Learning in Discrete Optimization

Learning to Branch
AAAi 2016 with Pierre Le Bodic, Le Song, George Nemhauser, Bistra Dilkina

- Branching rules typically hand-designed (pseudocost PC), or too slow (strong branching SB)
- PC: a single ranking metric that may not capture all information

Goal: Learn a data-driven branching rule (fast + effective)

On-the-fly, Instance-specific learning within Branch&Bound to rank variables

- MIP instance
- Parameters
- Nodes
- until termination

Data Collection

Machine Learning

Oracle: Success Prediction

New Instance

1. Add oracle to predictions
2. Oracle predicts with predictions
3. Use oracle predictions
4. Decision: Run / Don’t run Run-When-Successful

Proposed Method

Offline learning across instances to predict whether a heuristic will find incumbent given node features

Experimental Results

<table>
<thead>
<tr>
<th>GISP – Num. Instances = 120</th>
<th>DEF</th>
<th>ML</th>
<th>ML/DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primal integral</td>
<td>2,384.30</td>
<td>1,012.90</td>
<td>0.42</td>
</tr>
<tr>
<td>Time to first incumbent</td>
<td>0.18</td>
<td>0.19</td>
<td>1.02</td>
</tr>
<tr>
<td>Time to best incumbent</td>
<td>5,365.93</td>
<td>2,219.09</td>
<td>0.41</td>
</tr>
<tr>
<td>Total calls (ML hours.)</td>
<td>147.38</td>
<td>90.14</td>
<td>0.61</td>
</tr>
<tr>
<td>Total time (ML hours.)</td>
<td>371.44</td>
<td>272.12</td>
<td>1.96</td>
</tr>
<tr>
<td>Success Rate (ML hours.)</td>
<td>0.01473</td>
<td>0.02697</td>
<td>1.83</td>
</tr>
<tr>
<td>Num. Instances Solved (% Gap)</td>
<td>0 (219.11)</td>
<td>0 (180.45)</td>
<td>N/A</td>
</tr>
<tr>
<td>Total time (BinB)</td>
<td>7,200.00</td>
<td>7,200.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>


Learning to Run Primal Heuristics
UCAi 2017 with Bistra Dilkina, George Nemhauser, Shabbir Ahmed, Yufeng Shao

- Solver runs heuristic every k nodes (e.g. k=1,5,10), a fixed param.
- Some nodes may have features that make them more promising for a heuristic, i.e. more likely to find incumbent if run at node

Goal: Learn to run heuristic at promising nodes

Learning Greedy Heuristics on Graphs
arXiv 2017 with Hanjun Dai, Yuyu Zhang, Bistra Dilkina, Le Song

- Example of Min Vertex Cover on social graph
- MVCApprox-Greedy: add nodes of edge with max degree sum; captures 1-hop effects
- In real-world applications, similar instances of the same problem are solved repeatedly
- Heuristic algorithms do not exploit the distribution of instances

Goal: Learn powerful heuristics from similar instances

On-the-fly learning of branching strategies
data collection: run SB and use its scores to rank variables at each node; compute variable features
model learning: learn SVM ranking model that ranks variables in dataset s.t. good ones are ranked better than bad ones
ML-based branching: use learned model to rank variables based on their features

CPLEX - MIPLIB2010 Benchmark (10 seeds)

Example of Min Vertex Cover on social graph

MVC Approx

Greedy action

Reinforcement Learning

State-action value function

4. Train

3. Q-function

= βₜσ(θ₂ ∑ j∈γₜ μ_j) + βₜHₜ)

aggregated embedding

individual embedding

MVC Barabasi-Albert

Each point is a vertex cover instance (100)