ug[SCIP-Jack, MPI]: A Massively Parallel Steiner Tree Solver

Daniel Rehfeldt · Yuji Shinano · Thorsten Koch

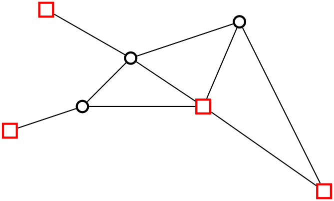
Zuse Institute Berlin

Technische Universität Berlin

UG19, Berlin, January 2019

Given:

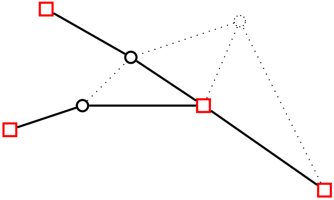
- $G = (V, E)$: undirected graph
- $T \subseteq V$: subset of vertices
- $c \in \mathbb{R}^{E}_{>0}$: positive edge costs



The Steiner Tree Problem in Graphs

Given:

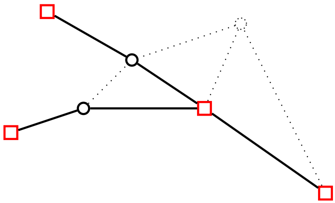
- $G = (V, E)$: undirected graph
- $T \subseteq V$: subset of vertices
- $c \in \mathbb{R}^{E}_{>0}$: positive edge costs



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

Given:

- $G = (V, E)$: undirected graph
- $T \subseteq V$: subset of vertices
- $c \in \mathbb{R}^E_{>0}$: positive 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 $\sum_{e \in E(S)} c(e)$

SPG is one of the fundamental combinatorial optimization problems; decision variant is one of Karp's 21 $\mathcal{NP}$-complete problems.

Why not using a general MIP solver?

Consider (small-scale) network design instance with:

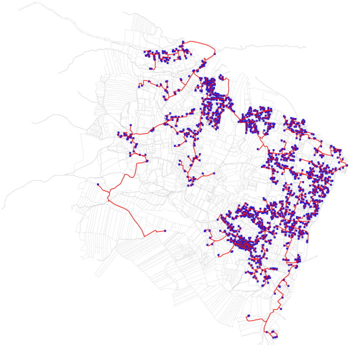
$$|V| = 12\,715$$

$$|E| = 41\,264$$

$$|T| = 475$$

- ▷ CPLEX 12.7.1: No optimal solution within 72 hours
- ▷ SCIP-Jack: Solves to optimality in 7.5 seconds

For larger problems CPLEX runs out of memory almost immediately (largest real-world instance SCIP-Jack solved so far has 64 million edges, 11 million vertices)

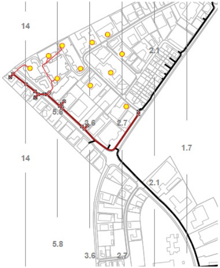


Network telecommunication design for Austrian cities, see *New Real-world Instances for the Steiner Tree Problem in Graphs* (Leitner et al., 2014)

Applications

Some real-world applications of Steiner trees:

- ▷ design of fiber optic networks
- ▷ prediction of tumor evolution
- ▷ deployment of drones
- ▷ computer vision
- ▷ wire routing
- ▷ computational biology
- ▷ . . .



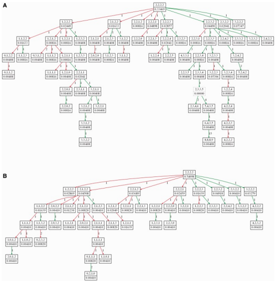
Rooted prize-collecting Steiner tree problem

E.g. *An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem* (Ljubic et al., 2006)

Applications

Some real-world applications of Steiner trees:

- ▷ design of fiber optic networks
- ▷ prediction of tumor evolution
- ▷ deployment of drones
- ▷ computer vision
- ▷ wire routing
- ▷ computational biology
- ▷ ...



Rectilinear Steiner minimum tree problem

E.g. *Phylogenetic analysis of multiprobe fluorescence in situ hybridization data from tumor cell populations* (Chowdhury et al., 2013)

Applications

Some real-world applications of Steiner trees:

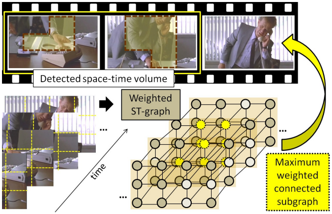
- design of fiber optic networks
- prediction of tumor evolution
- deployment of drones
- computer vision
- wire routing
- computational biology
- ...

Hop-constrained directed Steiner tree problem

E.g. *Local Search for Hop-constrained Directed Steiner Tree Problem with Application to UAV-based Multi-target Surveillance* (Burdakov, 2014)

Some real-world applications of Steiner trees:

- design of fiber optic networks
- prediction of tumor evolution
- deployment of drones
- computer vision
- wire routing
- computational biology
- ...

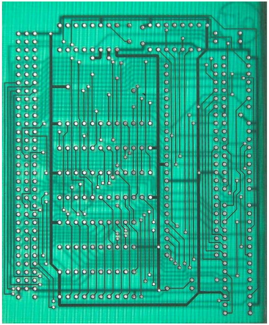


Maximum-weight connected subgraph problem

E.g. *Efficient activity detection with max-subgraph search* (Chen, Grauman, 2012)

Some real-world applications of Steiner trees:

- design of fiber optic networks
- prediction of tumor evolution
- deployment of drones
- computer vision
- write routing
- computational biology
- ...

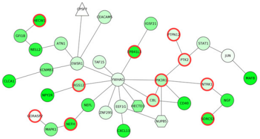


Group Steiner tree problem

E.g. *Rectilinear group Steiner trees and applications in VLSI design* (Zachariasen, Rohe, 2003)

Some real-world applications of Steiner trees:

- ▷ design of fiber optic networks
- ▷ prediction of tumor evolution
- ▷ deployment of drones
- ▷ computer vision
- ▷ wire routing
- ▷ computational biology
- ▷ . . .



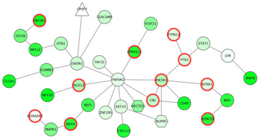
Maximum-weight connected subgraph problem

E.g. *Solving Generalized Maximum-Weight Connected Subgraph Problems for Network Enrichment Analysis* (Loboda et al., 2016)

Applications

Some real-world applications of Steiner trees:

- design of fiber optic networks
- prediction of tumor evolution
- deployment of drones
- computer vision
- wire routing
- computational biology
- . . .



Maximum-weight connected subgraph problem

Real-world applications usually require variations of SPG

What we wanted: Solver for Steiner tree and many related problems

How it started...Jack

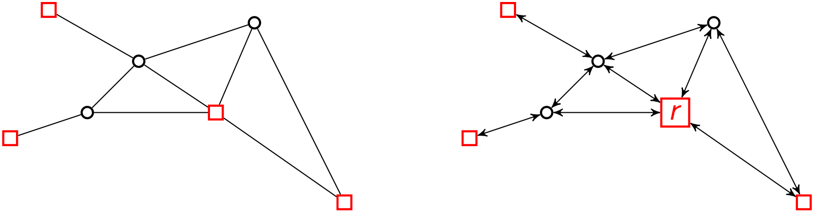
What we wanted: Solver for Steiner tree and many related problems

What we had: an old solver for SPG, Jack-III

What we wanted: Solver for Steiner tree and many related problems

What we had: an old solver for SPG, Jack-III; based on

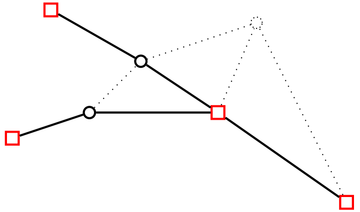
- transformation into a Steiner arborescence problem and ...

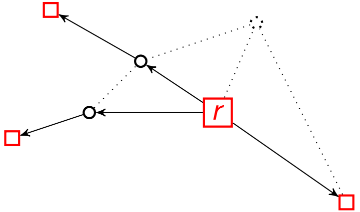


What we wanted: Solver for Steiner tree and many related problems

What we had: an old solver for SPG, Jack-III; based on

- ▷ transformation into a Steiner arborescence problem and ...

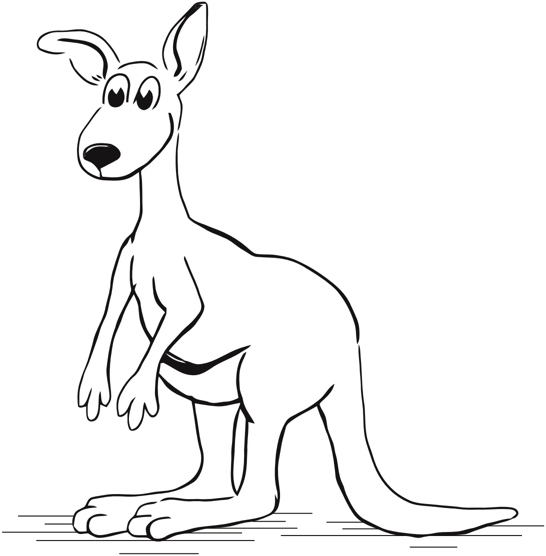




... cutting plane algorithm based on flow balance directed-cut formulation:

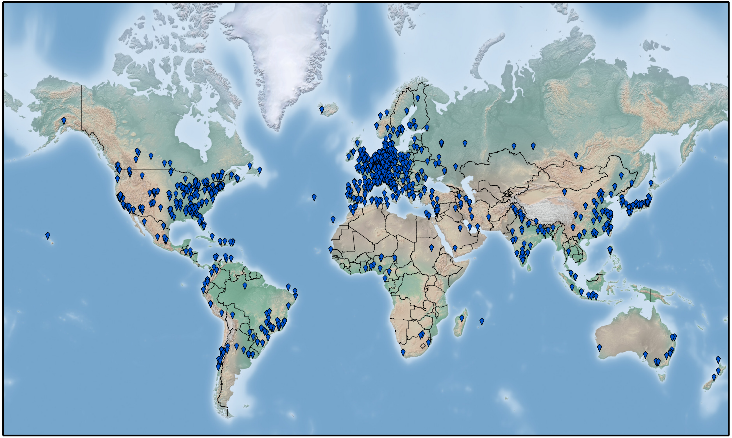
Formulation

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



- general setup
  - plugin based system
  - default plugins handle MIPs and nonconvex MINLPs
  - support for branch-and-price and custom relaxations
- documentation and guidelines
  - more than 500 000 lines of C code, 20% documentation
    - 36 000 assertions, 5 000 debug messages
  - HowTos: plugins types, debugging, automatic testing
  - 11 examples and 5 applications illustrating the use of SCIP
  - active mailing list `scip@zib.de` (300 members)
- interface and usability
  - user-friendly interactive shell
  - interfaces to AMPL, GAMS, ZIMPL, MATLAB, Python and Java
  - `C++` wrapper classes
  - LP solvers: CLP, CPLEX, Gurobi, MOSEK, QSopt, SoPlex, Xpress
  - over 1 600 parameters and 15 emphasis settings

(Some) Universities and Institutes using SCIP



about 8000 downloads per year

SCIP

Jack

SCIP-Jack

Development of SCIP-Jack has been joint work with:
Gerald Gamrath · Thorsten Koch · Stephen J. Maher · Yuji Shinano

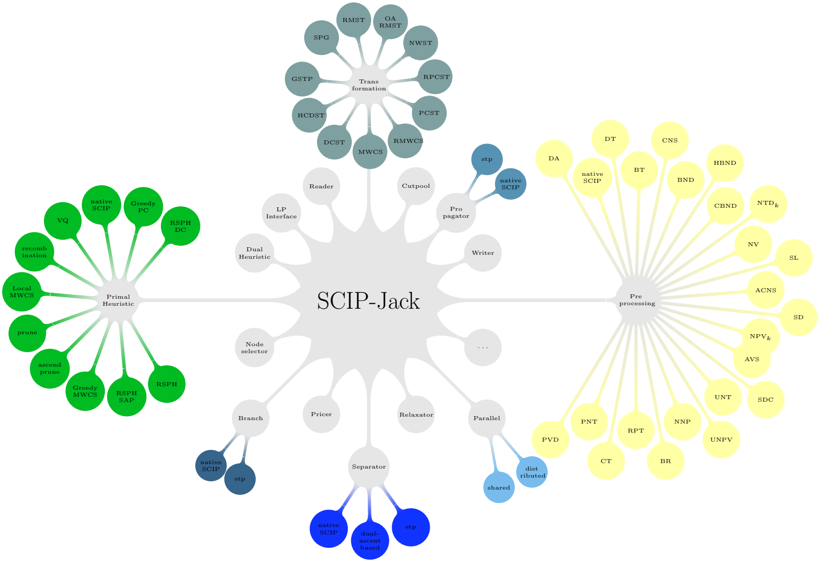
SCIP-Jack can solve SPG and 12 related problems:

Abbreviation	Problem Name
SPG	Steiner tree problem in graphs
SAP	Steiner arborescence problem
RSMT	Rectilinear Steiner minimum tree problem
OARSMT	Obstacle-avoiding rectilinear Steiner minimum tree problem
NWSTP	Node-weighted Steiner tree problem
NWPTSTP	Node-weighted partial terminal Steiner tree problem
PCSTP	Prize-collecting Steiner tree problem
RPCSTP	Rooted prize-collecting Steiner tree problem
MWCSP	Maximum-weight connected subgraph problem
RMWCSP	Rooted maximum-weight connected subgraph problem
DCSTP	Degree-constrained Steiner tree problem
GSTP	Group Steiner tree problem
HCDSTP	Hop-constrained directed Steiner tree problem

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

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

- generic
  - extremely fast separator routine based on new max-flow implementation
  - all general methods provided by SCIP
    e.g., generic cutting planes
- problem specific
  - efficient transformations to Steiner arborescence problem (needed for applying generic separator)
  - preprocessing and propagation routines
  - primal and dual heuristics


SCIP-Jack
Trans formation
RMST
OA RMST
SPG
NWST
GSTP
RPCST
HCDST
PCST
DCST
MWCS
RMWCS
Reader
Cutpool
LP Interface
Pro pagator
stp
native SCIP
Dual Heuristic
Writer
Primal Heuristic
native SCIP
Greedy PC
VQ
RSPH DC
recomb ination
Local MWCS
prune
ascend prune
Greedy MWCS
RSPH SAP
RSPH
Pre processing
DA
DT
CNS
native SCIP
BT
HBND
BND
CBND
NTD$_k$
NV
SL
ACNS
SD
NPV$_k$
AVS
UNT
SDC
PVD
PNT
RPT
NNP
UNPV
CT
BR
Node selector
. . .
Branch
native SCIP
stp
Pricer
Relaxator
Parallel
shared
dist ributed
Separator
native SCIP
dual-ascent based
stp

Performance of SCIP-Jack for SPG

SCIP-Jack is

- usually between two and three orders of magnitude faster than Jack-III (both using CPLEX 12.7.1 as LP-solver)
- fastest publicly available solver for SPG
- also fastest solver for several related problems...

PACE 2018 https://pacechallenge.wordpress.com/pace-2018

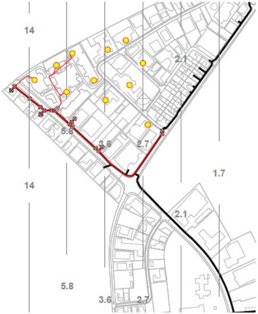
The Parameterized Algorithms and Computational Experiments Challenge

fixed-parameter tractable instances; more than 100 participants

Setting: 3 tracks, 100 instances in each track, time limit 30 min.

- Track A: exact, few terminals; results on public instances:
  - Winner: 94/100 Iwata, Shigemura (NII, Japan)
  - SCIP-Jack/SoPlex: 93/100 3rd place
  - SCIP-Jack/CPLEX: 100/100
- Track B: exact, low treewidth; results on public instances:
  - SCIP-Jack/SoPlex: 98/100 1st place
  - SCIP-Jack/CPLEX: 99/100
  - best other: 84/100
- Track C: heuristic, mean ratio to best known upper bound:
  - Winner: 99.92 Ruiz, Cuevas, López, González (CIMAT, Mexico)
  - SCIP-Jack/SoPlex: 99.81 2nd place
  - SCIP-Jack/CPLEX: 100

Prize-collecting Steiner tree problem



Given:

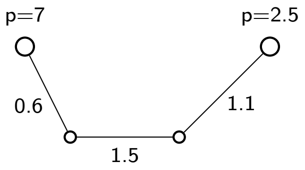
- $\triangleright$ undirected graph $G = (V, E)$
- $\triangleright$ vertex costs $p \in \mathbb{R}^V_{\geqslant 0}$
- $\triangleright$ edge costs $c \in \mathbb{R}^E_{\geqslant 0}$

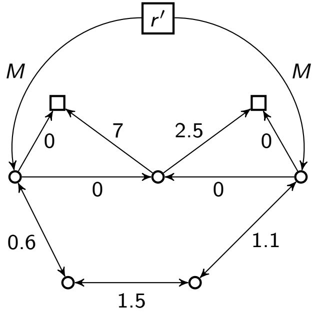
Prize-Collecting Steiner Tree Problem (PCSTP)

Find a tree $S \subseteq G$ such that

- $\triangleright$ $\sum_{e \in E(S)} c(e) + \sum_{v \in V \setminus V(S)} p(v)$ is minimized

$T_p := \{v \in V \mid p(v) > 0\}$ are called potential terminals.

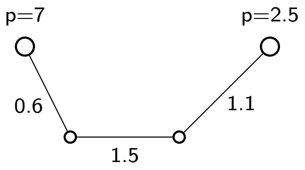


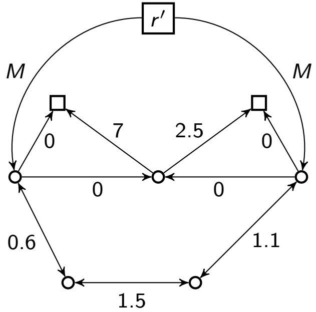


Transformation to SAP ...

- ▷ allows to use powerful cut-separation routine of SCIP-Jack
- ▷ allows to employ strong heuristics based on a dual-ascent algorithm for SAP

Transformation: PCSPG to SAP





Transformation to SAP ...

- allows to use powerful cut-separation routine of SCIP-Jack
- allows to employ strong heuristics based on a dual-ascent algorithm for SAP

works, but not enough to be competitive...

Some recent work ...

Definition

Let $v_i, v_j \in V$. Call walk $W = (v_{i_1}, e_{i_1}, v_{i_2}, e_{i_2}, ..., e_{i_r}, v_{i_r})$ with $v_{i_1} = v_i$ and $v_{i_r} = v_j$ *prize-constrained* (v_i, v_j)*-walk* if no $v \in T_p \cup \{v_i, v_j\}$ contained more than once in W.

Definition

Define *prize-collecting cost* of W as

$$c_{pc}(W) := \sum_{e \in E(W)} c(e) - \sum_{v \in V(W) \setminus \{v_i, v_j\}} p(v).$$

Definition

Define *prize-constrained length* of W:

$$l_{pc}(W) := \max\{c_{pc}(W(v_{i_k}, v_{i_l})) \mid 1 \leqslant k \leqslant l \leqslant r,\ v_{i_k}, v_{i_l} \in T_p \cup \{v_i, v_j\}\}.$$

Let $\mathcal{W}_{pc}(v_i, v_j)$ prize-constrained (v_i, v_j)-walks define *prize-constrained distance* between v_i and v_j:

$$d_{pc}(v_i, v_j) := \min\{l_{pc}(W') \mid W' \in \mathcal{W}_{pc}(v_i, v_j)\}.$$

Proposition

Let $\{v_i, v_j\} \in E$. If

$$c(\{v_i, v_j\}) > d_{pc}(v_i, v_j)$$

is satisfied, then $\{v_i, v_j\}$ cannot be contained in any optimal solution.

Prize-constrained concept allows for very powerful reduction tests; dominates previous concept from Uchoa 2006 (Oper. Res. Let.) and generalizes tests known for SPG.

Downside: NP-hard. But: Nice (easy to implement) approximation by an extension of Dijkstra's algorithm.

Another use of prize-constrained walks:

Definition

Let W be prize-constrained (v_i, v_j) walk. Define *left-rooted prize-constrained length* of W as:

$$l^{-}_{pc}(W) := \max\{c_{pc}(W(v_i, v_{i_k})) \mid v_{i_k} \in V(W) \cap (T_p \cup \{v_j\})\}.$$

Definition

Define *left-rooted prize-constrained* (v_i, v_j)*-distance* as:

$$d^{-}_{pc}(v_i, v_j) := \min\{l^{-}_{pc}(W') \mid W' \in \mathcal{W}_{pc}(v_i, v_j)\}.$$

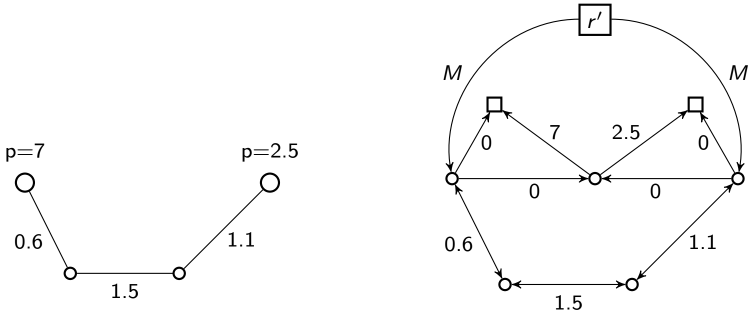
Proposition

Let $v_i, v_j \in V$. If

$$p(v_i) > d^-_{pc}(v_i, v_j),$$

then every optimal solution that contains v_j also contains v_i.

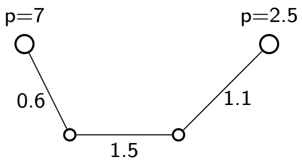
Using left-rooted prize-constrained walks

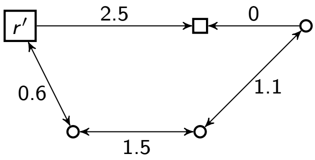


- Use prize-constrained walks to identify terminals t_i that need to be part of all optimal solutions
- ...allows for better transformation to SAP:

INPUT: RPCSTP (V, E, T_f, c, p) and $t_p, t_q \in T_f$ $T_f := \{t_1, t_2, ..., t_z\}$ fixed terminals
OUTPUT: SAP

1. Set $V' := V$, $A' := \{(v, w) \in V' \times V' \mid \{v, w\} \in E\}$, $c' := c$, $r' := t_q$.
2. For each $i \in \{1, ..., z\}$:
   2.1 add node t'_i to V',
   2.2 add arc (t_i, t'_i) of weight 0 to A',
   2.3 add arc (t_p, t'_i) of weight $p(t_i)$ to A'.
3. Define set of terminals $T' := \{t'_1, ..., t'_z\} \cup T_f$.
4. **Return** (V', A', T', c', r').





We have recently proved: Choice of t_p, t_q does not change LP value! But: Can have strong impact in practival solving, natural candidate for UG racing ramp up parameter!

Prize-constrained walks in branch-and-cut

- preprocessing
- probing
- propagation
- cutting planes
- primal heuristics
- dual heuristics
- B&B

We can exploit that when using UG!

Performance of SCIP-Jack on PCSTP

Performance of SCIP-Jack on PCSTP

Many PCSTP solvers introduced in the literature lately, the two best:

- ▷ Fischetti et. al. 2017 (Math. Prog. C)
- ▷ Leitner et. al. 2018 (INFORMS J. Comput.) ... stronger by far

Test set	#	mean time [s] L18	mean time [s] SJ	max. time [s] L18	max. time [s] SJ	# solved L18	# solved SJ
JMP	34	0.0	0.0	0.0	0.0	34	34
Cologne1	14	0.0	0.0	0.1	**0.0**	14	14
Cologne2	15	0.1	0.1	0.2	**0.1**	15	15
CRR	80	0.1	0.1	5.7	**1.1**	80	80
ACTMOD	8	0.9	**0.3**	3.5	**1.5**	8	8
E	40	1.8	**0.2**	>3600	**34.5**	37	**40**
HANDBI	14	36.5	**14.9**	>3600	>3600	12	**13**
HANDBD	14	34.1	**13.3**	>3600	>3600	13	13
I640	100	8.7	**6.1**	>3600	>3600	90	**91**
PUCNU	18	278.9	**80.2**	>3600	>3600	7	**11**
H	14	488.7	**477.4**	>3600	>3600	4	**5**

SJ - SCIP-Jack

L18 - Leitner et. al. 2018

Performance of SCIP-Jack on PCSTP (2)

- SCIP-Jack often more than two or three orders of magnitude faster than next best solver (from Fischetti et. al. 2017, Math. Prog. C). E.g. max. time on Cologne2:
  - Fischetti et. al.: >200 seconds
  - SCIP-Jack: <0.1 seconds
- Recent PCSTP improvements of SCIP-Jack allowed us to solve two and improve best known solutions for more than one third of unsolved DIMACS 2014 instances.

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:

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

- added new features to UG to allow branching on constraints
- ...but (in particular for PCSTP) we need to adapt underlying graph!

Building new subproblems

- SCIP-Jack/UG transfers branching history together with subproblem. SCIP-Jack changes underlying graph (e.g. deletes vertices).
- Improves finding locally valid solutions and helps cut generation. Local cuts also transferred by UG.
- Using branching history for separation and heuristics, we got speed-up of about 30% with 32 threads.

Aggressive presolving of subproblems

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

Problem:

- For Steiner tree problems MIP presolving remarkably unsuccessful.
- Complex Steiner reduction techniques not easy to reflect in IP. Thus Steiner tree solvers only perform reductions to delete vertices and edges during B&B.

We make use of following observation:

Aggressive presolving of subproblems(2)

Each SCIP-Jack Steiner tree reduction transforms SPG (V, E, T, c) to SPG (V', E', T', c') and provides function $p : E' \to \mathcal{P}(E)$ such that for each (optimal) solution $S' \subseteq E'$ to transformed problem, set $\bigcup_{e \in S'} p(e)$ is (optimal) solution to original problem.

Observation

Let (V, E, T, c), (V', E', T', c'), and p as above. Define
$E'' := \bigcup_{e \in E'} p(e)$,
$V'' := \{v \in V \mid \exists (v, w) \in E'', w \in V\}$,
$T'' := \{t \in T \mid \exists (t, w) \in E'', w \in V\}$,
$c'' := c|_{E''}$.
Each (optimal) solution to (V'', E'', T'', c'') is (optimal) solution to (V, E, T, c).

Aggressive presolving of subproblems(3)

Observation allows us to perform aggressive presolving whenever new subproblems are initialized. About 25% speed-up with 32 threads and improved scaling behaviour.

Racing ramp-up with customized racing paramaters (new feature of UG) is also used.

Shared memory results for PCSTP

Threads	i640-115	i640-141	cc10-2nu	cc7nu
1	201	809	812	4,333
8	102	314	311	1,922
16	80	210	200	1,023
32	78	110	167	843
64	79	101	165	723
root time	15	28	107	55
max # solvers	13	48	34	64
first max active time	55	152	211	301

All times in seconds

We performed experiments on PUC, the most difficult Steiner tree test set: 29 of 50 instances have remained unsolved.

SCIP-Jack (sequential) is currently the strongest solver on PUC:

- # instances solved in one hour:
  - 13 by SCIP-Jack:
  - 12 by best other (Polzin, Vahdati Daneshmand, 2014)[1]
- # instances solved in 12 hours:
  - 18 by SCIP-Jack:
  - 13 by best other (Polzin, Vahdati Daneshmand, 2014)

[2] Run time is scaled, as solver is not publicly available

Two years ago on the PUC test set ug[SCIP-Jack, MPI]:

- ▷ improved primal bounds for 14 instances
- ▷ solved three instances to optimality for first time

In recent computational experiments we were able to

- ▷ improve two best known primal bounds
- ▷ solve one previously unsolved instance

Table 1: Statistics for solving hc9p on supercomputers

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes	Open nodes
1	ISM	72	604,796 (317)	< 0.3	738	30,242.0000	29,879.3721	1.21	0	0
						30,242.0000	30,058.9366	0.61	110,012,624	1,257,112
2	ISM	2,304	604,794	< 1.5	979,695	30,242.0000	30,058.7930	0.61	0	15
						30,242.0000	30,102.7556	0.46	3,758,532,600	723,167
3	HLRN III	24,576	86,336	< 1.7	8,811,512	30,242.0000	30,102.6645	0.46	0	35
						30,242.0000	30,116.3592	0.42	2,402,406,311	575,678
4	HLRN III	12,288	43,199	< 1.5	1,709,027	30,242.0000	30,115.3331	0.42	0	3,709
						30,242.0000	30,120.4801	0.40	664,909,985	602,323
5	HLRN III	12,288	118,259	1.5	9,158,920	30,242.0000	30,120.4801	0.40	0	285
						30,242.0000	30,242.0000	0.00	1,677,724,126	0

Supercomputers used:

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

How open nodes and active solvers evolved (hc9p)

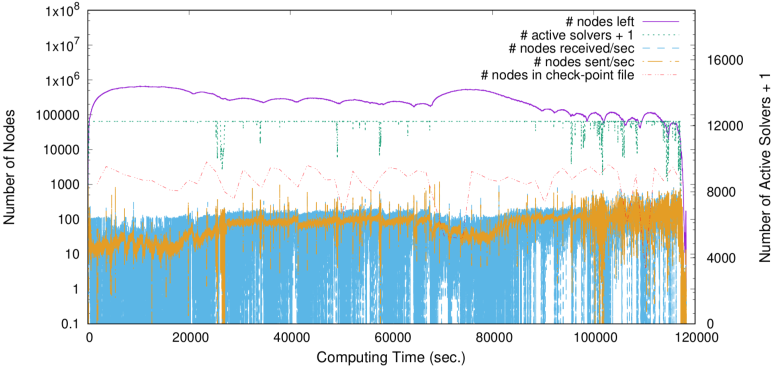


Figure 1: Evolution of computation for solving hc9p by using 12,288 cores (Run 5)

Table 2: Statistics for solving hc11p on supercomputers

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes	Open nodes
1.1	ISM	72	604,799 (2,558)	< 0.3	71	119,492.0000	117,388.8528	1.79	0	0
						119,297.0000	117,496.5470	1.53	4,314,198	1,109,629
1.2	HLRN III	12,288	43,149 (7,164)	< 0.5	31,304	119,297.0000	117,388.7971	1.63	0	0
						119,297.0000	117,426.2226	1.59	28,491,470	5,433,482
2	HLRN III	43,000	86,354	< 4.9	86,152	119,297.0000	117,426.2226	1.59	0	103
						119,297.0000	117,468.8459	1.56	267,513,609	40,499,188

Supercomputers used:

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

How open nodes and active solvers evolved (hc11p)

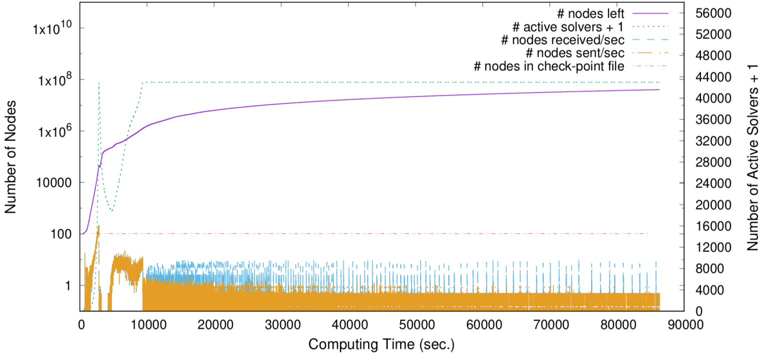


Figure 2: Evolution of computation for solving hc11p by using 43,000 cores (Run 2)

We plan to

- (considerably) improve sequential performance of both SPG and related problems
- add internal shared memory parallelizations to SCIP-Jack

And:

- solve more open PUC instances with ug[SCIP-Jack, MPI]
- solve open instances of related problems with ug[SCIP-Jack, MPI]

We plan to

- (considerably) improve sequential performance of both SPG and related problems
- add internal shared memory parallelizations to SCIP-Jack

And:

- solve more open PUC instances with ug[SCIP-Jack, MPI]
- solve open instances of related problems with ug[SCIP-Jack, MPI]

Thank you!

Some references

- Gamrath, Koch, Maher, Rehfeldt, Shinano: *SCIP-Jack – A solver for STP and variants with parallelization extentions*, Math. Prog. Comp. (2017)
- Rehfeldt, Koch: *Transformations for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem to SAP*, J. of Comp. Math. (2018)
- Rehfeldt, Koch: *Reduction-based exact solution of prize-collecting Steiner tree problems* ZR 18-55 (2018)
- Shinano, Rehfeldt, Koch: *Building Optimal Steiner Trees on Supercomputers by using up to 43,000 Cores* ZR 18-58 (2018)
- Rehfeldt, Koch, Maher, *Reduction Techniques for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem*, Networks (2019, in press)
- Rehfeldt, Koch: *Combining NP-Hard Reduction Techniques and Strong Heuristic in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem* SIAM J. Opt. (2019, in press)

SCIP Opt. Suite: `http://scip.zib.de`