Learning-Driven Algorithms for Discrete Optimization

Bistra Dilkina

Assistant Professor of Computer Science, USC
Associate Director, USC Center of AI in Society

MIT CSAIL
April 30, 2019
Design of policies to manage limited resources for best impact translate into large-scale decision / optimization and learning problems, combining discrete and continuous effects.
Constraint Reasoning and Optimization

Decision making problems of larger size and new problem structure drive the continued need to improve combinatorial solving methods.
Opportunity:

Automatically tailor algorithms to a family of instances to discover novel search strategies

A realistic setting

• Same problem is solved repeatedly with slightly different data

• Delivery Company in Los Angeles:
  • Daily routing in the same area with slightly different customers
Learning-Driven Algorithm Design

ML Paradigm

Self-Supervised Learning
Reinforcement Learning
Supervised Learning

Problem Type

Graph Optimization
Integer Programming

Decision-focused Learning

General IP Heuristic

Greedy Heuristic

Exact Solving

Branching
Heuristic Selection
Mixed Integer Linear Programs (MIP) & their Applications

\[
(MIP) \quad z^* = \min \{ c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \ \forall j \in I \}
\]

Widely used to model real-world decision-making scenarios:

Conservation planning

Road Infrastructure Planning

Kidney exchange

Airline fleet and crew scheduling
Branch-and-Bound in a Nutshell

$x_2 = 0 \quad x_2 = 1$

$x_1 = 0 \quad x_4 = 0 \quad x_4 = 1$

$x_4 = 0 \quad x_4 = 1$

$x_k = 0 \quad x_k = 1$

$x_3$? $x_4 = 0 \quad x_4 = 1$

$x_5$? ...

$x_n$?
Branch-and-Bound in a Nutshell

\[
\min_{x} c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n
\]

Repeat:
- Select Node
- Solve Relaxation
- Add Cuts
- Run Heuristics
- Branch

Feasible solution? Update Best Solution

Worse than best solution? Prune!
Opportunities for Machine Learning in B&B

<table>
<thead>
<tr>
<th>Task</th>
<th>Issue</th>
<th>Current Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Branching Variable</td>
<td>what selection rule?</td>
<td>single hand-designed ranking metric</td>
</tr>
<tr>
<td>Add Cuts</td>
<td>which cuts to add?</td>
<td>hand-designed ranking formula</td>
</tr>
<tr>
<td>Run Heuristics</td>
<td>which heuristics to run?</td>
<td>run each every $k$ nodes (fixed parameter $k$)</td>
</tr>
<tr>
<td>Select Node</td>
<td>what selection rule?</td>
<td>best-first</td>
</tr>
</tbody>
</table>

Improve B&B with ML-driven strategies for these tasks.

- Most related work uses ML at a “meta level”, e.g. algorithm portfolios, parameter tuning, strategy selection.
- Here: use ML to discover novel strategies that dynamically guide the behavior of the BnB algorithm.
Opportunities for Machine Learning in B&B

<table>
<thead>
<tr>
<th>Task</th>
<th>Issue</th>
<th>Current Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Branching Variable</td>
<td>what selection rule?</td>
<td>single <strong>hand-designed</strong> ranking metric</td>
</tr>
<tr>
<td>Add Cuts</td>
<td>which cuts to add?</td>
<td><strong>hand-designed</strong> ranking formula</td>
</tr>
<tr>
<td>Run Heuristics</td>
<td>which heuristics to run?</td>
<td>run each every $k$ nodes (<strong>fixed parameter $k$)</strong></td>
</tr>
<tr>
<td>Select Node</td>
<td>what selection rule?</td>
<td>best-first</td>
</tr>
</tbody>
</table>

Improve B&B with ML-driven strategies for these tasks.

- Most related work uses ML at a “meta level”, e.g. algorithm portfolios, parameter tuning, strategy selection
- Here: use ML to discover **novel** strategies that **dynamically** guide the behavior of the BnB algorithm
Opportunities for Machine Learning in B&B

<table>
<thead>
<tr>
<th>Task</th>
<th>Issue</th>
<th>Current Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Branching Variable</td>
<td>what selection rule?</td>
<td>single hand-designed ranking metric</td>
</tr>
<tr>
<td>Add Cuts</td>
<td>which cuts to add?</td>
<td>hand-designed ranking formula</td>
</tr>
<tr>
<td>Run Heuristics</td>
<td>which heuristics to run?</td>
<td>run each every $k$ nodes (fixed parameter $k$)</td>
</tr>
<tr>
<td>Select Node</td>
<td>what selection rule?</td>
<td>best-first</td>
</tr>
</tbody>
</table>

Improve B&B with ML-driven strategies for these tasks.

- Most related work uses ML at a “meta level”, e.g. algorithm portfolios, parameter tuning, strategy selection
- Here: use ML to discover novel strategies that dynamically guide the behavior of the BnB algorithm
Branching on the “right” variables can have a dramatic impact on the number of nodes in B&B tree
- e.g. small backdoors in MIPs [Dilkina et al, CPAIOR 2009]

Joint work with my PhD student Elias Khalil, and our collaborators George Nemhauser, Le Song and Pierre Le Bodic [AAAI 2016]
Learning to Branch

**Ideally**, select variables that lead to small sub-tree ↔ many infeasible nodes

**Strong Branching (SB)** achieves that, but is extremely costly

---

**Given**: dataset of
(variable features, Strong Branching score)

**Learn**: a **ranking** model that imitates SB
Learning to Branch

**Given:** dataset of (variable features, Strong Branching score)

**Learn:** a ranking model that imitates SB

<table>
<thead>
<tr>
<th>$y_j$</th>
<th>feature 1</th>
<th>feature 2</th>
<th>feature 3</th>
<th>feature 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2$</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>$x_4$</td>
<td>1</td>
<td>0.3</td>
<td>0.9</td>
<td>0.4</td>
</tr>
<tr>
<td>$x_5$</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
</tr>
<tr>
<td>$x_6$</td>
<td>1</td>
<td>0.5</td>
<td>0.8</td>
<td>0.4</td>
</tr>
</tbody>
</table>

72 features, e.g.: objective coefficient, pseudocosts, statistics for constraint degrees in node subproblem + pairwise product of features

**labels:** 1 if $SB(x_i)$ close to max. $SB$ at node; 0 o.w.
Learning to Branch in Practice

Wildlife Corridor Design

Optimality gap (%)

- Pseudocost Branching
- Learning to Branch

Gap (Geometric Mean): 12
Gap (Median): 15
Gap (Maximum): 58

Gap (Geometric Mean): 1.2
Gap (Median): 1.2
Gap (Maximum): 8
Road Network Design for Flooding

- **Pseudocost Branching**
  - Gap (Geometric Mean): 15
  - Gap (Median): 0.7
  - Gap (Maximum): 14

- **Learning to Branch**
  - Gap (Geometric Mean): 7
  - Gap (Median): 0.0
  - Gap (Maximum): 75
Learning to Branch On-The-Fly (Per Instance)

MIPLIB2010 Benchmark

- **Pseudocost**
- **Strong Branching + Pseudocost**
- **Learning to Branch**

**Medium (120 instances):**
- Pseudocost: 395,199
- Strong Branching + Pseudocost: 288,916
- Learning to Branch: 234,093

**Hard (148 instances):**
- Pseudocost: 1,971,333
- Strong Branching + Pseudocost: 1,979,660
- Learning to Branch: 1,314,263

Compared to PC on Hard instances:
- 33% fewer nodes, 14% less time
Learning-Driven Algorithm Design

Takeaways

‣ First ML framework for branching
‣ Feature Engineering + Linear Ranking Model
‣ Significant improvements on families of instances
‣ On-the-fly version for limited data settings
In many applications, obtaining **good solutions quickly** is equally or more important than proving optimality.
Using Primal Heuristics in Branch-and-Bound

- **Primal heuristic**: an incomplete algorithm that can find a feasible solution (and hence can improve the primal bound).

- **Low Success Rate**: \( \frac{\text{#incumbents found}}{\text{#runs}} \)

- **Time Cost**: time for heuristic could instead be used to solve additional nodes.

- **Current approach in solvers**: run heuristic every \( k \) nodes
  - Frequency \( k \): a parameter that is manually tuned.

**Goal**

Automate this decision-making task using ML and improve performance

Joint work with my PhD student Elias Khalil, and our collaborators George Nemhauser, Shabbir Ahmed, Yufen Shao [IJCAI 2017]
Learning to Run Heuristics

Given: dataset of (node features, 0/1 success flag)

Learn: a classifier of heuristic success

Data Collection

New Instance

Machine Learning

Oracle: Success Prediction

1. Ask oracle for predictions
2. Oracle replies with predictions
3. Use oracle predictions
4. Decision: Run / Don’t run
Learning to Run Heuristics in Practice

Forest Harvesting: Generalized Independent Set

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>Learned</th>
<th>Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primal Integral</td>
<td>2,622</td>
<td>1,039</td>
<td></td>
</tr>
<tr>
<td></td>
<td>93</td>
<td>36</td>
<td>60% reduction</td>
</tr>
</tbody>
</table>
Learning-Driven Algorithm Design

Takeaways

‣ First ML framework for heuristic selection in B&B
‣ Dynamic, node-dependent decision-making
‣ Forest Harvesting: 60% reduction in Primal Integral
‣ MIPLIB2010: Even on the heterogeneous benchmark 6% reduction in Primal Integral
Goal: bring the state of the art in MIP solving to a significantly higher level by innovation at the intersection of ML + OR

- Explore variety of heuristic decisions in the MIP solver
- Consider different learning settings: Learning on-the-fly per instance, Learning offline over distribution of instances, Learning over sequences of streaming instances / changing distributions
- Explore diverse ML approaches in the BnB context (with theoretical guarantees?)
Algorithmic Template: Greedy

- **Minimum Vertex Cover**: Find smallest vertex subset $S$ s.t. each edge has at least one end in $S$
  - Example: advertising optimization in social networks
  - 2-approx:
    - **greedily** add vertices of edge with **max degree sum**

**Goal**: Learn a better criterion for greedy solution construction for a given **family of graphs**
Learning Greedy Heuristics

**Given:** graph problem, family of graphs

**Learn:** a scoring function to guide a greedy algorithm

<table>
<thead>
<tr>
<th>Problem</th>
<th>Minimum Vertex Cover</th>
<th>Maximum Cut</th>
<th>Traveling Salesman Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Social network snapshots</td>
<td>Spin glass models</td>
<td>Package delivery</td>
</tr>
<tr>
<td>Greedy operation</td>
<td>Insert nodes into cover</td>
<td>Insert nodes into subset</td>
<td>Insert nodes into sub-tour</td>
</tr>
</tbody>
</table>

Joint work with Elias Khalil, Hanjun Dai, Yuyu Zhang and Le Song [NIPS 2017]
Challenge #1: How to Learn

Possible approach: **Supervised learning**

- **Data**: collect *(partial solution, next vertex)* pairs
  - features
  - label
  - from precomputed (near) optimal solutions

**PROBLEM**

Supervised learning → Need to compute good/optimal solutions to NP-Hard problems in order to learn!!
Reinforcement Learning Formulation

Minimum Vertex Cover

\[
\min_{x_i \in \{0,1\}} \sum_{i \in V} x_i \\
\text{s.t.} \quad x_i + x_j \geq 1, \forall (i,j) \in E
\]

Start with \( \text{COVER} = \) empty

Repeat until all edges covered:
1. Compute score for each vertex
2. Select vertex with largest score
3. Add best vertex to \( \text{COVER} \)

Reward: \( r^t = -1 \)

State \( S \): current partial solution

SOLUTION

Improve policy by learning from experience → no need to compute optima

Greedy policy:
\[ v^* = \arg\max_v \hat{Q}(S,v) \]

Update state \( S \)
Challenge #2: How to Represent

• **Action value function:** $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex $v$ in state $S_t$

• **Representation of $v$**
  - A feature vector that describes $v$ in state $S_t$
Challenge #2: How to Represent

- Action value function: \( \hat{Q}(S_t, v; \theta) \)
  - Estimate of goodness of vertex \( v \) in state \( S_t \)

- Representation of \( v \): Feature engineering
  - Degree, 2-hop neighborhood size, other centrality measures…

PROBLEMS
1- Task-specific engineering needed
2- Hard to tell what is a good feature
3- Difficult to generalize across diff. graph sizes
We first provide an introduction to where the parameters \( G \) is computed based on the parameters from the graph embedding network, \( \Theta \). This approach is not applicable to our case due to the lack of training labels. Instead, we train a discriminative embeddings of latent variable models for structured data. This graph embedding network will compute the node features will propagate and get aggregated nonlinearly at distant nodes. In the end, if one makes the nonlinear transformations more powerful, we can add some more layers of neighborhood as determined by graph topology, the involved node features and the propagation process where the node features also defines a process where the node features are aggregated recursively according to an input graph structure.

\[ \mu_v^{(t+1)} \leftarrow \text{relu}(\theta_1 x_v + \text{Node’s own tag } x_v) + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u^{(t)} \text{ Neighbors’ features} + \theta_3 \sum_{u \in \mathcal{N}(v)} \text{relu}(\theta_4 w(v, u)) \text{ Neighbors’ edge weights} \]

\( \Theta : \text{model parameters} \)
Deep Representation Learning


Repeat embedding $T$ times:

For each node:

Update feature vector $\mu_v^{(t+1)}$.
where $v$. This approach is not applicable to our case due to the lack of training labels. Instead, we train on a collection of 7 parameters $\theta_{7, \mu_v(T)}$. For simplicity of exposition, over neighbors is one way of aggregating neighborhood information invariant to the permutation of linear unit($\theta_{4, \mu_{u}(T)}$). neighborhood as determined by graph topology, the involved node features and the propagation will depend on $\theta_{5, \mu_v(T)}$ at each node as $\theta_{6, \mu_{u}(T)}$. In the end, if one pass embedding $\theta_{3, \mu_{u}(T)}$ at each node as $\theta_{2, \mu_{u}(T)}$. After a few step of recursion, the network will produce a new embedding for each node, taking into account both graph characteristics and long-range interactions between these nodes.

### Compute Q-value:

$$\hat{Q}(h(S), v; \Theta) = \theta_{5}^T \text{relu}([\theta_{6} \sum_{u \in V} \mu_{u}(T), \theta_{7} \mu_v(T)])$$

- **Sum-pooling over nodes**
- $\Theta$: model parameters

### Repeat embedding $T$ times:

For each node:
- Update feature vector $\mu_{v}^{(t+1)}$. 

Deep Representation Learning
Deep Representation Learning

\[ \hat{Q}(S_t, v; \Theta) \]

Repeat embedding \( T \) times:

For each node:

1- No feature engineering needed
2- Features’ parameters trained to be good
3- Can handle different graph sizes

\[ Q(h(S), v; \Theta) = \theta_5 \text{ relu}(\theta_6 \sum_{u \in V} \mu_u^{(T)}, \theta_7 \mu_v^{(T)}) \]

Sum-pooling over nodes

\( \Theta \): model parameters
Minimum Vertex Cover - BA

Our Approach

S2V-DQN is near-optimal, barely visible.

Approximation Ratio

Number of nodes in train/test graphs

S2V-DQN
PN-AC [Vinyals et al. 2015]
MVCApprox
MVCApprox-Greedy
MaxCut - BA

Approximation Ratio

S2V-DQN  Our Approach
PN-AC    [Vinyals et al. 2015]
SDP
MaxcutApprox

Number of nodes in train/test graphs

15-20  40-50  50-100  100-200  200-300
TSP - clustered

Our Approach

Approximation Ratio

- S2V-DQN
- 2-opt
- PN-AC
- Cheapest
- Christofides
- Closest
- Nearest
- MST

[Vinyals et al. 2015]
Takeaways

- RL tailors greedy search to family of graph instances
- Learn features jointly with greedy policy
- Human priors encoded via meta-algorithm (Greedy)
- Interesting, novel strategies emerge
General Heuristics for Integer Optimization

\[
\min_{x} c^T x \quad s.t. \quad Ax \leq b, x \in \{0,1\}^n
\]

Power Systems

Data Center Resource Management

Software Verification

Strengths

• Applicable to many problems
• Usable inside Branch-and-Bound

Weaknesses

• May not work well for your problem
• Cannot exploit distribution of instances

Feasible Solution

General Heuristic
Feasible Region of LP Relaxation

Round to nearest integer

1. Round to nearest integer, return if LP-feasible
2. Project integer point to nearest LP-feasible point
3. Go back to step 1

Repeated Projections maintain constraint feasibility via LP solving

Figure in part from Berthold (2014)
Towards Learning General Heuristics

0 Start with LP-feasible (fractional) solution

\[ \text{Round to nearest integer, return if LP-feasible} \]

2 Project integer point to nearest LP-feasible point

3 Go back to step 1

Key Step:

\[ \min_x \Delta(x, [\bar{x}^t]) \]

s.t. \( Ax \leq b \), \( x \in [0, 1]^n \)

L1-distance

\[ \sum_j |x_j - [\bar{x}^t]_j| \]

= \sum_{j:[\bar{x}^t]=0} x_j + \sum_{j:[\bar{x}^t]=1} (1 - x_j) \]
Towards Learning General Heuristics

\[ \min_x \ell_1(x, [\bar{x}^t]) \]
\[ \text{s.t. } Ax \leq b, \quad x \in [0, 1]^n \]

**Properties of NeuralNet**

- **Parameters shared** across variables
- **Recurrent** across iterations

\[ p_i = \text{NeuralNet} \left( \bar{x}_i^t, [\bar{x}_i^t]; \Theta \right) \]

Learn the projection coefficients!!
How can we train the algorithm?

\[
\min_x c^T x \text{ s.t. } Ax \leq b, x \in \{0,1\}^n
\]

**input to Recurrent Neural Network** → **predict projection coefficients** → **solve LP projection** → **penalize fractional variables**

- History vector
- Iterate if \( \bar{x}^t \) is infeasible

### Diagram Elements
- \( p_1 \) from \( \bar{x}_1^t \)
- \( p_n \) from \( \bar{x}_n^t \)
- \( \text{Loss}(\bar{x}^t) \)
Learning IP Heuristics in Practice

Generalized Assignment Problem (GAP)
Train on small instances, Test on larger instances

Learned heuristic solves most instances in < 20 iterations
Learning IP Heuristics in Practice

Two-Stage Stochastic Integer Programs (STOC)
Train on small instances, Test on larger instances

STOC (k=10, p=10)

STOC (k=20, p=10)
Learning-Driven Algorithm Design

ML Paradigm
- Self-Supervised Learning
- Reinforcement Learning
- Supervised Learning

Greedy Heuristic
- General IP Heuristic
- Exact Solving
  - Branching
  - Heuristic Selection

Takeaways
- Incorporate LP-projections into neural network model
- Can learn heuristics for arbitrary Integer Programs
- No supervised or reinforcement learning required!
- Outperforms a basic Feasibility Pump generic IP heuristic
Learning-driven Algorithm Design

- Exciting and growing research area
- Many Open Directions:
  - Local Search, Nonlinear Optimization, Constraint Programming, Decision Diagrams
  - Theoretical Foundations
  - Lifelong Learning
The data-decisions pipeline

- Many real-world applications of AI involve a common template:
  - [Horvitz and Mitchell 2010; Horvitz 2010]
Typical two-stage approach

Machine learning models:
- Neural network
- Gaussian process
- Logistic regression
- Random forest

Goal: maximize accuracy

Optimization algorithms:
- Greedy
- Local search
- Mixed-integer program
- LP relaxation

Goal: maximize decision quality
Google maps

Data → Predictive model → Predicted travel times → Routing algorithm → Shortest path
Two-stage training

Data → Predictive model → Predicted delays vs Actual delays

Update model to make predictions closer to actual delays
Challenge

• Maximizing accuracy ≠ maximizing decision quality
• “All models are wrong, some are useful”
• Two-stage training doesn’t align with end goal
Decision-focused learning

[AAAI 2019, Bryan Wilder, Bistra Dilkina, Milind Tambe]
This work

*Automatically* shape the model’s loss by incorporating the optimization problem into the training loop

**Contributions:**
- Framework for integrating combinatorial optimization into ML training
- Efficient instantiation for
  - linear programs and
  - submodular maximization
- Experiments: when and how does decision-focused learning help?
Approach

• Objective function $f(x, \theta)$
  • $x \in \{0, 1\}^n$ is the **decision variable**
  • $\theta$ is an **unknown parameter** (e.g., true travel times)

• Idea: differentiate optimal solution with respect to $\theta$, train model via gradient descent (e.g. convex opt. [Donti et al ’17])

• Challenge: the optimization problem is discrete!
Approach

• Objective function $f(x, \theta)$
  • $x \in \{0, 1\}^n$ is the **decision variable**
  • $\theta$ is an **unknown parameter** (e.g., true travel times)

• Idea: differentiate optimal solution with respect to $\theta$, train model via gradient descent (e.g. convex opt. [Donti et al ’17])

• Challenge: the optimization problem is discrete!
• **Solution**: relax to continuous problem, differentiate, round

![Discrete vs Continuous Formulation](image.png)

- $x = \text{binary decision}$
- $F = \text{continuous objective}$

Note: The diagram illustrates the transformation from a discrete decision variable $x$ to a continuous one $F$, where $x$ is binary and $F$ is fractional in the continuous problem formulation.
Differentiating through optimization

• How to compute $\frac{dx^*}{d\theta}$?
  • Differentiate the output of the optimization algorithm wrt predictions

• Idea: (locally) optimal solution must satisfy KKT conditions

• Differentiate those equations at optimum

• See paper for details
Results

- Decision-focused has consistently **better solution quality**
  - 15-70% improvement, across three domains

| $k = |$ | Budget allocation | Matching | Diverse recommendation |
|---|---|---|---|
| | 5 | 10 | 20 | 5 | 10 | 20 |
| NN1-Decision | 49.18 ± 0.24 | 72.62 ± 0.33 | 98.95 ± 0.46 | 2.50 ± 0.56 | 15.81 ± 0.50 | 29.81 ± 0.85 | 52.43 ± 1.23 |
| NN2-Decision | 44.35 ± 0.56 | 67.64 ± 0.62 | 93.59 ± 0.77 | 6.15 ± 0.38 | 13.34 ± 0.77 | 26.32 ± 1.38 | 47.79 ± 1.96 |
| NN1-2Stage | 32.13 ± 2.47 | 45.63 ± 3.76 | 61.88 ± 4.10 | 2.99 ± 0.76 | 4.08 ± 0.16 | 8.42 ± 0.29 | 19.16 ± 0.57 |
| NN2-2Stage | 9.69 ± 0.05 | 18.93 ± 0.10 | 36.16 ± 0.18 | 3.49 ± 0.32 | 11.63 ± 0.43 | 22.79 ± 0.66 | 42.37 ± 1.02 |

- But typically much less accurate (wrt AUC, MSE etc.)
Application: Tuberculosis treatment

- Follow-on work improving treatment in Indian TB system
- In collaboration with Everwell (NGO)
- **Predict** when patients will miss a dose
- **Optimize** health worker visits

Learning to Prescribe Interventions for Tuberculosis Patients using Digital Adherence Data
J. Killian, B. Wilder, A. Sharma, V. Choudhary, B. Dilkina, M. Tambe
KDD 2019
Application: Tuberculosis treatment

Less “accurate”, but +15% successful interventions!
Thank you!