Introducing Adaptive Algorithmic Behavior of Primal Heuristics in SCIP for Solving Mixed Integer Programs

Gregor Hendel, hendel@zib.de
joint work with Matthias Miltenberger and Jakob Witzig
Monash University, Melbourne, Australia, 22 March, 2019
A research institute and computing center of the State of Berlin with research units:

- Numerical Analysis and Modeling
- Visualization and Data Analysis
- Optimization:
  - Energy – Transportation – Health – Mathematical Optimization Methods
- Scientific Information Systems
- Computer Science and High Performance Computing
Meet the SCIP Team

26 active developers

• 4 running Bachelor and Master projects
• 14 running PhD projects
• 8 postdocs and professors

4 development centers in Germany

• ZIB: SCIP, SoPlex, UG, ZIMPL
• TU Darmstadt: SCIP and SCIP-SDP
• FAU Erlangen-Nürnberg: SCIP
• RWTH Aachen: GCG

Many international contributors and users

• more than 10 000 downloads per year from 100+ countries

Careers

• 10 awards for Masters and PhD theses: MOS, EURO, GOR, DMV
• 7 former developers are now building commercial optimization software at CPLEX, FICO Xpress, Gurobi, MOSEK, and GAMS
Overview

Introduction
  Large Neighborhood Search for MIP
  Multi-Armed Bandit Selection

SCIP’s Adaptive LNS
  Reward Function for LNS
  Computational Results

Diving & Adaptive Diving

Outlook: Adaptive LP Pricing
Introduction
Mixed Integer Programs

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax \geq b \\
& \quad \ell \leq x \leq u \\
& \quad x \in \{0, 1\}^{n_b} \times \mathbb{Z}_{n_i}^{n_b-n_b} \times \mathbb{Q}_i^{n-n_i}
\end{align*}
\]

Solution method:

• typically solved with branch-and-cut
• at each node, an LP relaxation is (re-)solved with the dual Simplex algorithm
• primal heuristics, e.g., Large Neighborhood Search and diving methods, support the solution process
Introduction

Large Neighborhood Search for MIP
**Auxiliary MIP**

Let $P$ be a MIP with solution set $\mathcal{F}_P$. For a polyhedron $\mathcal{N} \subseteq \mathbb{Q}^n$ and objective coefficients $c_{\text{aux}} \in \mathbb{Q}^n$, a MIP $P_{\text{aux}}$ defined as

$$\min \left\{ c_{\text{aux}}^T x \mid x \in \mathcal{F}_P \cap \mathcal{N} \right\}$$

is called an auxiliary MIP of $P$, and $\mathcal{N}$ is called neighborhood.

**Large Neighborhood Search (LNS)** heuristics solve auxiliary MIPs and can be distinguished by their respective neighborhoods.
Let $\mathcal{M} \subseteq \{1, \ldots, n_i\}$, $x^* \in \mathbb{Q}^n$.

- **Fixing neighborhood**

$$\mathcal{N}^{\text{fix}}(\mathcal{M}, x^*) := \{x \in \mathbb{Q}^n | x_j = x_j^* \ \forall \ j \in \mathcal{M}\}$$
Let $\mathcal{M} \subseteq \{1, \ldots, n_i\}$, $x^* \in \mathbb{Q}^n$.

- **Fixing neighborhood**

$$\mathcal{N}_{\text{fix}}(\mathcal{M}, x^*) := \{x \in \mathbb{Q}^n | x_j = x_j^* \ \forall j \in \mathcal{M}\}$$

- **Improvement neighborhood**

$$\mathcal{N}_{\text{obj}}(\delta, x^{\text{inc}}) := \{x \in \mathbb{Q}^n | c^T x \leq (1 - \delta) \cdot c^T x^{\text{inc}} + \delta \cdot c^{\text{dual}}\}$$
Examples of LNS Heuristics

Relaxation Induced Neighborhood Search (RINS) [Danna et al., 2005]

$$\mathcal{N}_{RINS} := \mathcal{N}^{fix} \left( \mathcal{M} = \left\{ x^{lp}, x^{inc} \right\}, x^{inc} \right) \cap \mathcal{N}^{obj} \left( \delta, x^{inc} \right).$$
Examples of LNS Heuristics

**Relaxation Induced Neighborhood Search (RINS)** [Danna et al., 2005]

\[ \mathcal{N}_{\text{RINS}} := \mathcal{N}^{\text{fix}} \left( \mathcal{M} = \left( \{ x^{\text{lp}}, x^{\text{inc}} \} \right), x^{\text{inc}} \right) \cap \mathcal{N}^{\text{obj}} \left( \delta, x^{\text{inc}} \right). \]

**Local Branching** [Fischetti and Lodi, 2003]

\[ \mathcal{N}_{\text{LBranch}} := \left\{ x \in \mathbb{Q}^n \mid \left\| x - x^{\text{inc}} \right\|_b \leq d_{\text{max}} \right\} \cap \mathcal{N}^{\text{obj}} \left( \delta, x^{\text{inc}} \right). \]
Famous LNS Heuristics

- Relaxation Induced Neighborhood Search (RINS) [Danna et al., 2005]
- Local Branching [Fischetti and Lodi, 2003]
Famous LNS Heuristics

- Relaxation Induced Neighborhood Search (RINS) [Danna et al., 2005]
- Local Branching [Fischetti and Lodi, 2003]
- Crossover, Mutation [Rothberg, 2007]
- RENS [Berthold, 2014]
- Proximity [Fischetti and Monaci, 2014]
- DINS [Ghosh, 2007]
- Zeroobjective [in SCIP, Gurobi, XPress,...]
- Analytic Center Search [Berthold et al., 2017]
- ...
Introduction

Multi-Armed Bandit Selection
The Multi-Armed Bandit Problem

- Discrete time steps $t = 1, 2, \ldots$
- Finite set of actions $\mathcal{H}$

1. Choose $h_t \in \mathcal{H}$
2. Observe reward $r(h_t, t) \in [0, 1]$
3. Goal: Maximize $\sum_t r(h_t, t)$
The Multi-Armed Bandit Problem

- Discrete time steps $t = 1, 2, \ldots$
- Finite set of actions $\mathcal{H}$

1. Choose $h_t \in \mathcal{H}$
2. Observe reward $r(h_t, t) \in [0, 1]$
3. Goal: Maximize $\sum_t r(h_t, t)$

Two main scenarios:

- **stochastic** i.i.d. rewards for each action over time
The Multi-Armed Bandit Problem

Discrete time steps \( t = 1, 2, \ldots \)
Finite set of actions \( \mathcal{H} \)

1. Choose \( h_t \in \mathcal{H} \)
2. Observe reward \( r(h_t, t) \in [0, 1] \)
3. Goal: Maximize \( \sum_t r(h_t, t) \)

Two main scenarios:

- **stochastic** i.i.d. rewards for each action over time
- **adversarial** an opponent tries to maximize the player’s regret.

Literature: [Bubeck and Cesa-Bianchi, 2012]
Let $T_h(t) = \sum_{t' \leq t} \mathbb{1}_{h=h_t}$ and $\bar{r}_h(t) = \frac{1}{T_h(t)} \sum_{t' \leq t} r_{h,t} \mathbb{1}_{h=h_t}$.

$\varepsilon$-greedy
Select heuristic at random with probability $\varepsilon_t = \varepsilon \sqrt{\frac{|\mathcal{H}|}{t}}$, otherwise use best.
Let $T_h(t) = \sum_{t' \leq t} \mathbf{1}_{h=ht}$ and 
\[ \bar{r}_h(t) = \frac{1}{T_h(t)} \sum_{t' \leq t} r_{h,t} \mathbf{1}_{h=ht} \]

$\varepsilon$-greedy
Select heuristic at random with probability $\varepsilon_t = \varepsilon \sqrt{\frac{|\mathcal{H}|}{t}}$, otherwise use best.

Upper Confidence Bound (UCB)
\[ h_t \in \begin{cases} 
\arg\max_{h \in \mathcal{H}} \left\{ \bar{r}_h(t-1) + \sqrt{\frac{\alpha \ln(1+t)}{T_h(t-1)}} \right\} & \text{if } t > |\mathcal{H}|, \\
\{H_t\} & \text{if } t \leq |\mathcal{H}|.
\end{cases} \]
Let $T_h(t) = \sum_{t' \leq t} 1_{h=h_{t'}}$ and $\bar{r}_h(t) = \frac{1}{T_h(t)} \sum_{t' \leq t} r_{h,t} 1_{h=h_{t'}}$

$\varepsilon$-greedy
Select heuristic at random with probability $\varepsilon_t = \varepsilon \sqrt{\frac{|H|}{t}}$, otherwise use best.

Upper Confidence Bound (UCB)
$$h_t \in \left\{ \begin{array}{ll}
\arg\max_{h \in H} \left\{ \bar{r}_h(t-1) + \sqrt{\frac{\alpha \ln(1+t)}{T_h(t-1)}} \right\} & \text{if } t > |H|, \\
\{H_t\} & \text{if } t \leq |H|.
\end{array} \right.$$ 

Exp.3
$$p_{h,t} = (1 - \gamma) \cdot \frac{\exp(w_{h,t})}{\sum_{h'} \exp(w_{h',t})} + \gamma \cdot \frac{1}{|H|}$$
Bandit Algorithms

Let $T_h(t) = \sum_{t' \leq t} 1_{h=t'}$ and $\bar{T}_h(t) = \frac{1}{T_h(t)} \sum_{t' \leq t} r_{h,t} 1_{h=t'}$

$\varepsilon$-greedy
Select heuristic at random with probability $\varepsilon_t = \varepsilon \sqrt{\frac{|\mathcal{H}|}{t}}$, otherwise use best.

Upper Confidence Bound (UCB)

$h_t \in \begin{cases} \arg\max_{h \in \mathcal{H}} \left\{ \bar{T}_h(t - 1) + \sqrt{\frac{\alpha \ln(t+1)}{T_h(t-1)}} \right\} & \text{if } t > |\mathcal{H}|, \\ \{H_t\} & \text{if } t \leq |\mathcal{H}|. \end{cases}$

Exp.3

$p_{h,t} = (1 - \gamma) \cdot \frac{\exp(w_{h,t})}{\sum_{h'} \exp(w_{h',t})} + \gamma \cdot \frac{1}{|\mathcal{H}|}$

Individual parameters $\alpha, \varepsilon, \gamma \geq 0$ can be calibrated to the problem at hand.
SCIP’s Adaptive LNS
Adaptive Large Neighborhood Search

- new primal heuristic plugin `heur_alns.c`
- controls 8 neighborhoods
- neighborhoods are bandit-selected based on their reward
- further algorithmic steps: generic fixings, adaptive fixing rate
- released with SCIP 5.0, improved in SCIP 6.0
SCIP’s Adaptive LNS

Reward Function for LNS
Rewarding Neighborhoods

Goal A suitable reward function $r_{\text{alns}}(h_t, t) \in [0, 1]$
Rewarding Neighborhoods

Goal A suitable reward function \( r_{\text{alns}}(h_t, t) \in [0, 1] \)

Solution Reward

\[
r_{\text{sol}}(h_t, t) = \begin{cases} 
1 & \text{if } x^{\text{old}} \neq x^{\text{new}} \\
0 & \text{else}
\end{cases}
\]
**Rewarding Neighborhoods**

**Goal** A suitable reward function $r_{\text{alns}}(h_t, t) \in [0, 1]$

**Solution Reward**

$$r_{\text{sol}}(h_t, t) = \begin{cases} 
1 & \text{if } x_{\text{old}} \neq x_{\text{new}} \\
0 & \text{else}
\end{cases}$$

**Gap Reward**

$$r_{\text{gap}}(h_t, t) = \frac{c^T x_{\text{old}} - c^T x_{\text{new}}}{c^T x_{\text{old}} - c_{\text{dual}}}$$
Goal: A suitable reward function $r^{\text{alns}}(h_t, t) \in [0, 1]$

Solution Reward

$$r^{\text{sol}}(h_t, t) = \begin{cases} 1, & \text{if } x^{\text{old}} \neq x^{\text{new}} \\ 0, & \text{else} \end{cases}$$

Gap Reward

$$r^{\text{gap}}(h_t, t) = \frac{c^T x^{\text{old}} - c^T x^{\text{new}}}{c^T x^{\text{old}} - c^{\text{dual}}}$$

Failure Penalty

$$r^{\text{fail}}(h_t, t) = \begin{cases} 1, & \text{if } x^{\text{old}} \neq x^{\text{new}} \\ 1 - \phi(h_t, t) \frac{n(h_t)}{n^{\text{lim}}} \end{cases}$$
Goal A suitable reward function \( r^{\text{alns}}(h_t, t) \in [0, 1] \)

Solution Reward
\[
r^{\text{sol}}(h_t, t) = \begin{cases} 
1, & \text{if } x^{\text{old}} \neq x^{\text{new}} \\ 
0, & \text{else} 
\end{cases}
\]

Gap Reward
\[
r^{\text{gap}}(h_t, t) = \frac{c^T x^{\text{old}} - c^T x^{\text{new}}}{c^T x^{\text{old}} - c^{\text{dual}}}
\]

Failure Penalty
\[
r^{\text{fail}}(h_t, t) = \begin{cases} 
1, & \text{if } x^{\text{old}} \neq x^{\text{new}} \\ 
1 - \phi(h_t, t) \frac{n(h_t)}{n^{\text{lim}}} & \text{else} 
\end{cases}
\]

Default settings in ALNS: \( \eta_1 = 0.8, \eta_2 = 0.5 \)
SCIP’s Adaptive LNS

Computational Results
• **Always** execute all 8 neighborhoods with ALNS (disable old LNS heuristics)
• Disable solution transfer
• Record each reward
• Fixing rates 0.1 – 0.9

**Test Set**
666 instances from the test sets MIPLIB3, MIPLIB2003, MIPLIB2010, Cor@l, 5h time limit.
Simulate 100 repetitions of UCB, Exp.3, and $\epsilon$-greedy on the data

\[ h_t \in \begin{cases} \arg \max_{h \in \mathcal{H}} \left\{ \tilde{r}_h(t-1) + \sqrt{\frac{\alpha \ln(1+t)}{T_h(t-1)}} \right\} & \text{if } t > |\mathcal{H}|, \\ \{H_t\} & \text{if } t \leq |\mathcal{H}|. \end{cases} \]
Learning Curve of UCB

\[ h_t \in \begin{cases} \arg \max_{h \in \mathcal{H}} \left\{ \bar{r}_h(t-1) + \sqrt{\frac{\alpha \ln(1+t)}{\bar{r}_h(t-1)}} \right\} & \text{if } t > |\mathcal{H}|, \\
\{H_t\} & \text{if } t \leq |\mathcal{H}|. \end{cases} \]  

(UCB)

Gregor Hendel – SCIP’s Adaptive Primal Heuristics
More Learning Curves

Exp. 3
- gamma_0.15
- gamma_0.45
- gamma_0.95
- gamma_0.07041455
- avg

Eps-greedy
- eps_0.5
- eps_0.4685844
- avg

Gregor Hendel – SCIP's Adaptive Primal Heuristics
Performance of the ALNS Framework

![Chart showing the performance of the ALNS Framework for different instance groups and settings. The chart includes bars for each instance group (all, Diff, Eq, MIPLIB2010) and settings (ALNS off, Eps-greedy, Exp.3, UCB). The bars indicate the relative time percentage, with values ranging from -12 to 0. Various categories within each instance group are represented with different colors.](chart.png)
Diving & Adaptive Diving
9 different diving heuristics explore an auxiliary tree in probing mode.

Diving Heuristics in SCIP \cite{Achterberg2007}

- coefficient diving
- fractionality diving
- guided diving \cite{Danna2005}
- pseudo costs
- ...

Information from Diving:

- Primal solutions
- Variable branching history (pseudo costs, ...)
- Conflict clauses
Goal of Selection
Improving both primal solutions and relevant search information

Problem: Solutions are only rarely found by diving heuristics, see also [Khalil et al., 2017].

Possible reward measures that discriminate better:

- minimum avg. depth
- minimum backtracks/conflict ratio
- minimum avg. probing nodes
- minimum avg. LP iterations

Unlikely that there is a unique best diving algorithm ⇒ use weighted sampling method with inverse probabilities as in LP pricing.
### Computational Results

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>Setting</th>
<th>Solved</th>
<th>Time</th>
<th>rel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>1477</td>
<td>default</td>
<td>1005</td>
<td>152.54</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adaptive diving</td>
<td>1020</td>
<td>146.05</td>
<td>0.957</td>
</tr>
<tr>
<td>≥ 100 sec.</td>
<td>396</td>
<td>default</td>
<td>363</td>
<td>485.39</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adaptive diving</td>
<td>378</td>
<td>436.99</td>
<td>0.900</td>
</tr>
</tbody>
</table>

**Setup:**

- adaptive diving selects from 9 diving heuristics. It is called in addition to the SCIP diving heuristics.
- Test set: 496 instances from MIPLIBs & Cor@l benchmark sets 1h time limit, default + 2 LP Seeds, 48 node cluster with 16 Intel Xeon Gold 5122 @ 3.60GHz, 96GB, Ubuntu 16.04
- Instance,seed pairs are treated as individual observations.
Adaptive Diving will be one of the main new features in SCIP 7.0. It

- automatically incorporates user diving heuristics\(^1\), if the user sets the visibility to *public*.
- Selects via weighted sampling based on conflicts/backtrack, or simply revolves through the available diving strategies.
- Provides new score types, including number of found solutions.
- learns, by default, from its own calls, but also from individual runs of the heuristics.

\(^1\)see [SCIP Docu](#) for information how to write diving heuristics.
Outlook: Adaptive LP Pricing
SCIP features the parameter `lp/pricing = ...`

<table>
<thead>
<tr>
<th>Problem</th>
<th>Steepest Edge</th>
<th>Devex</th>
<th>Quick Start Steep</th>
</tr>
</thead>
<tbody>
<tr>
<td>neos-1601936</td>
<td>1098.50</td>
<td>2126.55</td>
<td>1502.57</td>
</tr>
<tr>
<td>nw04</td>
<td>46.90</td>
<td>21.34</td>
<td>31.08</td>
</tr>
<tr>
<td>pigeon-12</td>
<td>3600.00</td>
<td>3600.00</td>
<td>3.02</td>
</tr>
</tbody>
</table>

Automatic selection strategy within SoPlex: run devex for 10000 iterations, then switch to steepest edge.
**LP Pricing Selection**

SCIP features the parameter `lp/pricing = ...`

<table>
<thead>
<tr>
<th></th>
<th>s(teepest edge)</th>
<th>d(evex)</th>
<th>q(uick start steep)</th>
</tr>
</thead>
<tbody>
<tr>
<td>neos-1601936</td>
<td>1098.50</td>
<td>2126.55</td>
<td>1502.57</td>
</tr>
<tr>
<td>nw04</td>
<td>46.90</td>
<td>21.34</td>
<td>31.08</td>
</tr>
<tr>
<td>pigeon-12</td>
<td>3600.00</td>
<td>3600.00</td>
<td>3.02</td>
</tr>
</tbody>
</table>

[Forrest and Goldfarb, 1992] [Harris, 1973]

Automatic selection strategy within SoPlex: run devex for 10000 iterations, then switch to steepest edge.

**Goal:** Maximize LP throughput
Maximize LP throughput \( \Leftrightarrow \) discover and select the LP pricing with minimum expected running time \( \tau^*_p, p \in \{\text{devex, steep, qsteep}\} \)

Problem for UCB: Need \([0, 1]\) score to maximize
Maximize LP throughput $\Leftrightarrow$ discover and select the LP pricing with minimum expected running time $\tau_p^*$, $p \in \{\text{devex, steep, qsteep}\}$

Problem for UCB: Need $[0, 1]$ score to maximize

Solution: Scale the (normalized) reward

- Let $\tau_{t,p}$ be the measured running time for pricer $p$ at step $t$
- Use reward $r_{t,p} = \frac{1}{1 + \frac{\tau_{t,p}}{\tau_p}}$ for UCB
Maximize LP throughput $\iff$ discover and select the LP pricing with minimum expected running time $\tau_p^*$, $p \in \{\text{devex, steep, qsteep}\}$

Problem for UCB: Need $[0, 1]$ score to maximize

Solution: Scale the (normalized) reward

- Let $\tau_{t,p}$ be the measured running time for pricer $p$ at step $t$
- Use reward $r_{t,p} = \frac{1}{1 + \frac{\tau_{t,p}}{\tau_p}}$ for UCB

1st alternative: UCB variant (shifted greedy) (thanks to Tobias Achterberg)

- select a favorite pricer, w.l.o.g. $p_1$
- use shift vector $\sigma \in \mathbb{R}_+^P \sigma_{p_1} = 100$, $\sigma_p = 50$ for $p \neq p_1$
- always start with $p_1$ for a couple of resolves
- only start selection process if average iterations of $p_1$ exceed a threshold, e.g., 20.
- always select the pricer that minimizes

$$\bar{\tau}^\sigma_p = \frac{\sum_t \mathbb{1}_{p_t=p} \tau_{t,p}}{T_p(t - 1) + \sigma_p}$$
**LP Pricing Goal and Setup**

Maximize LP throughput \( \Leftrightarrow \) discover and select the LP pricing with minimum expected running time \( \tau^*_p, p \in \{\text{devex, steep, qsteep}\} \)

Problem for **UCB**: Need \([0, 1]\) score to maximize

Solution: Scale the (normalized) reward

- Let \( \tau_{t,p} \) be the measured running time for pricer \( p \) at step \( t \)
- Use reward \( r_{t,p} = \frac{1}{1 + \frac{\tau_{t,p}}{\tau_p}} \) for UCB

**1st alternative**: UCB variant (shifted greedy) (thanks to Tobias Achterberg)

- select a favorite pricer, w.l.o.g. \( p_1 \)
- use shift vector \( \sigma \in \mathbb{R}_+ P, \sigma_{p_1} = 100, \sigma_p = 50 \) for \( p \neq p_1 \)
- always start with \( p_1 \) for a couple of resolves
- only start selection process if average iterations of \( p_1 \) exceed a threshold, e.g., 20.
- always select the pricer that minimizes

\[
\bar{\tau}_p^\sigma = \frac{\sum_t 1_{p_t=p} \tau_{t,p}}{T_p (t - 1) + \sigma_p}
\]

**2nd alternative**: Turn shifted greedy weights into weighted sampling weights

- compute shifted version of average as in shifted greedy
- sample from weight distribution \( w_{p,t} \propto \frac{1}{\bar{\tau}_p^\sigma + 10^{-4}} \)
## Computational Results

### LP Solver **Cplex 12.7.1**

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>Pricing</th>
<th>solved</th>
<th>LP throughput</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>abs.</td>
<td>rel.</td>
</tr>
<tr>
<td>all</td>
<td>593</td>
<td>devex</td>
<td>288</td>
<td>72.4</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>qsteep</td>
<td>289</td>
<td>74.7</td>
<td>1.032</td>
</tr>
<tr>
<td></td>
<td></td>
<td>steep</td>
<td>288</td>
<td>76.4</td>
<td>1.056</td>
</tr>
<tr>
<td></td>
<td></td>
<td>weighted</td>
<td>289</td>
<td>73.0</td>
<td>1.009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UCB</td>
<td>292</td>
<td>79.6</td>
<td>1.100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sh. greedy</td>
<td>292</td>
<td>80.8</td>
<td>1.117</td>
</tr>
</tbody>
</table>

### LP Solver **SoPlex 3.1.1**

<table>
<thead>
<tr>
<th>Group</th>
<th>#</th>
<th>Pricing</th>
<th>solved</th>
<th>LP throughput</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>abs.</td>
<td>rel.</td>
</tr>
<tr>
<td>all</td>
<td>587</td>
<td>devex</td>
<td>279</td>
<td>44.2</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>qsteep</td>
<td>272</td>
<td>35.0</td>
<td>0.793</td>
</tr>
<tr>
<td></td>
<td></td>
<td>steep</td>
<td>280</td>
<td>37.7</td>
<td>0.854</td>
</tr>
<tr>
<td></td>
<td></td>
<td>weighted</td>
<td>282</td>
<td>42.7</td>
<td>0.966</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UCB</td>
<td>284</td>
<td>45.5</td>
<td>1.031</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sh. greedy</td>
<td>288</td>
<td>50.5</td>
<td>1.144</td>
</tr>
</tbody>
</table>

Test set: 150 instances from a total of 666 (MIPLIBs & Cor@l), time limit, default + 3 LP Seeds, 48 node cluster with 16 Intel Xeon Gold 5122 @ 3.60GHz, 96GB, Ubuntu 16.04
Conclusion

- bandit selection variants for LP pricing selection, diving heuristics, and Large Neighborhood Search heuristics
- different scenarios require different reward functions and selection strategies
- adaptive selection yields computational benefits in all three cases.

In the future, we would like to

- finalize the LP pricing prototype
  - switch to deterministic LP time measurement
  - calibrate bandit parameters
  - exploit seemingly lognormal distribution of LP solving time for simulation and different bandit algorithm (Thompson sampling)
- investigate the usefulness of keeping learned information for future solves.

LP counts in diving, probing, and normal lp mode for timtab1.
Further Literature


Thank you for your attention!

Visit scip.zib.de.


**Local branching.**
*Mathematical Programming*, 98(1-3):23–47.

**Proximity search for 0-1 mixed-integer convex programming.**
Technical report, DEI - Università di Padova.

**Steepest-edge simplex algorithms for linear programming.**

**DINS, a MIP Improvement Heuristic.**
Harris, P. M. J. (1973). 
**Pivot selection methods of the devex lp code.**

**Learning to run heuristics in tree search.**

**An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions.**