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

Daniel Rehfeldt · Yuji Shinano · Thorsten Koch
Zuse Institute Berlin
Technische Universität Berlin
The Steiner Tree Problem in Graphs

Given:
1. $G = (V, E)$: undirected graph
2. $T \subseteq V$: subset of vertices
3. $c \in \mathbb{R}^E_>$: 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 $NP$-complete problems.
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 \( T \subseteq V(S) \)
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 $T \subseteq V(S)$

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)
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
- ...

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

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

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
- ...

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

E.g. *Solving Generalized Maximum-Weight Connected Subgraph Problems for Network Enrichment Analysis* (Loboda et al., 2016)
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
- ...
What we wanted: Solver for Steiner tree and many related problems
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 ...

![Diagram of Steiner tree and related problems]
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:

\[
\begin{align*}
\min \ c^T y \\
y(\delta^+(W)) & \geq 1 & \text{for all } W \subset V, r \in W, (V \setminus W) \cap T \neq \emptyset \\
y(\delta^-(v)) & \leq y(\delta^+(v)) & \text{for all } v \in V \setminus T \\
y(\delta^-(v)) & \geq y(a) & \text{for all } a \in \delta^+(v), v \in V \setminus T \\
y(a) & \in \{0, 1\} & \text{for all } a \in A
\end{align*}
\]
Some Facts about SCIP

- **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
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:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Problem Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPG</td>
<td>Steiner tree problem in graphs</td>
</tr>
<tr>
<td>SAP</td>
<td>Steiner arborescence problem</td>
</tr>
<tr>
<td>RSMT</td>
<td>Rectilinear Steiner minimum tree problem</td>
</tr>
<tr>
<td>OARSMT</td>
<td>Obstacle-avoiding rectilinear Steiner minimum tree problem</td>
</tr>
<tr>
<td>NWSTP</td>
<td>Node-weighted Steiner tree problem</td>
</tr>
<tr>
<td>NWPTSTP</td>
<td>Node-weighted partial terminal Steiner tree problem</td>
</tr>
<tr>
<td>PCSTP</td>
<td>Prize-collecting Steiner tree problem</td>
</tr>
<tr>
<td>RPCSTP</td>
<td>Rooted prize-collecting Steiner tree problem</td>
</tr>
<tr>
<td>MWCSP</td>
<td>Maximum-weight connected subgraph problem</td>
</tr>
<tr>
<td>RMWCSP</td>
<td>Rooted maximum-weight connected subgraph problem</td>
</tr>
<tr>
<td>DCSTP</td>
<td>Degree-constrained Steiner tree problem</td>
</tr>
<tr>
<td>GSTP</td>
<td>Group Steiner tree problem</td>
</tr>
<tr>
<td>HCDSTP</td>
<td>Hop-constrained directed Steiner tree problem</td>
</tr>
</tbody>
</table>
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 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...
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
Prize-Collecting Steiner Trees

Given:

- undirected graph \( G = (V, E) \)
- vertex costs \( p \in \mathbb{R}^V \geq 0 \)
- edge costs \( c \in \mathbb{R}^E \geq 0 \)

Prize-Collecting Steiner Tree Problem (PCSTP)

Find a tree \( S \subseteq G \) such that

\[
\sum_{e \in E(S)} c(e) + \sum_{v \in V \setminus V(S)} p(v) \text{ is minimized}
\]

\( T_p := \{ v \in V \mid p(v) > 0 \} \) 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 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 $W = (v_{i_1}, e_{i_1}, v_{i_2}, e_{i_2}, \ldots, e_{i_r}, v_{i_r})$ 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 $W$:

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

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:

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

**Definition**

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

\[
d_{pc}^- (v_i, v_j) := \min \{ I_{pc}^-(W') \mid W' \in W_{pc}(v_i, v_j) \}.
\]
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 \(t_p, t_q \in T_f\) \(T_f := \{t_1, t_2, \ldots, t_z\}\) 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 \(i \in \{1, \ldots, z\}\):
   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 \(T' := \{t_1', \ldots, t_z'\} \cup T_f\).

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 practical 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
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
<table>
<thead>
<tr>
<th>Test set</th>
<th>#</th>
<th>L18 [s]</th>
<th>SJ [s]</th>
<th>L18 [s]</th>
<th>SJ [s]</th>
<th># solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>34</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>34</td>
</tr>
<tr>
<td>Cologne1</td>
<td>14</td>
<td>0.0</td>
<td>0.0</td>
<td>0.1</td>
<td><strong>0.0</strong></td>
<td>14</td>
</tr>
<tr>
<td>Cologne2</td>
<td>15</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td><strong>0.1</strong></td>
<td>15</td>
</tr>
<tr>
<td>CRR</td>
<td>80</td>
<td>0.1</td>
<td>0.1</td>
<td>5.7</td>
<td><strong>1.1</strong></td>
<td>80</td>
</tr>
<tr>
<td>ACTMOD</td>
<td>8</td>
<td>0.9</td>
<td><strong>0.3</strong></td>
<td>3.5</td>
<td><strong>1.5</strong></td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>40</td>
<td>1.8</td>
<td><strong>0.2</strong></td>
<td>&gt;3600</td>
<td><strong>34.5</strong></td>
<td>37</td>
</tr>
<tr>
<td>HANDBI</td>
<td>14</td>
<td>36.5</td>
<td><strong>14.9</strong></td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>12</td>
</tr>
<tr>
<td>HANDBD</td>
<td>14</td>
<td>34.1</td>
<td><strong>13.3</strong></td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>13</td>
</tr>
<tr>
<td>I640</td>
<td>100</td>
<td>8.7</td>
<td><strong>6.1</strong></td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>90</td>
</tr>
<tr>
<td>PUCNU</td>
<td>18</td>
<td>278.9</td>
<td><strong>80.2</strong></td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>7</td>
</tr>
<tr>
<td>H</td>
<td>14</td>
<td>488.7</td>
<td><strong>477.4</strong></td>
<td>&gt;3600</td>
<td>&gt;3600</td>
<td>4</td>
</tr>
</tbody>
</table>

SJ - SCIP-Jack
L18 - Leitner et. al. 2018
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.
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!
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.
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:
Each SCIP-Jack Steiner tree reduction transforms SPG \((V, E, T, c)\) to SPG \((V', E', T', c')\) and provides function \(p : E' \rightarrow \mathcal{P}(E)\) 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 \((V, E, T, c)\).
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 parameters (new feature of UG) is also used.
### Shared memory results for PCSTP

<table>
<thead>
<tr>
<th>Threads</th>
<th>i640-115</th>
<th>i640-141</th>
<th>cc10-2nu</th>
<th>cc7nu</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>201</td>
<td>809</td>
<td>812</td>
<td>4,333</td>
</tr>
<tr>
<td>8</td>
<td>102</td>
<td>314</td>
<td>311</td>
<td>1,922</td>
</tr>
<tr>
<td>16</td>
<td>80</td>
<td>210</td>
<td>200</td>
<td>1,023</td>
</tr>
<tr>
<td>32</td>
<td>78</td>
<td>110</td>
<td>167</td>
<td>843</td>
</tr>
<tr>
<td>64</td>
<td>79</td>
<td>101</td>
<td>165</td>
<td>723</td>
</tr>
</tbody>
</table>

- **root time**: 15, 28, 107, 55
- **max # solvers**: 13, 48, 34, 64
- **first max active time**: 55, 152, 211, 301

All times in seconds
Results for SPG

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)

¹Run time is scaled, as solver is not publicly available.
Distributed memory results

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

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

Supercomputers used:

- **ISM**: HPE SGI 8600 with 384 compute nodes, each node has two Intel Xeon Gold 6154 3.0GHz CPUs (18 cores × 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)
Solving hc11p

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

Table 2: Statistics for solving \texttt{hc11p} on supercomputers

<table>
<thead>
<tr>
<th>Run</th>
<th>Computer</th>
<th>Cores</th>
<th>Time (sec.)</th>
<th>Idle (%)</th>
<th>Trans.</th>
<th>Primal bound (Upper bound)</th>
<th>Dual bound (Lower bound)</th>
<th>Gap (%)</th>
<th>Nodes</th>
<th>Open nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>ISM</td>
<td>72</td>
<td>604,799 (2,558)</td>
<td>&lt; 0.3</td>
<td>71</td>
<td>119,492.0000</td>
<td>117,388.8528</td>
<td>1.79</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>119,297.0000</td>
<td>117,496.5470</td>
<td>1.53</td>
<td>4,314,198</td>
<td>1,109,629</td>
</tr>
<tr>
<td>1.2</td>
<td>HLRN III</td>
<td>12,288</td>
<td>43,149 (7,164)</td>
<td>&lt; 0.5</td>
<td>31,304</td>
<td>119,297.0000</td>
<td>117,388.7971</td>
<td>1.63</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>119,297.0000</td>
<td>117,426.2226</td>
<td>1.59</td>
<td>28,491,470</td>
<td>5,433,482</td>
</tr>
<tr>
<td>2</td>
<td>HLRN III</td>
<td>43,000</td>
<td>86,354</td>
<td>&lt; 4.9</td>
<td>86,152</td>
<td>119,297.0000</td>
<td>117,426.2226</td>
<td>1.59</td>
<td>0</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>119,297.0000</td>
<td>117,468.8459</td>
<td>1.56</td>
<td>267,513,609</td>
<td>40,499,188</td>
</tr>
</tbody>
</table>
Figure 2: Evolution of computation for solving $\text{hc11p}$ 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!


Shinano, Rehfeldt, Koch: \textit{Building Optimal Steiner Trees on Supercomputers by using up to 43,000 Cores} ZR 18-58 (2018)

Rehfeldt, Koch, Maher, \textit{Reduction Techniques for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem}, Networks (2019, in press)
