ORF522 – Linear and Nonlinear Optimization
22. Data-driven algorithms
Ed forum

- Updated proof of spacial branch and bound convergence to clarify last step.

- Although on slide 15 we assume that lower bound $L$ is non-decreasing, what if after a new refinement and a new relaxation process at step $k+1$, our new lower bound $L^{k+1} \leq L^k$? Does this happen in applications? If it happens, do we keep the new one ($L^{k+1}$) or do we keep the "better" one ($L^k$).
Today’s lecture

[The Voice of Optimization, Bertsimas and Stellato]
[Online Mixed-Integer Optimization in Milliseconds, Bertsimas and Stellato]
[On learning and branching: a survey, Lodi and Zarpellon]

Data-driven algorithms (research topics)

• Machine learning
• Learning heuristics in branch and bound algorithms
• Learning strategies for parametric optimization
  - Strategies definition
  - Learning and sampling the strategies
  - Examples
Methods for nonconvex optimization

Convex optimization algorithms: global and typically fast

Nonconvex optimization algorithms: must give up one, global or fast

• **Local methods:** fast but not global
  Need not find a global (or even feasible) solution. They cannot certify global optimality because KKT conditions are not sufficient.

• **Global methods:** global but often slow
  They find a global solution and certify it.
Data to the rescue!

Nonconvex optimization is hard

Many algorithmic choices inside solvers

Lots of data available from experience

Can we use machine learning to build better algorithms?
Similar problems

- In practice, we solve many similar problems with varying data
- Most solvers do not exploit it
- We will consider families of similar problems
Machine learning
Imitation learning

Machine Learning

- Discover patterns
- Understand structure

Minimize expected loss

\[
\min_w \mathbb{E}_{X,Y \in \mathcal{P}} \ell(Y, f_w(X))
\]

\(f_w: \) model

\(w: \) parameters

(we do not know \(\mathcal{P}\))

Training data

\[D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N\]

Empirical probability

\[
\min_w \sum_{i=1}^N \ell(y_i, f_w(x_i))
\]
Learning algorithmic decisions

Learning from demonstrations

\[ D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N} \]

state, situation, conditions…

expert decisions

Goal: mimic expert decisions as closely as possible
Learning heuristics in branch and bound algorithms
Branch and bound for integer optimization

minimize \( c^T x \)
subject to \( Ax \leq b \)
\( x \in \{0, 1\}^n \)

1. **Branch:** pick node \( i \) and index \( k \)
   form subproblems for \( x_k = 0 \) and \( x_k = 1 \)

2. **Bound:**
   - Compute **lower** and **upper bounds**
   - Update global lower bounds on \( f(x^*) \)
     \[
     L = \min_i \{L_i\}, \quad U = \min_i \{U_i\} \]

3. If \( U - L \leq \epsilon \), **break**
Branch and bound decisions

Node selection: which node $i$?
- best-first: node with smallest lower bound
- depth-first: node with greatest depth

Variable selection: which fractional variable $k$?
- “least ambivalent”: $x_k^* \approx 0$ or $1$
- “most ambivalent”: $|x_k^* - 1/2|$ is minimum

Heuristic selection: which upper bound algorithm? when?
- Rounding
- Randomization
- Neighborhood search

Can we learn better heuristics from data?
Variable selection and strong branching

Relaxed problem at node $i$

minimize $c^T x$

subject to $Ax \leq b$

integer fixed components

$\underline{x} = \bar{x}$

$0 \leq x \leq 1$

Potential branching variables

Fractional $x_k$, $k \in \mathcal{F} = \{1, \ldots, n\} \setminus \mathcal{I}$

Strong branching

- Split all potential candidates $k$
- For each one, solve relaxed problems for $x_k = 0$ and $x_k = 1$
- Pick $k$ with highest "score": the left and right lower bound increase the most

Too expensive!
Learning strong branching

Node features \( \theta_i \) \( \rightarrow \) Strong branching scores \( (f_w(\theta_i))_k = s_k, \; k \in \mathcal{F} \) \( \rightarrow \) Best variable \( k = \arg\max_k s_k \)

Feature types

- **Static (problem instance):**
  - objective function coefficients,
  - constraint coefficients stats.,
  - constraint degrees (# of variables), etc.
- **Dynamic (incumbent, current LP relaxation, etc.):**
  - relaxed \( x^* \) distance to rounding,
  - constraint degrees (# of variables), etc.

Multiclass classifier

- Linear function (SVM\(^{\text{rank}}\))
- Decision tree
- Neural network
Learning strong branching results

MIPLIB Examples with node limit 10,000

<table>
<thead>
<tr>
<th>Solved by all methods</th>
<th>Not solved by at least one method</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/T</td>
<td>Nodes</td>
</tr>
<tr>
<td>Most ambivalent</td>
<td>9/44</td>
</tr>
<tr>
<td>Strong</td>
<td>9/44</td>
</tr>
<tr>
<td>Learned</td>
<td>9/44</td>
</tr>
</tbody>
</table>

Extensions

- What if we learn the 2-step strong branching (doubly-strong branching)?
- Can we learn while we solve the problem?

[A Machine Learning-Based Approximation of Strong Branching, Marcos Alvarez, Wehenkel, Louveaux]
Many more directions in branch and bound

**Optimal node selection**
[Learning to Search in Branch-and-Bound Algorithms, He et al]

**Upper bound heuristic selection**
[Learning to Run Heuristics in Tree Search, Khalil et al]

**What if we do not have expert demonstrations?**

**Reinforcement learning**
ecole.ai: OpenAI gym-like environment for Reinforcement Learning and Combinatorial Optimization
Learning for parametric optimization
Parametric optimization

Limitations

\[
\begin{align*}
\text{minimize} & \quad f(x, \theta) \\
\text{subject to} & \quad g(x, \theta) \leq 0
\end{align*}
\]

Real-time optimization

Fast real-time requirements  +  Low-cost computing platforms
End to end learning

\[ \theta \rightarrow \text{Machine Learning} \rightarrow x^* \]

Very small problems
Imprecise
Needs lots of “babysitting”

[Smith (1999)]
[Bello et al (2017)]
[Vinyals et al (2017)]
Machine learning optimizer

$\theta \rightarrow \text{Machine Learning} \rightarrow s(\theta) \rightarrow \text{Solution Decoding} \rightarrow x^*$

[The Voice of Optimization, Bertsimas and Stellato]
[Online Mixed-Integer Optimization in Milliseconds, Bertsimas and Stellato]
Strategies in optimization
What is a strategy?

The complete information we need to efficiently compute the optimal solution.
Parametric linear optimization

minimize \( c(\theta)^T x \)
subject to \( A(\theta)x \leq b(\theta) \)

How can we define a strategy?
Tight constraints in linear optimization

\[ \mathcal{T}(\theta) = \{ i \mid A_i(\theta)x^* = b_i(\theta) \} \]

**Strategies for linear optimization**

\[ s(\theta) = \mathcal{T}(\theta) \]

| \( \mathcal{T}(\theta) \) | = # variables if non-degenerate
| \( \mathcal{T}(\theta) \) | \( \ll \) # constraints in general
Computing the solution from the strategy

\[ s(\theta) \rightarrow \text{Solution Decoding} \rightarrow x^* \]

Convex optimization

minimize \[ c(\theta)^T x \]
subject to \[ A(\theta)x \leq b(\theta) \]

minimize \[ c(\theta)^T x \]
subject to \[ A_i(\theta)x = b_i(\theta), \quad \forall i \in T(\theta) \]

KKT Linear system
Parametric mixed-integer linear optimization

minimize \( c(\theta)^T x \)
subject to \( A(\theta)x \leq b(\theta) \)
\( x_I \in \mathbb{Z}^d \)

integers

How can we define a strategy?
Tight constraints are not enough

\[ Ax \leq b \]

\[ x^* \]

\[ x_1 \leq b \]

\[ x_2 \]
Strategies for mixed-integer optimization

\[ s(\theta) = (T(\theta), x^*_I(\theta)) \]

Tight constraints \quad Integer variables

\[ Ax \leq b \]

\[ x^* \]
Computing the solution from the strategy

\[ s(\theta) \xrightarrow{\text{Solution Decoding}} x^* \]

Convex optimization

\begin{align*}
\text{minimize} & \quad c(\theta)^{T} x \\
\text{subject to} & \quad A(\theta) x \leq b(\theta) \\
& \quad x_{\mathcal{I}} \in \mathbb{Z}^{d} \end{align*}

\begin{align*}
\text{minimize} & \quad c(\theta)^{T} x \\
\text{subject to} & \quad A_i(\theta) x = b_i(\theta), \quad \forall i \in \mathcal{T}(\theta) \\
& \quad x_{\mathcal{I}} = x_{\mathcal{I}}^*(\theta) \end{align*}

KKT Linear system
Mixed-integer convex optimization

minimize \( f(x, \theta) \)
subject to \( g(x, \theta) \leq 0 \)
\( z_\mathcal{I} \in \mathbb{Z}^d \)

Same strategy definition
\( s(\theta) = (\mathcal{T}(\theta), x^*_\mathcal{I}(\theta)) \)

How can we recover the solution?
Learning the strategies
Predicting the strategies

\[ \theta \xrightarrow{\text{Machine Learning}} s(\theta) \xrightarrow{\text{Strategy}} x^* \]

\[ N \text{ data } (\theta_i, s(\theta_i)) \]
\[ M \text{ labels (strategies) } S \]

\[ \text{Multiclass classification} \]

\[ \theta \xrightarrow{\text{Machine Learning}} \hat{s}(\theta) \]
Interpretable classifier

Decision Trees

$\theta_1 < 2.45$

3

$\theta_2 < 1.75$

$\theta_3 < 4.95$

Features

- Easy to understand
- It works for small problems

[Optimal Classification Trees, Bertsimas and Dunn]
Neural network classifiers

\[
\begin{align*}
\theta &\rightarrow f_1 \rightarrow y_1 \rightarrow f_2 \rightarrow y_2 \ldots \rightarrow f_{L-1} \rightarrow y_{L-1} \rightarrow f_L \rightarrow \hat{s}
\end{align*}
\]

**Single layer**
\[y_i = f(y_{i-1}) = (W_i y_{i-1} + b_i)_+\]

**Output layer** (softmax)
\[
\hat{s} = f(y_L) = \sigma(y_L), \quad \text{with} \quad (\sigma(x))_i = \frac{e^{x_i}}{\sum_{j=1}^M e^{x_j}}
\]

**Features**
- Hard to understand
- It works for large problems
Sampling the strategies
Have we seen enough data?

Multiclass classification

\[ \theta \rightarrow \text{Machine Learning} \rightarrow \hat{s}(\theta) \]

\( N \) data \((\theta_i, s(\theta_i))\)

\( M \) labels (strategies) \( S \)

\[ \theta_1 \]
\[ \theta_2 \]
\[ \vdots \]
\[ \theta_N \]

\[ s_1 \]
\[ s_2 \]
\[ \vdots \]
\[ s_M \]

What happens with \( \theta_{N+1} \)?
Alan Turing
Already worked on this…
Good-Turing estimator

\[ \text{# strategies appeared once} \]
\[ GT = \frac{N_1}{N} \approx P(s(\theta_{N+1}) \notin S(\Theta_N)) \]

Probability of unseen strategies

**Concentration bound** (confidence \( \beta \))
\[ P(s(\theta_{N+1}) \notin S(\Theta_N)) \leq GT + C \sqrt{\frac{1}{N} \ln \left( \frac{3}{\beta} \right)} \]

Sample until \( \leq \epsilon \)

**Example**

\( N = 15 \)
\( M = 5 \)

\( s_1 \) 6 times
\( s_2 \) 3 times
\( s_3 \) 1 time
\( s_4 \) 3 times
\( s_5 \) 2 times

\( GT = 1/15 \)
MLOPT: Machine Learning Optimizer

github.com/bstellato/mlopt

Offline learning

CVXPY modelling → Strategy sampling → ML predictor training

Fast online predictions

Machine Learning → Top-$k$ Strategies → Solution Decoding → $x^*$
Examples
Inventory management

minimize

subject to

inventory

parameters

order

demand

$\sum_{t=0}^{T-1} h(x_t) + o(u_t)$

$x_{t+1} = x_t + u_t - d_t$

$x_0 = x_{\text{init}}$

$0 \leq u_t \leq M$
Inventory management strategies

Strategies:

- **Strategy 2**
  - $x_{\text{init}} < 7.91$
  - $d_5 < 1.91$
  - $x_{\text{init}} < 11.61$
  - $d_5 < 1.91$
  - $x_{\text{init}} < 12.2$

- **Strategy 4**
  - $x_{\text{init}} < 9.97$
  - $x_{\text{init}} < 7.91$

Minimize subject to:

$$
\sum_{t=0}^{T-1} h(x_t) + o(u_t)
$$

$$
x_{t+1} = x_t + u_t - d_t
$$

$$
x_0 = x_{\text{init}}
$$

$$
0 \leq u_t \leq M
$$

Parameters:

- $u_t = 0$
- $t \leq 3$
- $0 \leq u_t \leq M$
- $t > 3$
- $0 \leq u_t \leq M$
- $t > 4$

Summary:

- **Strategy 2**
  - $u_t = 0$
  - $t \leq 4$
  - $0 \leq u_t \leq M$
  - $t > 4$

- **Strategy 4**
  - $u_t = 0$
  - $t \leq 3$
  - $0 \leq u_t \leq M$
  - $t > 3$
Inventory management trajectory

Strategy 2

\[ u_t = 0 \quad t \leq 4 \]

\[ 0 \leq u_t \leq M \quad t > 4 \]
Example

Motion planning with obstacles

$p^{\text{init}}$ $p^{\text{des}}$

$p_t$ position $\in \mathbb{R}^d$
$v_t$ velocity $\in \mathbb{R}^d$

$p^{\text{init}}$ initial position
$v^{\text{init}}$ initial velocity

$p^{\text{des}}$ desired position

Obstacles

Obstacle $i$ is a box $[\underline{o}^i, \overline{o}^i]$
Motion planning formulation

\[
\text{minimize} \quad \|p_T - p^{\text{des}}\|_2^2 + \sum_{t=0}^{T-1} \|p_t - p^{\text{des}}\|_2^2 + \gamma \|u_t\|_2^2
\]

subject to

\[
(p_{t+1}, v_{t+1}) = A(p_t, v_t) + B u_t
\]
\[
p_0 = p^{\text{init}}, \quad v_0 = v^{\text{init}}
\]

\[
\overline{o}_i - M \overline{\delta}_t \leq p_t \leq \underline{o}_i + M \underline{\delta}_t, \quad i = 1, \ldots, n_{\text{obs}}
\]
\[
1^T \delta_t^i + 1^T \overline{\delta}_t^i \leq 2d - 1
\]

\[
\overline{\delta}_t^i, \underline{\delta}_t^i \in \{0, 1\}^d, \quad i = 1, \ldots, n_{\text{obs}}
\]
## Motion planning with obstacles

### Worst-case timings

<table>
<thead>
<tr>
<th>$n_{obstacles}$</th>
<th>$n_{var}$</th>
<th>$n_{constr}$</th>
<th>$t_{max}$ MLOPT [s]</th>
<th>$t_{max}$ Gurobi [s]</th>
<th>$t_{max}$ Gurobi heuristic [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1135</td>
<td>3773</td>
<td>0.4145</td>
<td>2.3776</td>
<td>2.2962</td>
</tr>
<tr>
<td>4</td>
<td>1615</td>
<td>10133</td>
<td>0.1878</td>
<td>11.8172</td>
<td>8.1443</td>
</tr>
<tr>
<td>6</td>
<td>2095</td>
<td>20333</td>
<td>0.3173</td>
<td>33.7869</td>
<td>11.5292</td>
</tr>
<tr>
<td>8</td>
<td>2575</td>
<td>34373</td>
<td>0.2235</td>
<td>392.3073</td>
<td>128.4948</td>
</tr>
<tr>
<td>10</td>
<td>3055</td>
<td>52253</td>
<td>0.2896</td>
<td>773.1476</td>
<td>206.4520</td>
</tr>
</tbody>
</table>

**2600x speedups**

[Online Mixed-Integer Optimization in Milliseconds, Bertsimas and Stellato]
Motion planning with obstacles

Circles
optimal

Squares
MLOPT
Learning strategies in parametric optimization

Benefits

• Extremely fast
• Simple online method for nonconvex optimization
• It learns from your pool of problems

Downsides

• No optimality guarantees
• Relies on many offline solutions (expert demonstrations)

Future directions

• Better NN architectures
• Optimality guarantees
• Reinforcement learning when we do not have offline solutions
Data-driven algorithms

Today, we learned recent research on data-driven algorithms:

• **Learning heuristics** in branch and bound search (global algorithm)
• **Learning strategies** in parametric optimization (heuristic algorithm)

**Many more exciting directions**

Differentiable optimization layers, reinforcement learning in optimization, learning-augmented first order methods, …

[CS159 Caltech, https://sites.google.com/view/cs-159-spring-2020/]
Next lecture

• Course recap and conclusions