

Table 2: Transformations, heuristics, and preprocessing according to problem type

Problem	Special Constraints	Virtual Vertices	Virtual Arcs	Special Preprocessing	Special Heuristics
STP	–	–	✓	✓	✓
SAP	–	–	–	✓	✓
RSMT	–	✓	✓	–	–
OARSMT	–	✓	✓	–	–
NWSTP	–	–	✓	–	–
PCSTP	–	✓	✓	✓	✓
RPCSTP	–	✓	✓	✓	✓
MWCSP	–	✓	✓	✓	✓
RMWCSP	–	✓	✓	–	✓
DCSTP	✓	–	✓	–	✓
GSTP	–	✓	✓	–	–
HCDSTP	✓	–	–	✓	✓

fixings in D.

A further important development is the reimplementation of the separation algorithm of SCIP-Jack, which is based on the warm-start preflow-push algorithm described in [7]. The new separation algorithm is for many instances more than ten times faster than the old one. The cause of this speed-up lies both with an improved, cache-optimized implementation and the use of new heuristics. Notably, the underlying maximum-flow routine also vastly outperforms the algorithm described in [8], which is commonly used as a benchmark for maximum-flow algorithms.

2.2 Improvements for MWCSP and PCSTP

Maximum-weight connected subgraph problem. Given an undirected graph $G = (V, E)$ and node weights $p : V \to \mathbb{Q}$, the objective is to find a connected subgraph $S = (V_S, E_S) \subseteq G$ such that $\sum_{v \in V_S} p_v$ is maximized.

Prize-collecting Steiner tree problem. Given an undirected graph $G = (V, E)$, edge-weights $c : E \to \mathbb{Q}_+$, and node-weights $p : V \to \mathbb{Q}_{\geq 0}$, a tree $S = (V_S, E_S) \subseteq G$ is required that minimizes

$$C(S) := \sum_{e \in E_S} c_e + \sum_{v \in V \setminus V_S} p_v. \tag{1}$$

The most important component for accelerating exact solving of both PCSTP and MWCSP is graph reduction—which can for instance be employed in preprocessing. A simple reduction routine for the MWCSP is for example to contract all adjacent vertices of positive weight. However, several reduction techniques are considerably more sophisticated—in [9], for instance, three techniques for the MWCSP are described that involve NP-hard subproblems. By adding these preprocessing techniques to SCIP-Jack, not only most MWCSP problems can be solved during preprocessing, but also several instances could be solved for the first time to optimality [6, 9].

Another important component is constituted by heuristics. The current version of SCIP-Jack includes for instance a straightforward greedy heuristic for the PCSTP that starts with a single vertex tree $S_0 = v$ with $v \in V$ and repeatedly connects the current tree S_i to another vertex $w \in V$ with $p_w > 0$ such that this extension leads to a tree S_{i+1} with $C(S_{i+1}) \leq C(S_i)$. This procedure is implemented by a modification of Dijkstra's algorithm. More refined heuristics in SCIP-Jack for both PCSTP and MWCSP are described in [5, 9].

3 Computational results

The computational experiments described in the following were performed on a cluster of Intel Xeon X5672 CPUs with 3.20 GHz and 48 GB RAM. SCIP 4.0.0 was used and CPLEX 12.6[1]) was employed as the underlying LP solver. Moreover, the overall run time for each instance was limited by two hours. If an instance was not solved to optimality within the time limit, the gap is reported, which is defined as $\frac{|\text{pb}-\text{db}|}{\max\{|\text{pb}|,|\text{db}|\}}$ for final primal bound (pb) and dual bound (db). The average gap is obtained as an arithmetic mean. The averages of the number of nodes and the solving time are computed by taking the shifted geometric mean with a shift of 10.0 and 1.0, respectively. For reasons of space we only provide results for STP, PCSTP, and MWCSP.

The results in Table 3 show that the majority of STP instances can be solved within short time. The new version of SCIP-Jack can solve several more instances to optimality than the previous version described in [4]. Also, the run time has been more than halved for the majority of instances.

Table 3: Computational results for Steiner tree problem in graphs

			optimal		timeout	
test set	#	solved	∅ nodes	∅ time [s]	∅ nodes	∅ gap [%]
X	3	3	1.0	0.1	–	–
E	20	20	1.8	0.3	–	–
I640	100	80	23.3	6.6	980.1	0.7
ALUE	15	13	1.3	22.9	1.0	1.9
vienna-i-advanced	85	83	1.8	70.2	1.8	0.0

As can be seen in Tables 4 and 5, with the combination of the new reduction techniques, heuristics and transformations most of the PCSTP and MWCSP instances can be solved easily. Only the PUCNU test set has unsolved instances left. Notably, the results not only demonstrate a speed-up of more than 200% for many instances as compared to the previous version of SCIP-Jack, but also mark a demarcation from other state-of-the-art PCSTP or MWCSP solvers. For example for several problems from the SHINY test, SCIP-Jack outperforms the best run-times reported in the literature [10] by three orders of magnitude and solves problems in less than 0.1 seconds that are intractable for other solvers.

4 Conclusions and outlook

The computational results of SCIP-Jack demonstrate that improved preprocessing and transformation techniques can have a dramatic effect on performance when solving Steiner tree variants. In many cases it is possible to solve problems to optimality even before it is necessary to employ the branch-and-cut kernel.

[1] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Table 4: Computational results for (rooted) prize-collecting Steiner tree problem ((R)PCSTP)

			optimal		timeout	
test set	#	solved	∅ nodes	∅ time [s]	∅ nodes	∅ gap [%]
cologne1	14	14	1.0	0.0	–	–
cologne2	15	15	1.0	0.1	–	–
JMP	34	34	1.0	0.0	–	–
CRR	80	80	1.0	0.2	–	–
PUCNU	18	11	28.0	26.2	865.6	1.8

Table 5: Computational results for maximum-weight connected subgraph problem (MWCSP)

			optimal		timeout	
test set	#	solved	∅ nodes	∅ time [s]	∅ nodes	∅ gap [%]
JMPALMK	72	72	1.0	0.0	–	–
SHINY	39	39	1.0	0.0	–	–
ACTMOD	8	8	1.0	0.1	–	–

In the future we will continue on this path, adding more specific routines, while at the same time improving those sections that apply to all problem variants. The aim is to both improve the run-time of SCIP-Jack, in particular for the STP, and, equally important, tackle additional previously unsolved instances.

5 Acknowledgements

The work for this article has been conducted within the *Research Campus Modal* funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM). It was supported by the Federal Ministry for Economic Affairs and Energy within the BEAM-ME project (ID: 03ET4023A-F). It has been further supported by a Google Faculty Research Award.

References

[1] Karp, R.: Reducibility among combinatorial problems. In Miller, R., Thatcher, J., eds.: Complexity of Computer Computations. Plenum Press (1972) 85–103

[2] Warme, D., Winter, P., Zachariasen, M.: Exact algorithms for plane Steiner tree problems: A computational study. In Du, D.Z., Smith, J., Rubinstein, J., eds.: Advances in Steiner Trees. Kluwer (2000) 81–116

[3] Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G.W., Mutzel, P., Fischetti, M.: An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. Mathematical Programming **105**(2) (Feb 2006) 427–449

[4] Gamrath, G., Koch, T., Maher, S., Rehfeldt, D., Shinano, Y.: SCIP-Jack—a solver for STP and variants with parallelization extensions. Mathematical Programming Computation **9**(2) (2017) 231 – 296

[5] Rehfeldt, D., Koch, T.: Transformations for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem to SAP. Technical Report 16-36, ZIB, Takustr.7, 14195 Berlin (2016)

[6] Rehfeldt, D., Koch, T., Maher, S.: Reduction Techniques for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem. Technical Report 16-47, ZIB, Takustr.7, 14195 Berlin (2016)

[7] Hao, J., Orlin, J.B.: A faster algorithm for finding the minimum cut in a directed graph. J. Algorithms **17**(3) (1994) 424–446

[8] Cherkassky, B.V., Goldberg, A.V.: On implementing the push—relabel method for the maximum flow problem. Algorithmica **19**(4) (1997) 390–410

[9] Rehfeldt, D., Koch, T.: Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem. Technical Report 17-45, ZIB, Takustr.7, 14195 Berlin (2017)

[10] Loboda, A.A., Artyomov, M.N., Sergushichev, A.A. In: Solving Generalized Maximum-Weight Connected Subgraph Problem for Network Enrichment Analysis. Springer International Publishing, Cham (2016) 210–221