ug[SCIP-Jack,*]: A parallel solver for Steiner tree and related problems

Daniel Rehfeldt · Thorsten Koch · Yuji Shinano

TU Berlin

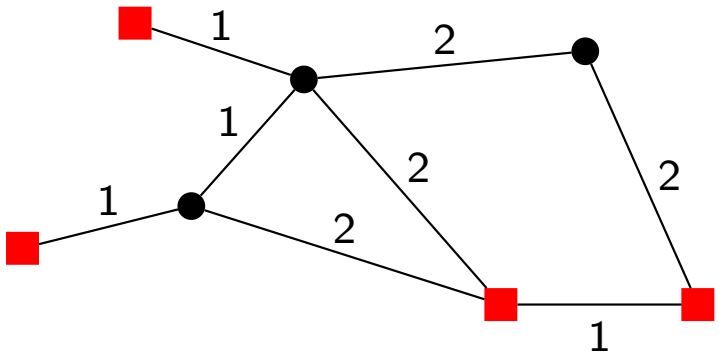
Zuse Institute Berlin

UG Workshop 2021

The Steiner tree problem in graphs

Given:

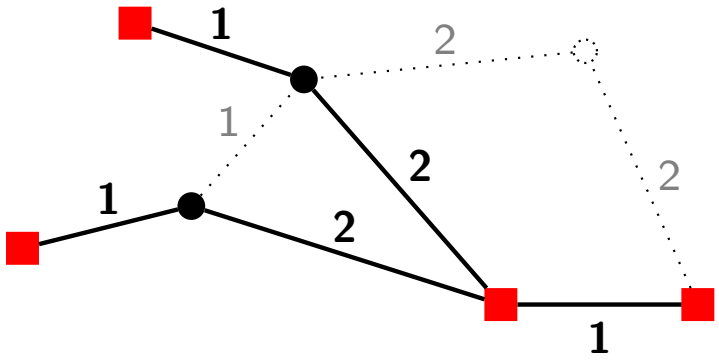
- $G = (V, E)$: undirected graph
- $T \subseteq V$: subset of vertices
- $c \in \mathbb{R}^E_{\geqslant 0}$: non-negative edge costs



The Steiner tree problem in graphs

Given:

- $G = (V, E)$: undirected graph
- $T \subseteq V$: subset of vertices
- $c \in \mathbb{R}^E_{\geqslant 0}$: non-negative edge costs

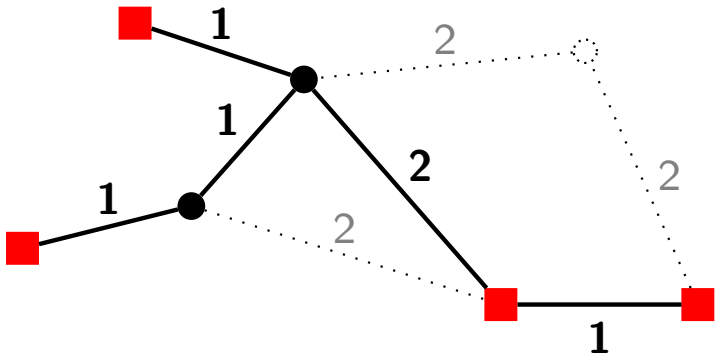


A tree $S \subseteq G$ is called Steiner tree in (G, T, c) if $T \subseteq V(S)$

The Steiner tree problem in graphs

Given:

- $G = (V, E)$: undirected graph
- $T \subseteq V$: subset of vertices
- $c \in \mathbb{R}^E_{\geqslant 0}$: non-negative edge costs



A tree $S \subseteq G$ is called Steiner tree in (G, T, c) if $T \subseteq V(S)$

Steiner tree problem in graphs (SPG)

Find a Steiner tree S in (G, T, c) with minimum edge costs $\sum_{e \in E(S)} c(e)$

Decision variant of SPG is one of Karp's 21 NP-complete problems

- SPG is among the most studied problems in combinatorial optimization
- SPG and close relatives are used in many real-world applications

- SPG is among the most studied problems in combinatorial optimization
- SPG and close relatives are used in many real-world applications

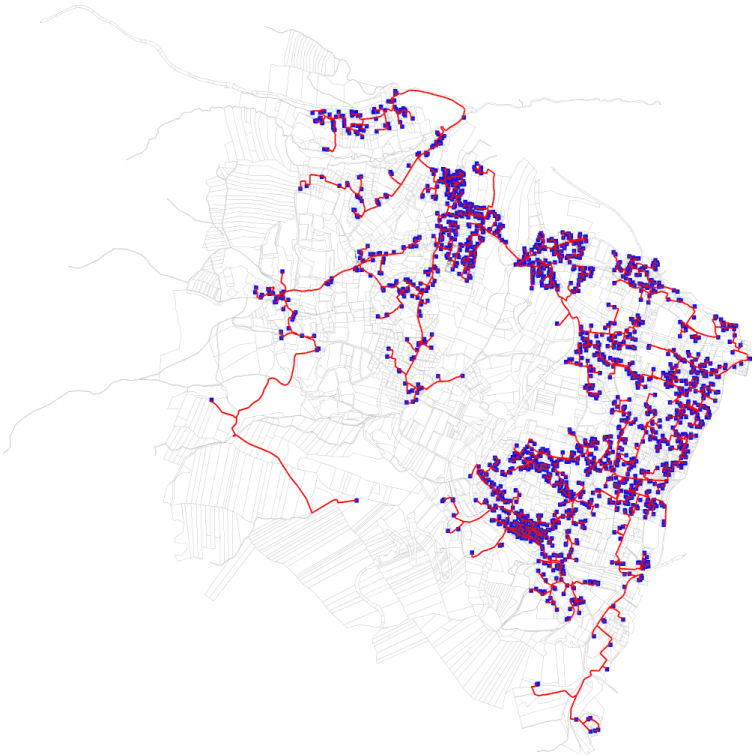
In the following, we describe SCIP-Jack, an exact solver for SPG and 15 related problems, and its parallel extension ug[SCIP-Jack,*].

Why not using a general MIP solver?

Consider exemplary (rather small-scale) network design instance with:

$$|V| = 12\,715$$
$$|E| = 20\,632$$
$$|T| = 475$$

- CPLEX 12.10: No optimal solution after one week



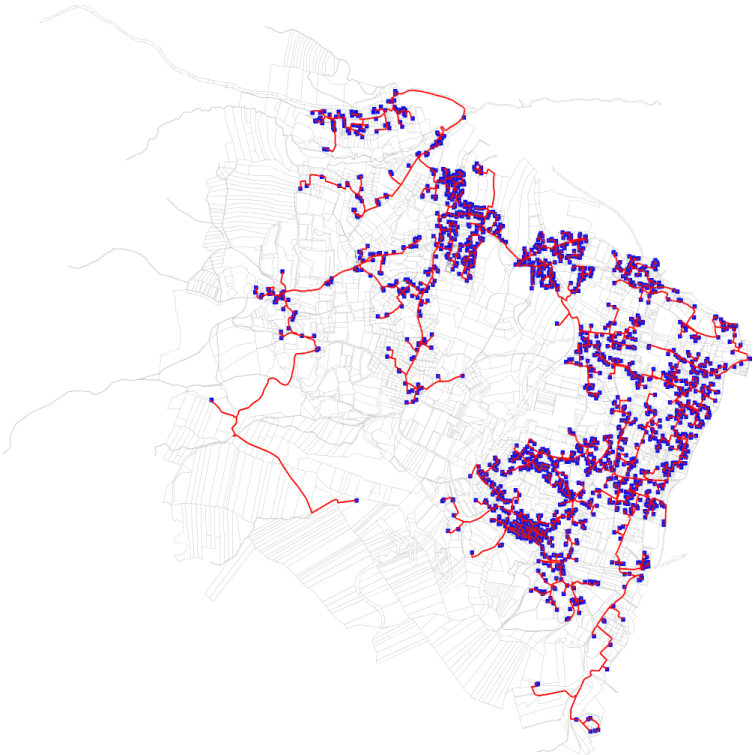
Instance from network telecommunication design for Austrian cities, see *New Real-world Instances for the Steiner Tree Problem in Graphs* (Leitner et al., 2014)

Why not using a general MIP solver?

Consider exemplary (rather small-scale) network design instance with:

$$|V| = 12\,715$$
$$|E| = 20\,632$$
$$|T| = 475$$

- CPLEX 12.10: No optimal solution after one week
- SCIP-Jack: Solves to optimality in 0.2 seconds



Instance from network telecommunication design for Austrian cities, see *New Real-world Instances for the Steiner Tree Problem in Graphs* (Leitner et al., 2014)

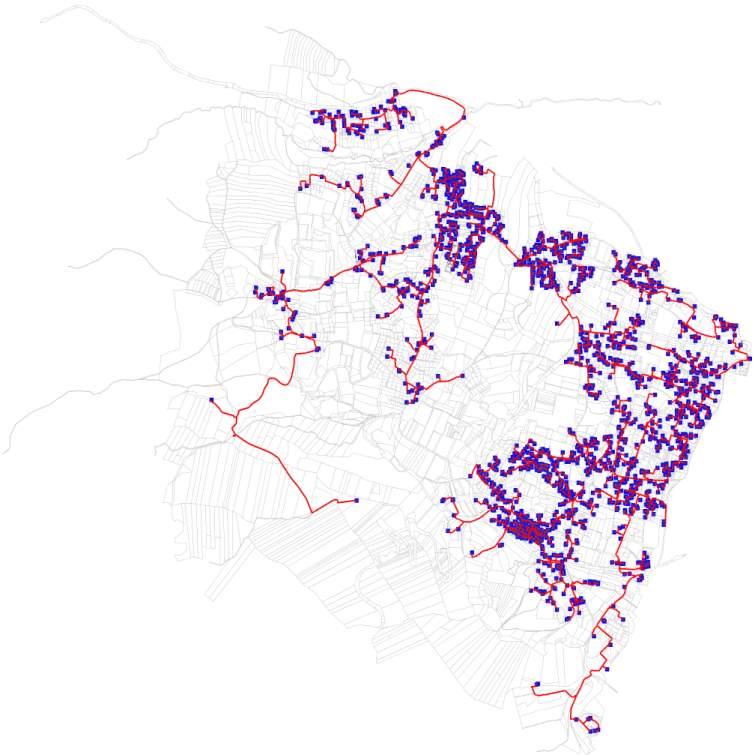
Why not using a general MIP solver?

Consider exemplary (rather small-scale) network design instance with:

$$|V| = 12\,715$$
$$|E| = 20\,632$$
$$|T| = 475$$

- CPLEX 12.10: No optimal solution after one week
- SCIP-Jack: Solves to optimality in 0.2 seconds

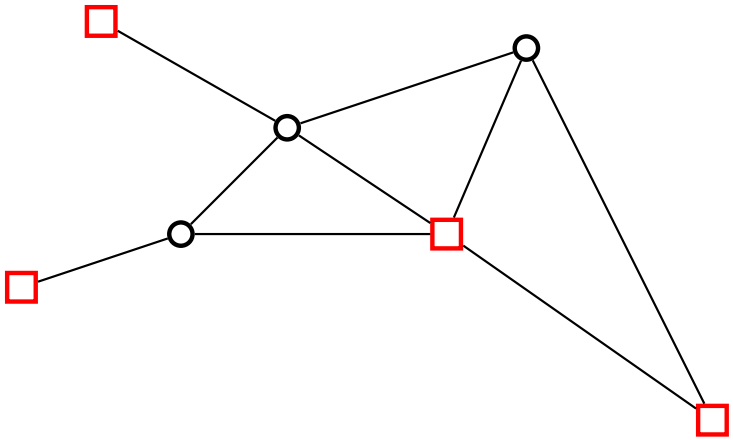
SCIP-Jack is routinely employed for instances with millions of edges in industry

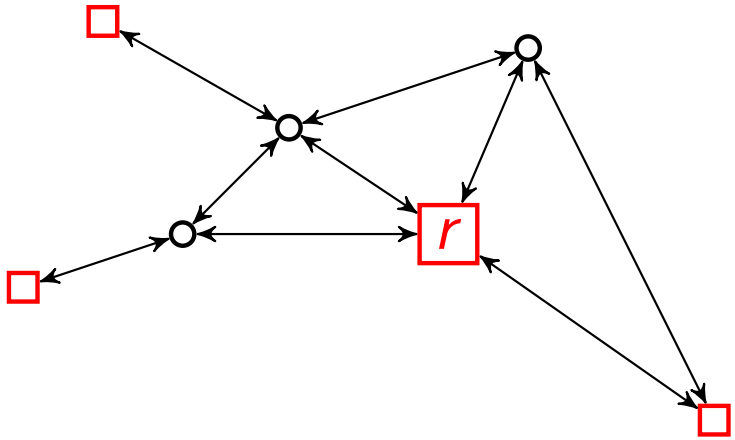


Instance from network telecommunication design for Austrian cities, see *New Real-world Instances for the Steiner Tree Problem in Graphs* (Leitner et al., 2014)

Solving by integer programming

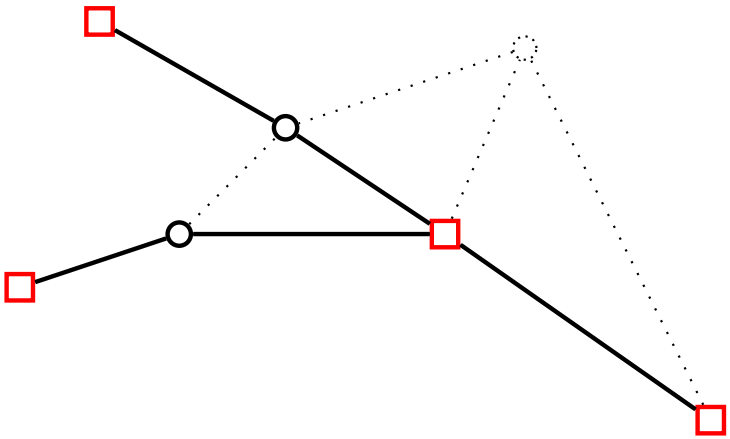
First, transform SPG instance to equivalent (bi-)directed Steiner tree problem.

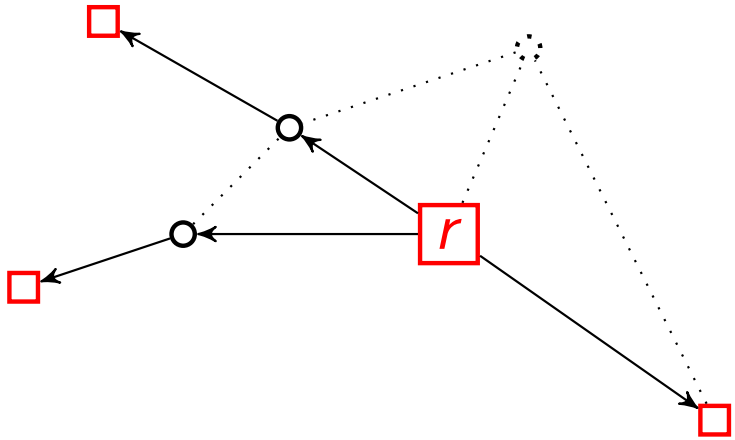




Solving by integer programming

First, transform SPG instance to equivalent (bi-)directed Steiner tree problem.





Formulation

Second, use cutting plane algorithm based on well-known flow-balance directed-cut formulation:

Formulation (Wong, 1984; Duin, 1993)

$$
\begin{array}{rcll}
\min c^T y & & & \\
y(\delta^+(W)) & \geqslant & 1 & \text{for all } W \subset V, r \in W, (V \setminus W) \cap T \neq \emptyset \\
y(\delta^-(v)) & \leqslant & y(\delta^+(v)) & \text{for all } v \in V \setminus T \\
y(a) & \in & \{0,1\} & \text{for all } a \in A
\end{array}
$$

Formulation

Second, use cutting plane algorithm based on well-known flow-balance directed-cut formulation:

Formulation (Wong, 1984; Duin, 1993)

$$
\begin{array}{rcll}
\min c^T y & & & \\
y(\delta^+(W)) & \geqslant & 1 & \text{for all } W \subset V, r \in W, (V \setminus W) \cap T \neq \emptyset \\
y(\delta^-(v)) & \leqslant & y(\delta^+(v)) & \text{for all } v \in V \setminus T \\
y(a) & \in & \{0,1\} & \text{for all } a \in A
\end{array}
$$

Theorem (Rehfeldt, Franz, Koch, 2020)

If $|T| < 4$, then flow-balance directed-cut formulation has tight LP relaxation.

The framework: SCIP (Solving Constraint Integer Programs)

Functionality

- full-scale mixed-integer optimizer for general MIPs and MINLPs
- generic framework for implementing problem-specific branch-cut-and-price

SCIP-Jack: A Steiner tree solver

SCIP-Jack works by combining generic and problem specific algorithms:

SCIP-Jack: A Steiner tree solver

SCIP-Jack works by combining generic and problem specific algorithms:

- *examples for generic*
  - fast separator routine based on new max-flow implementation
  - dual-ascent heuristic
- *examples for problem specific*
  - efficient transformations to directed Steiner tree problem (needed for applying generic separator)
  - preprocessing routines
  - domain propagation
  - dynamic-programming algorithms
  - primal heuristics

Current development version of SCIP-Jack consists of $> 110\,000$ lines of code.

Computational results: PACE Challenge 2018

The goal of the Parameterized Algorithms and Computational Experiments (PACE) Challenge is to investigate the applicability of algorithmic ideas studied and developed in the subfields of parameterized, or fixed-parameter tractable algorithms.

PACE 2018:

- dedicated to Steiner tree problems in graphs, for which several fixed-parameter tractable (FPT) algorithms are known
- Tracks A,B,C, each including 100 instances, time limit 30 min.
- took place in May 2018, 40 participating teams from 16 countries

PACE 2018

SCIP-Jack[1] with SoPlex as LP solver competed in all three tracks (although did not include any FPT algorithms). Result:

- Track A (exact, few terminals): *3rd place*[2]
- Track B (exact, low treewidth): *1st place*
- Track C (heuristic, different FPT structures): *2nd place*[3]

[1] Team members: T. Koch, D. Rehfeldt

[2] Winning team: Yoichi Iwata, Takuto Shigemura (NII, Japan)

[3] Winning team: Emmanuel Romero Ruiz, Emmanuel Antonio Cuevas, Irwin Enrique Villalobos López, Carlos Segura González (CIMAT, Mexico)

PACE 2018

Current SCIP-Jack significantly outperforms PACE version:

		SCIP-Jack-NEW		SCIP-Jack-PACE		Best other
Track	# instances	solved	avg. time [s]	solved	avg. time [s]	solved
A	100	**99**	**38**	94	111	95
B	100	**99**	**25**	92	132	77

Table: Updated computational results for PACE 2018 instances.

(also much stronger on heuristic Track C)

Computational results (2)

- Fastest SPG solver established by Polzin and Vahdati Daneshmand 20 years ago (non-public)
- Drastically outperforms best other SPG solvers, even the winning solvers from DIMACS Challenge 2014 and PACE Challenge 2018 (did not participate in either challenge)
- ...we made a large effort in last two years to outperform Polzin and Vahdati Daneshmand

Consider 14 benchmark test-sets from the literature (almost all non-trivial sets from DIMACS Challenge and SteinLib library). Time limit of 24 hours, results of (non-public) solver by Polzin, Vahdati are scaled for fair comparison.

		# solved		mean time (sh. geo. mean)			maximum time		
Test-set	#	P.&V.	S.-J.	P.&V. [s]	S.-J. [s]	speedup	P.&V. [s]	S.-J. [s]	speedup
VLSI	116	116	116	0.5	0.8	0.63	53.9	83.3	0.65
TSPFST	76	76	76	1.5	1.1	**1.36**	1161.4	263.1	**4.41**
WRP4	62	62	62	3.2	2.4	**1.33**	106.1	90.8	**1.17**
2R	27	27	27	5.0	3.0	**1.67**	43.9	21.1	**2.08**
vienna-a	85	85⋆	85	7.2	5.2	**1.38**	441.3	59.0	**7.48**
vienna-s	85	85⋆	85	7.8	6.2	**1.26**	623.5	60.7	**10.27**
WRP3	63	63	63	22.8	13.5	**1.69**	6073.2	4886.4	**1.24**
GEO-a	23	23⋆	23	158.7	55.3	**2.87**	6476.5	880.9	**7.35**
GEO-org	23	23⋆	23	145.6	58.5	**2.49**	4385.0	842.1	**5.21**
ES10000	1	1	1	138.0	83.0	**1.66**	138.0	83.0	**1.66**
Cophag14	21	20⋆	**21**	27.7	13.8	**2.01**	>86400	3845.3	>**22.47**
SP	8	6	**8**	159.4	25.8	**6.18**	>86400	1688.3	>**51.18**
LIN	37	35	**36**	31.3	14.7	**2.13**	>86400	>86400	1.00
PUC	50	17⋆	**18**	14964.9	11901.1	**1.26**	>86400	>86400	1.00

Shift $s = 10$ used for shifted geometric mean.
Times marked by a $\star$ were obtained by P.&V. with (specialized) non-default settings.

Consider aggregated results. Group $[x, 86400]$: all instances solved by at least one solver in at least x and at most 86400 seconds.

		shifted geometric mean time			arithmetic mean time		
Group	#	P.&V. [s]	S.-J. [s]	speedup	P.&V. [s]	S.-J. [s]	speedup
$[0, 86400]$	644	12.2	7.9	**1.54**	1235.5	264.9	**4.66**
$[1, 86400]$	342	34.5	19.6	**1.76**	2326.4	498.7	**4.66**
$[10, 86400]$	180	122.5	52.0	**2.36**	4417.1	944.6	**4.68**
$[100, 86400]$	66	1403.2	286.6	**4.90**	11999.0	2546.3	**4.71**
$[1000, 86400]$	30	8035.8	1096.9	**7.33**	25923.1	5435.6	**4.77**

Table: Computational comparison of our solver (*S.-J.*) and solver by Polzin, Vahdati (*P.&V.*), with instance groups ordered by hardness.

Steiner tree relatives

SCIP-Jack significantly outperforms specialized solvers for several important Steiner tree relatives, e.g. for

- Maximum-weight connected subgraph problem
- Prize-collecting Steiner tree problem
- Geometric Steiner tree problems (for concatenation, see next slide)

SCIP-Jack also useful for solving Euclidean Steiner tree problem:
Can be used instead of commonly used hypergraphic concatenation.

Comparison on largest instances from the literature with state-of-the-art geometric Steiner tree solver GeoSteiner 5.1 (Juhl et al., 2018):

1) 15 instances with 50 000 terminals in the plane:

- GeoSteiner solves 8 within one week
- SCIP-Jack solves all 15 in three minutes

SCIP-Jack also useful for solving Euclidean Steiner tree problem:
Can be used instead of commonly used hypergraphic concatenation.

Comparison on largest instances from the literature with state-of-the-art geometric Steiner tree solver GeoSteiner 5.1 (Juhl et al., 2018):

1) 15 instances with 50 000 terminals in the plane:

- GeoSteiner solves 8 within one week
- SCIP-Jack solves all 15 in three minutes

2) 15 instances with 100 000 terminals in the plane:

- GeoSteiner solves 3 within one week
- SCIP-Jack solves all 15 in 11 minutes

$\rightarrow$ 19 instances solved for the first time to optimality.

Going parallel

Two approaches within SCIP-Jack:

- parallelizing reduction techniques and heuristics (shared-memory only)
- parallelizing branch-and-bound search (shared- and distributed-memory)

Going parallel

Two approaches within SCIP-Jack:

- parallelizing reduction techniques and heuristics (shared-memory only)
- parallelizing branch-and-bound search (shared- and distributed-memory) $\rightarrow$ using UG

ug[SCIP-*,*]-libraries

- Customized SCIP solvers (such as SCIP-Jack) include a set of SCIP plugins, to be integrated in the solution process

ug[SCIP-*,*]-libraries

- Customized SCIP solvers (such as SCIP-Jack) include a set of SCIP plugins, to be integrated in the solution process
- ug[SCIP-*,*]-libraries are general purpose libraries that allow one to parallelize the branch-and-bound search of such customized SCIP solvers

ug[SCIP-*,*]-libraries

- Customized SCIP solvers (such as SCIP-Jack) include a set of SCIP plugins, to be integrated in the solution process
- ug[SCIP-*,*]-libraries are general purpose libraries that allow one to parallelize the branch-and-bound search of such customized SCIP solvers
- Usually, fewer than 200 lines of glue code are required for this parallelization
- ...but some debugging maybe be necessary, especially when using SCIP in a non-standard way

Parallelization of SCIP-Jack by UG

- UG framework to parallelize B&B search both in shared, ug[SCIP-Jack, C++11 threads], and distributed, ug[SCIP-Jack, MPI], environments.
- Parallelization can be realized with a few lines of code.

Parallelization of SCIP-Jack by UG

- UG framework to parallelize B&B search both in shared, ug[SCIP-Jack, C++11 threads], and distributed, ug[SCIP-Jack, MPI], environments.
- Parallelization can be realized with a few lines of code.
- ... but to improve performance both UG and SCIP-Jack had to be extended.
- Difficulties:
  - Long running time in root node.
  - Special branching.
  - Distributing problem specific preprocessing effects.

Branching on vertices

SCIP-Jack branches on vertices of the graph.
Including vertex v corresponds to the constraint:

$$\sum_{a \in \delta^-(v)} x_a = 1$$

Excluding vertex v corresponds to constraint:

$$\sum_{a \in \delta^-(v) \cup \delta^+(v)} x_a = 0$$

$\rightarrow$ added new features to UG to allow branching on constraints

Building new subproblems

- To retain previous branching decisions, SCIP-Jack/UG allows to transfer the branching history together with a subproblem. SCIP-Jack then changes the underlying graph (adds terminals and deletes vertices).
- Improves finding heuristically locally valid solutions and helps with cut generation. Local cuts are also transferred by UG.

Aggressive presolving of subproblems

Strong point of UG: presolving of subproblems during branch-and-bound.

Problem:

- For Steiner tree problems MIP presolving quite unsuccessful.
- Specialized Steiner tree presolving is performed before SCIP initiated (for performance reasons)

$\rightarrow$ needed to adapt SCIP-Jack to perform Steiner tree presolving also after transfer of new branch-and-bound node to a UG solver

SCIP presolving vs. SCIP-Jack presolving

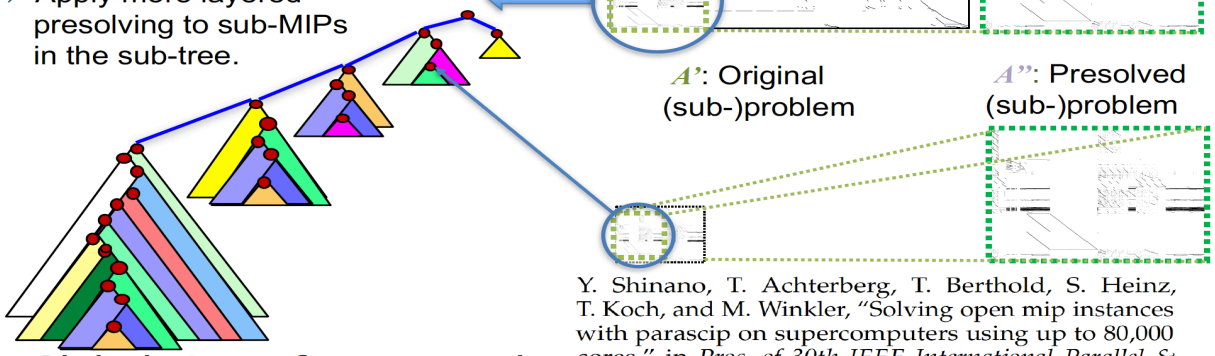
$$\min\{c'^{\top} x : A'x \leq b', l' \leq x \leq u', x_j \in \mathbb{Z}, \text{ for all } j \in I'\}$$

What does the Dynamic load balancing work?

Searches hardest part of sub-trees

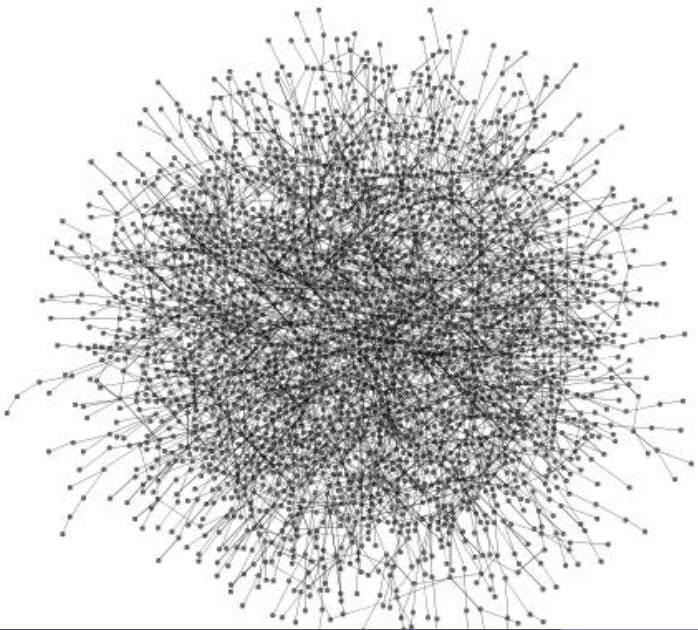
⇒ Assign more computing resources to the sub-trees

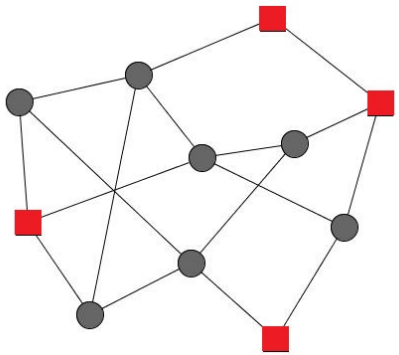
⇒ Apply more layered presolving to sub-MIPs in the sub-tree.



Global view of tree search

Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler, "Solving open mip instances with parascip on supercomputers using up to 80,000 cores," in *Proc. of 30th IEEE International Parallel & Distributed Processing Symposium*, 2016.





Parallel results

Using up to 43 thousand cores on supercomputers, we

- Solved five notoriously hard SPG instances for the first time to optimality
- Solved two prize-collecting Steiner tree instances from 11th DIMACS Challenge (results not published)

Solving benchmark instance hc9p to optimality

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes
1.1	ISM	72	604,796	< 0.3	738	30,242.0000	29,879.3721	1.21	0
			(317)			30,242.0000	30,058.9366	0.61	110,012,624
1.2	ISM	2,304	604,794	< 1.5	979,695	30,242.0000	30,058.7930	0.61	0
						30,242.0000	30,102.7556	0.46	3,758,532,600
1.3	HLRN III	24,576	86,336	< 1.7	8,811,512	30,242.0000	30,102.6645	0.46	0
						30,242.0000	30,116.3592	0.42	2,402,406,311
1.4	HLRN III	12,288	43,199	< 1.5	1,709,027	30,242.0000	30,115.3331	0.42	0
						30,242.0000	30,120.4801	0.40	664,909,985
1.5	HLRN III	12,288	118,259	1.5	9,158,920	30,242.0000	30,120.4801	0.40	0
						30,242.0000	30,242.0000	0.00	1,677,724,126

Table: Statistics for solving `hc9p` on supercomputers.

Resources:
ISM: HPE SGI 8600, 384 compute nodes, each node consisting of two Intel Xeon Gold 6154 3.0GHz CPUs (18 cores$\times$2) sharing 384GB of memory, and an Infiniband (Enhanced Hypercube) interconnect.
HLRN III: Cray XC40 with 1872 compute nodes, each node consisting of two 12-core Intel Xeon IvyBridge/Haswell CPUs sharing 64 GiB of RAM, with Aries interconnect.

Future work

Possible future work:

- combine internal parallelization of SCIP-Jack with UG (by dynamic thread allocation)
- solve further classic unsolved Steiner tree benchmark instances by using ug[SCIP-Jack, MPI]

New SCIP-Jack (and ug[SCIP-Jack, *]) will be made freely available for academic use as part of SCIP Optimization Suite 8 (`www.scipopt.org`).

References

- Rehfeldt, Shinano, Koch
  SCIP-Jack: An Exact High Performance Solver for Steiner Tree Problems in Graphs and Related Problems, HPSC 2018
- Shinano, Rehfeldt, Koch., *Building Optimal Steiner Trees on Supercomputers by Using up to 43,000 Cores*, CPAIOR 2019
- Rehfeldt, Koch, *Combining NP-Hard Reduction Techniques and Strong Heuristic in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem*, SIAM Journal on Optimization, 2019
- Rehfeldt, Koch, *Implications, Conflicts, and Reductions for Steiner Trees*, *IPCO 2021*
- Rehfeldt, Koch, *On the exact solution of prize-collecting Steiner tree problems*, INFORMS Journal on Computing, 2021

Thank you!