Exact Combinatorial Optimization
with Graph Convolutional Neural Networks

Laurent Charlin
with Maxime Gasse, Didier Chételat, Nicola Ferroni, Andrea Lodi

February 24, 2021
(slide deck adapted from M. Gasse)
Problem Setting

- **Goal:** Solve combinatorial optimization
  - Solvers provide exact solutions
- **Use case:**
  - Regularly solve problems from the same family (e.g. TSP)
    - Learn offline from a collected dataset (train)
    - Help solve new problems from the same family (test)
- Machine learning as a heuristic inside the CO solver
- Focus on the problem of branching
Overview

Brief introduction to Branching

Machine learning modelling
  MDP
  GCNN

Empirical Study
Brief introduction to Branching
Brief introduction to Branching

Mixed-Integer Linear Program (MILP)

\[
\begin{align*}
\arg \min_{x} & \quad c^T x \\
\text{subject to} & \quad Ax \leq b, \\
& \quad l \leq x \leq u, \\
& \quad x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}.
\end{align*}
\]

- \( c \in \mathbb{R}^n \) the objective coefficients
- \( A \in \mathbb{R}^{m \times n} \) the constraint coefficient matrix
- \( b \in \mathbb{R}^m \) the constraint right-hand-sides
- \( l, u \in \mathbb{R}^n \) the lower and upper variable bounds
- \( p \leq n \) integer variables

NP-hard problem.
Linear Program (LP) relaxation

\[
\begin{align*}
\arg \min \quad & \quad c^\top x \\
\text{subject to} \quad & \quad Ax \leq b, \\
& \quad l \leq x \leq u, \\
& \quad x \in \mathbb{R}^n.
\end{align*}
\]

Convex problem, efficient algorithms (e.g., simplex).

- \( x^* \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \) (lucky) \( \rightarrow \) solution to the original MILP
- \( x^* \notin \mathbb{Z}^p \times \mathbb{R}^{n-p} \) \( \rightarrow \) lower bound to the original MILP
Brief introduction to Branching

**Branch-and-Bound**

Split the LP recursively over a non-integral variable, i.e. \( \exists i \leq p \mid x_i^* \notin \mathbb{Z} \)

\[
\begin{align*}
x_i \leq \lfloor x_i^* \rfloor & \quad \lor \quad x_i \geq \lceil x_i^* \rceil.
\end{align*}
\]

**Lower bound** \((L)\): minimal among leaf nodes.

**Upper bound** \((U)\): minimal among leaf nodes with integral solution.

Stopping criterion:

- \( L = U \) (optimality certificate)
- \( L = \infty \) (infeasibility certificate)
- \( L - U < \) threshold (early stopping)
Brief introduction to Branching

Branch-and-Bound
Brief introduction to Branching

Branch-and-Bound: a sequential process

Sequential decisions:
- variable selection (branching)
- node selection
- cutting plane selection
- primal heuristic selection
- simplex initialization
- ...

State-of-the-art in B&B solvers: expert rules

Objective: no clear consensus
- \( L = U \) fast?
- \( U - L \) fast?
- \( L \) fast?
- \( U \) fast?
Expert branching rules: state-of-the-art

Strong branching (SB): one-step forward looking (greedy)
- solve both LPs for each candidate variable
- pick the variable resulting in tightest relaxation
  + small trees
  - computationally expensive

Pseudo-cost: backward looking
- keep track of tightenings in past branchings
- pick the most promising variable
  + very fast, almost no computations
  - cold start

Reliability pseudo-cost: best of both worlds
- compute SB scores at the beginning
- gradually switches to pseudo-cost (+ other heuristics)
  + best overall solving time trade-off (on MIPLIB)
Machine learning modelling
Objective: take actions which maximize the long-term reward

$$\sum_{t=0}^{\infty} \gamma^t r(s_t),$$

with $r : S \rightarrow \mathbb{R}$ a reward function and discount factor $\gamma$
Branching as a Markov Decision Process

State: the whole internal state of the solver, $s$.
Action: a branching variable, $a \in \{1, \ldots, p\}$.

Trajectory: $\tau = (s_0, \ldots, s_T)$

- initial state $s_0$: a MILP $\sim p(s_0)$;
- terminal state $s_T$: the MILP is solved;
- intermediate states: branching

$$s_{t+1} \sim p_\pi(s_{t+1}|s_t) = \sum_{a \in A} \pi(a|s_t) \cdot p(s_{t+1}|s_t, a).$$

Branching problem: solve

$$\pi^* = \arg\max_\pi \mathbb{E}_{\tau \sim p_\pi} [r(\tau)],$$

with $r(\tau) = \sum_{s \in \tau} r(s)$. 

Machine learning modelling
Challenges

MDP \Rightarrow Reinforcement learning (RL) ?

State representation: s
- global level: original MILP, tree, bounds, focused node...
- node level: variable bounds, LP solution, simplex statistics...
  - dynamically growing structure (tree)
  - variable-size instances (cols, rows) \Rightarrow Graph Neural Network

Sampling trajectories: \tau \sim p_\pi
- collect one \tau = solving a MILP (with \pi likely not optimal)
  - expensive \Rightarrow train on small instances

Reward function: r
- no consensus
  + a strong expert exists \Rightarrow imitation learning
Machine learning approaches (Pre-2020)

Node selection
- He et al., 2014
- Song et al., 2018

Variable selection (branching)
- Khalil, Le Bodic, et al., 2016 \implies \text{"online" imitation learning}
- Hansknecht et al., 2018 \implies \text{offline imitation learning}
- Balcan et al., 2018 \implies \text{theoretical results}

Cut selection
- Baltean-Lugojan et al., 2018
- Tang et al., 2019

Primal heuristic selection
- Khalil, Dilkina, et al., 2017
- Hendel et al., 2018
State encoding

Natural representation: variable / constraint bipartite graph

$$\arg \min_x c^\top x$$
subject to $$Ax \leq b,$$
$$l \leq x \leq u,$$
$$x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}.$$ 

- $$\mathcal{V}_i$$: variable features (type, coef. $$c$$, bounds, LP solution...)
- $$\mathcal{C}_j$$: constraint features (right-hand-side, LP slack...)
- $$\mathcal{E}_{i,j}$$: non-zero coefficients in $$A$$

Graph structure is fixed through time.

---

D. Selsam et al. (2019). Learning a SAT Solver from Single-Bit Supervision.
Branching Policy as a GCNN Model

Neighbourhood-based updates: \( v_i \leftarrow \sum_{j \in N_i} f_{\theta}(v_i, e_{i,j}, c_j) \)

\[
\begin{align*}
0.2 & \quad \leftarrow \quad v_0 \\
0.1 & \quad \leftarrow \quad v_1 \\
0.7 & \quad \leftarrow \quad v_2 \\
\end{align*}
\]

\( \pi(a \mid s) \)

\( s \)

Natural model choice for graph-structured data

- permutation-invariance
- benefits from sparsity

---

T. N. Kipf et al. (2016). Semi-Supervised Classification with Graph Convolutional Networks.
Implementation Details

Interleaved half convolutions

initial embedding \( \rightarrow \) \( C \)-side convolution \( \rightarrow \) \( V \)-side convolution \( \rightarrow \) final embedding + softmax

<table>
<thead>
<tr>
<th>Input</th>
<th>Output 1</th>
<th>Output 2</th>
<th>Final Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V )</td>
<td>( V^1 )</td>
<td>( V^2 )</td>
<td>( \pi(x) )</td>
</tr>
<tr>
<td>( n \times d )</td>
<td>( n \times 64 )</td>
<td>( n \times 64 )</td>
<td>( n \times 1 )</td>
</tr>
<tr>
<td>( E )</td>
<td>( E )</td>
<td>( E )</td>
<td>( E )</td>
</tr>
<tr>
<td>( m \times n \times e )</td>
<td>( m \times n \times e )</td>
<td>( m \times n \times e )</td>
<td>( m \times n \times e )</td>
</tr>
<tr>
<td>( C )</td>
<td>( C^1 )</td>
<td>( C^2 )</td>
<td>( C^2 )</td>
</tr>
<tr>
<td>( m \times c )</td>
<td>( m \times 64 )</td>
<td>( m \times 64 )</td>
<td>( m \times 64 )</td>
</tr>
</tbody>
</table>

Prenorm-layer

- standardize conv. operations
Empirical Study
Imitation learning

Full Strong Branching (FSB): good branching rule, but expensive. Can we learn a fast, good-enough approximation?

Behavioural cloning

- collect $D = \{(s, a^\star), \ldots\}$ from the expert agent (FSB)
- estimate $\pi^*(a | s)$ from $D$

+ no reward function, supervised learning, well-behaved
  - will never surpass the expert...

Implementation with the open-source solver SCIP

Not a new idea

- Alvarez et al., 2017 predict SB scores, XTrees model
- Khalil, Le Bodic, et al., 2016 predict SB rankings, SVMrank model
- Hansknecht et al., 2018 do the same, $\lambda$-MART model

---

1A. Gleixner et al. (July 2018). The SCIP Optimization Suite 6.
Empirical Study

Minimum set covering\(^2\)

- Comparing the accuracy of machine learning models

<table>
<thead>
<tr>
<th>Model</th>
<th>acc@1</th>
<th>acc@5</th>
<th>acc@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTrees</td>
<td>51.8 ± 0.3</td>
<td>80.5 ± 0.1</td>
<td>91.4 ± 0.2</td>
</tr>
<tr>
<td>SVMrank</td>
<td>57.6 ± 0.2</td>
<td>84.7 ± 0.1</td>
<td>94.0 ± 0.1</td>
</tr>
<tr>
<td>λ-MART</td>
<td>57.4 ± 0.2</td>
<td>84.5 ± 0.1</td>
<td>93.8 ± 0.1</td>
</tr>
<tr>
<td>GCNN</td>
<td>65.5 ± 0.1</td>
<td>92.4 ± 0.1</td>
<td>98.2 ± 0.0</td>
</tr>
</tbody>
</table>

\(^2\)E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.
## Minimum set covering\(^3\)

<table>
<thead>
<tr>
<th>Model</th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Wins</td>
<td>Time</td>
</tr>
<tr>
<td>FSB</td>
<td>17.30</td>
<td>0 / 100</td>
<td>411.34</td>
</tr>
<tr>
<td>RPB</td>
<td>8.98</td>
<td>0 / 100</td>
<td>60.07</td>
</tr>
<tr>
<td>XTrees</td>
<td>9.28</td>
<td>0 / 100</td>
<td>92.47</td>
</tr>
<tr>
<td>SVMrank</td>
<td>8.10</td>
<td>1 / 100</td>
<td>73.58</td>
</tr>
<tr>
<td>(\lambda)-MART</td>
<td>7.19</td>
<td>14 / 100</td>
<td>59.98</td>
</tr>
<tr>
<td>GCNN</td>
<td>6.59</td>
<td>85 / 100</td>
<td>42.48</td>
</tr>
</tbody>
</table>

3 problem sizes

- 500 rows, 1000 cols (easy), training distribution
- 1000 rows, 1000 cols (medium)
- 2000 rows, 1000 cols (hard)

Pays off: better than SCIP’s default in terms of solving time.  
**Generalizes to harder problems!**

\(^3\)E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.
## Combinatorial auction

<table>
<thead>
<tr>
<th>Model</th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Wins</td>
<td>Nodes</td>
</tr>
<tr>
<td>FSB</td>
<td>4.11</td>
<td>0 / 100</td>
<td>6</td>
</tr>
<tr>
<td>RPB</td>
<td>2.74</td>
<td>0 / 100</td>
<td>10</td>
</tr>
<tr>
<td>XTrees</td>
<td>2.47</td>
<td>0 / 100</td>
<td>86</td>
</tr>
<tr>
<td>SVMrank</td>
<td>2.31</td>
<td>0 / 100</td>
<td>77</td>
</tr>
<tr>
<td>λ-MART</td>
<td>1.79</td>
<td>75 / 100</td>
<td>77</td>
</tr>
<tr>
<td>GCNN</td>
<td>1.85</td>
<td>25 / 100</td>
<td>70</td>
</tr>
</tbody>
</table>

3 problem sizes

- 100 items, 500 bids (easy), training distribution
- 200 items, 1000 bids (medium)
- 300 items, 1500 bids (hard)

---

## Capacitated facility location\textsuperscript{5}

<table>
<thead>
<tr>
<th>Model</th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Wins</td>
<td>Nodes</td>
</tr>
<tr>
<td>FSB</td>
<td>30.36</td>
<td>4 / 100</td>
<td>14</td>
</tr>
<tr>
<td>RPB</td>
<td>26.55</td>
<td>9 / 100</td>
<td>22</td>
</tr>
<tr>
<td>XTrees</td>
<td>28.96</td>
<td>3 / 100</td>
<td>135</td>
</tr>
<tr>
<td>SVMrank</td>
<td>23.58</td>
<td>11 / 100</td>
<td>117</td>
</tr>
<tr>
<td>λ-MART</td>
<td>23.34</td>
<td>16 / 100</td>
<td>117</td>
</tr>
<tr>
<td>GCNN</td>
<td><strong>22.10</strong></td>
<td><strong>57 / 100</strong></td>
<td>107</td>
</tr>
</tbody>
</table>

### 3 problem sizes

- 100 facilities, 100 customers (easy), training distribution
- 100 facilities, 200 customers (medium)
- 100 facilities, 400 customers (hard)

Empirical Study

## Maximum independent set

<table>
<thead>
<tr>
<th>Model</th>
<th>Time</th>
<th>Wins</th>
<th>Nodes</th>
<th>Time</th>
<th>Wins</th>
<th>Nodes</th>
<th>Time</th>
<th>Wins</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSB</td>
<td>23.58</td>
<td>9 / 100</td>
<td>7</td>
<td>1503.55</td>
<td>0 / 74</td>
<td>38</td>
<td>3600.00</td>
<td>0 / 0</td>
<td>n/a</td>
</tr>
<tr>
<td>RPB</td>
<td>8.77</td>
<td>7 / 100</td>
<td><strong>20</strong></td>
<td>110.99</td>
<td>41 / 100</td>
<td>729</td>
<td>2045.61</td>
<td>22 / 42</td>
<td><strong>2675</strong></td>
</tr>
<tr>
<td>XTrees</td>
<td>10.75</td>
<td>1 / 100</td>
<td>76</td>
<td>1183.37</td>
<td>1 / 47</td>
<td>4664</td>
<td>3565.12</td>
<td>0 / 3</td>
<td>38296</td>
</tr>
<tr>
<td>SVMrank</td>
<td>8.83</td>
<td>2 / 100</td>
<td>46</td>
<td>242.91</td>
<td>1 / 96</td>
<td><strong>546</strong></td>
<td>2902.94</td>
<td>1 / 18</td>
<td>6256</td>
</tr>
<tr>
<td>λ-MART</td>
<td>7.31</td>
<td>30 / 100</td>
<td>52</td>
<td>219.22</td>
<td>15 / 91</td>
<td>747</td>
<td>3044.94</td>
<td>0 / 12</td>
<td>8893</td>
</tr>
<tr>
<td>GCNN</td>
<td><strong>6.43</strong></td>
<td>51 / 100</td>
<td>43</td>
<td>192.91</td>
<td><strong>42</strong> / 82</td>
<td>1841</td>
<td><strong>2024.37</strong></td>
<td><strong>25</strong> / 29</td>
<td>2997</td>
</tr>
</tbody>
</table>

3 problem sizes, Barabási-Albert graphs (affinity=4)

- 500 nodes (easy), training distribution
- 1000 nodes (medium)
- 1500 nodes (hard)

---

Conclusion

Heuristic vs data-driven branching:

+ tune B&B to your problem of interest automatically
− no guarantees outside of the training distribution
− requires training instances


Code: https://github.com/ds4dm/learn2branch

Since:

▶ Branching for reinforcement learning Sun et al., 2021
▶ Properties of GCNNs for CO Cappart et al., 2021
▶ GNN-like operations on the CPU Gupta et al., 2020
▶ Ecole.ai: API + benchmarks for MILP adaptive solving (based on the open-source SCIP solver) – See Maxime’s talk later today.
Exact Combinatorial Optimization with Graph Convolutional Neural Networks

Thank you!

Laurent Charlin
with Maxime Gasse, Didier Chételat, Nicola Ferroni, Andrea Lodi