Learning to Branch

Ellen Vitercik

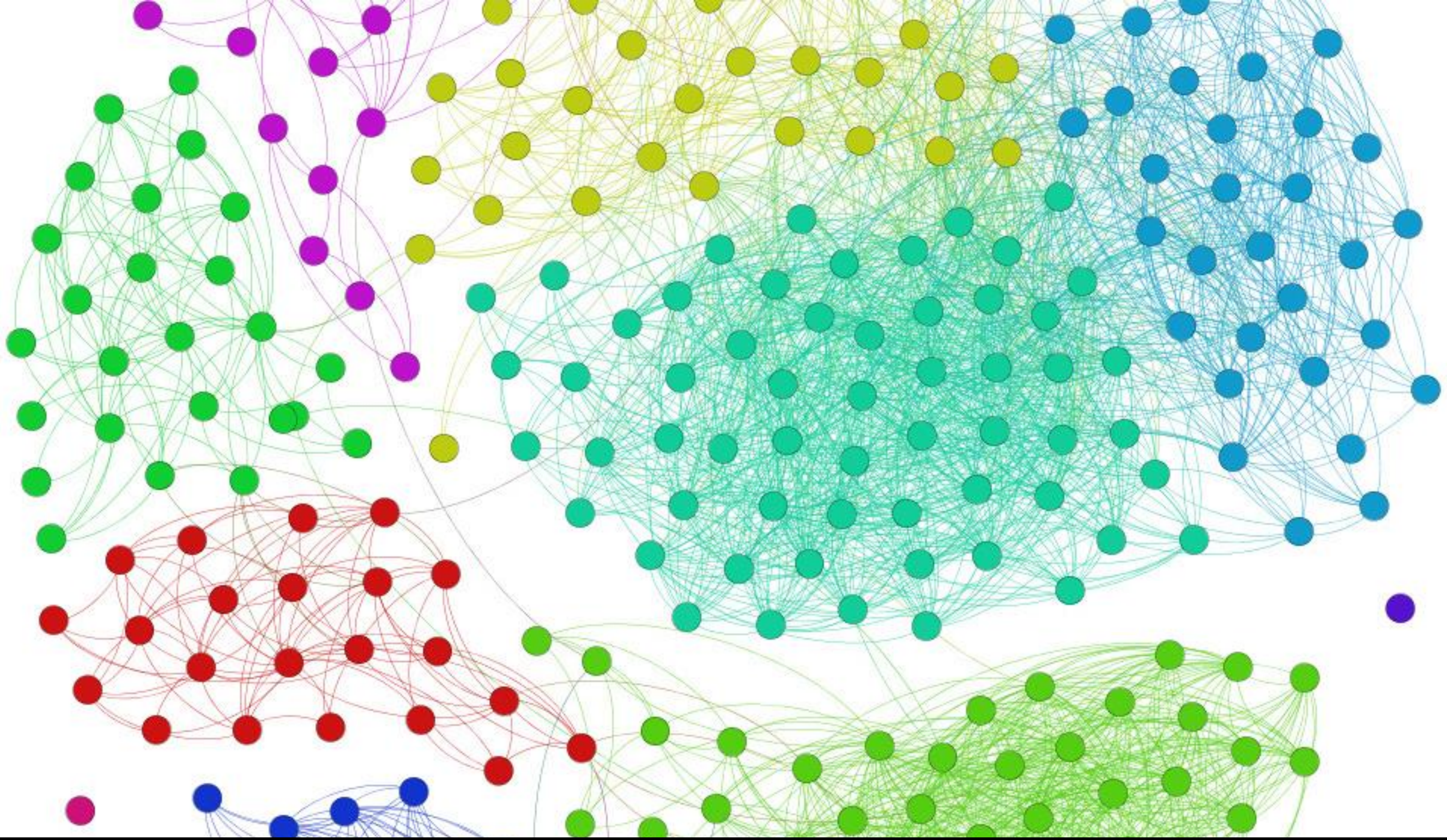
Joint work with Nina Balcan, Travis Dick, and Tuomas Sandholm

Published in ICML 2018

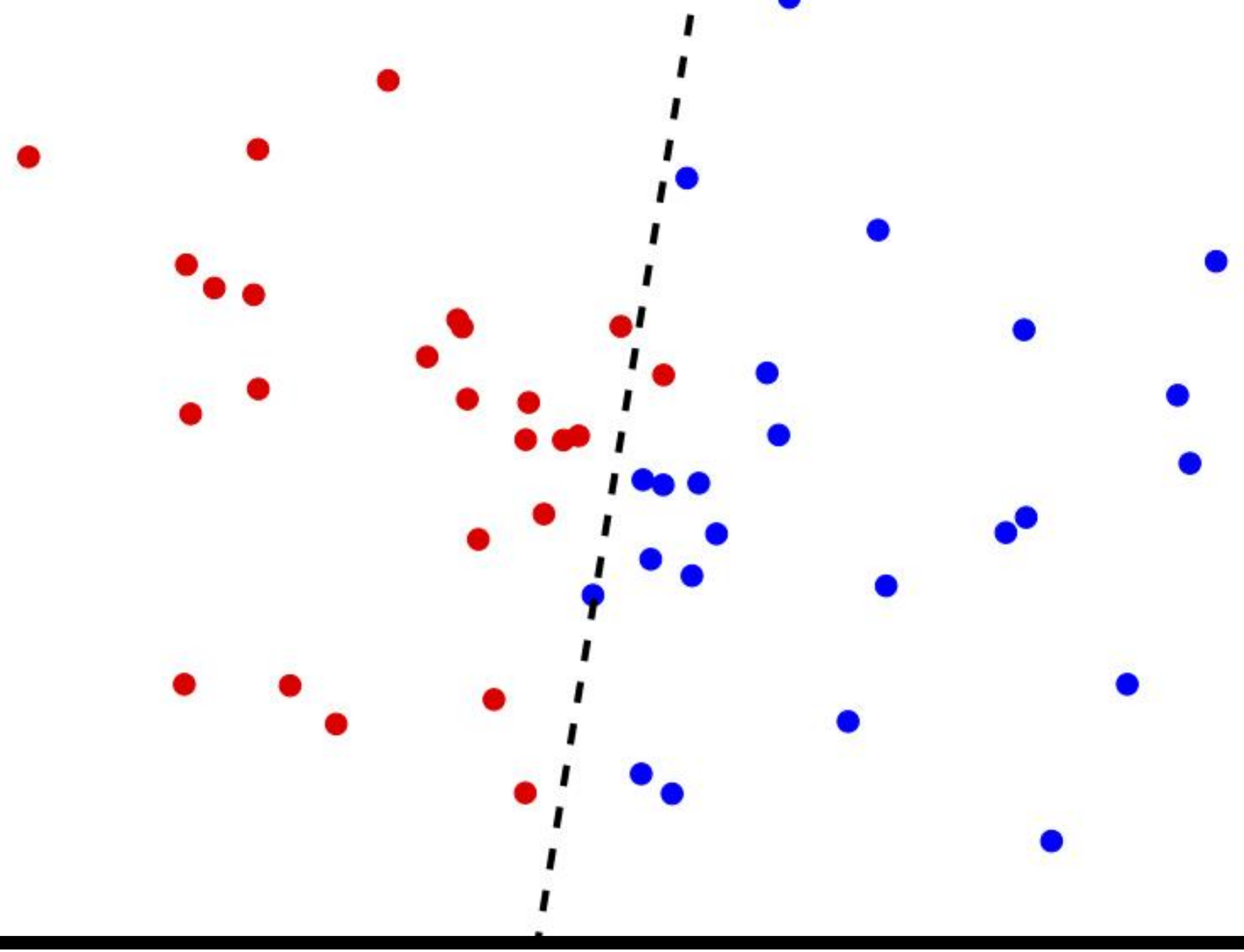
Integer Programs (IPs)

$$
\begin{array}{ll}
\text{maximize} & \boldsymbol{c} \cdot \boldsymbol{x} \\
\text{subject to} & A\boldsymbol{x} \leq \boldsymbol{b} \\
 & \boldsymbol{x} \in \{0,1\}^n
\end{array}
$$

Facility location problems can be formulated as IPs.



Clustering problems can be formulated as IPs.



Binary classification problems can be formulated as IPs.

Integer Programs (IPs)

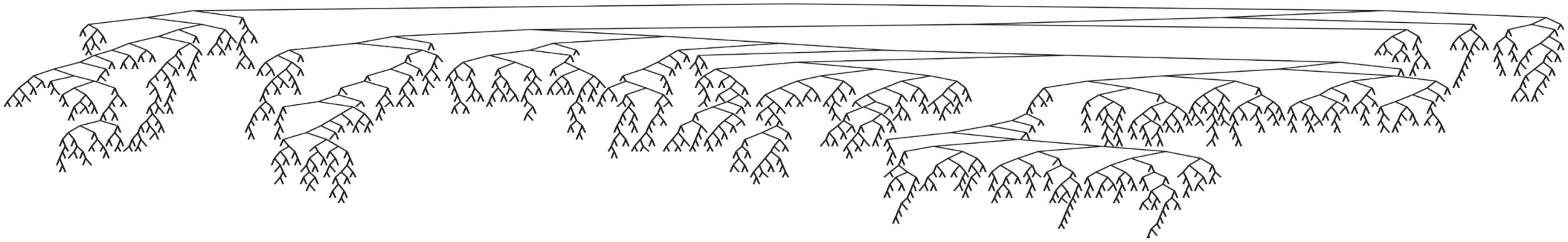
$$\begin{aligned} \text{maximize} \quad & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \{0,1\}^n \end{aligned}$$

NP-hard

Branch and Bound (B&B)

- Most widely-used algorithm for IP-solving (CPLEX, Gurobi)
- Recursively partitions search space to find an optimal solution
  - Organizes partition as a tree
- **Many** parameters
  - CPLEX has a 221-page manual describing 135 parameters

    "You may need to experiment."



Why is tuning B&B parameters important?

- Save time
  - Solve more problems
- Find better solutions

B&B in the real world

Delivery company routes trucks daily

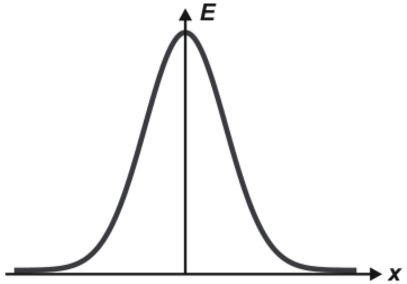
- Use integer programming to select routes

Demand changes every day

- Solve hundreds of similar optimizations

Using this set of typical problems…
can we **learn** best parameters?

Model



Application-Specific Distribution

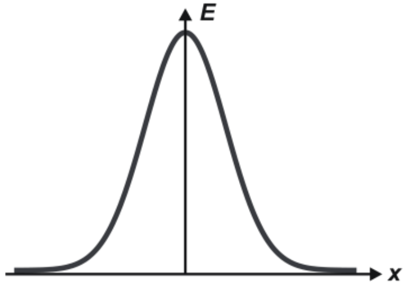
$$\left\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\right\}, \ldots, \left\{A^{(m)}, \boldsymbol{b}^{(m)}, \boldsymbol{c}^{(m)}\right\}$$

Algorithm Designer

B&B parameters

How to use samples to find best B&B parameters for my domain?

Model



Application-Specific Distribution

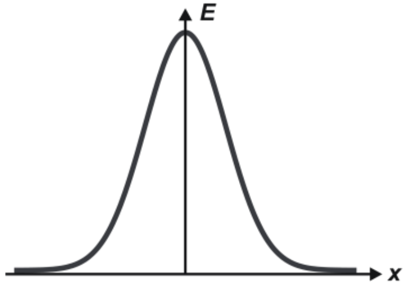
$$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}, \dots, \{A^{(m)}, \boldsymbol{b}^{(m)}, \boldsymbol{c}^{(m)}\} \longrightarrow$$

Algorithm Designer

$\boxed{\text{B\&B parameters}} \longleftarrow$

Model has been studied in applied communities [Hutter et al. ‘09]

Model



Application-Specific Distribution

$$\left\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\right\}, \ldots, \left\{A^{(m)}, \boldsymbol{b}^{(m)}, \boldsymbol{c}^{(m)}\right\} \longrightarrow$$

Algorithm Designer

$\longleftarrow$ B&B parameters

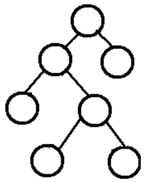
Model has been studied from a theoretical perspective

[Gupta and Roughgarden ‘16, Balcan et al., ‘17]

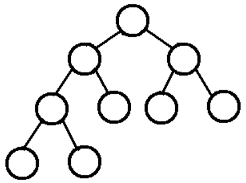
Model

1. Fix a set of B&B parameters to optimize
2. Receive sample problems from unknown distribution

$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}$



$\{A^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}\}$



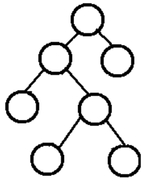
3. Find parameters with the best performance on the samples

“Best” could mean smallest search tree, for example

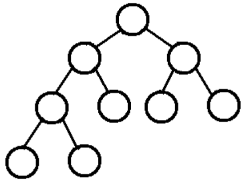
Questions to address

How to find parameters that are best on average over samples?

$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}$



$\{A^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{c}^{(2)}\}$



$\{A, \boldsymbol{b}, \boldsymbol{c}\}$

?

Will those parameters have high performance in expectation?

Outline

max	$(40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x}$
s.t.	$(40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100$
	$\boldsymbol{x} \in \{0,1\}^7$

max	$(40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x}$
s.t.	$(40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100$
	$\boldsymbol{x} \in \{0,1\}^7$

$\left(\frac{1}{2}, 1, 0, 0, 0, 0, 1\right)$
140

B&B

1. Choose leaf of tree

max	$(40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x}$
s.t.	$(40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100$
	$\boldsymbol{x} \in \{0,1\}^7$

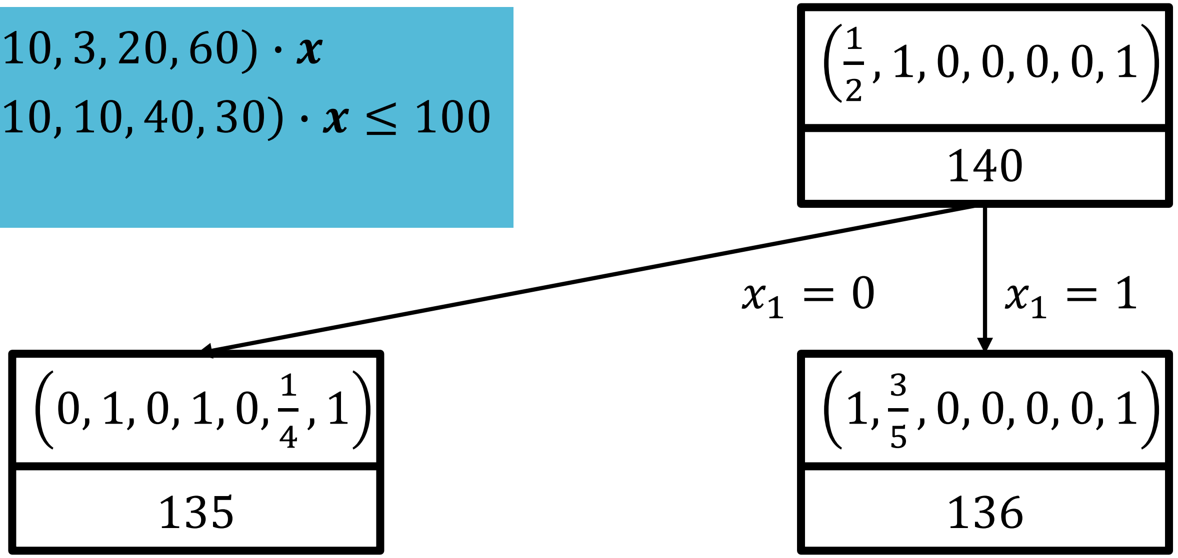
$\left(\frac{1}{2}, 1, 0, 0, 0, 0, 1\right)$

140

B&B

1. Choose leaf of tree
2. **Branch** on a variable

max	$(40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x}$
s.t.	$(40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100$
	$\boldsymbol{x} \in \{0,1\}^7$

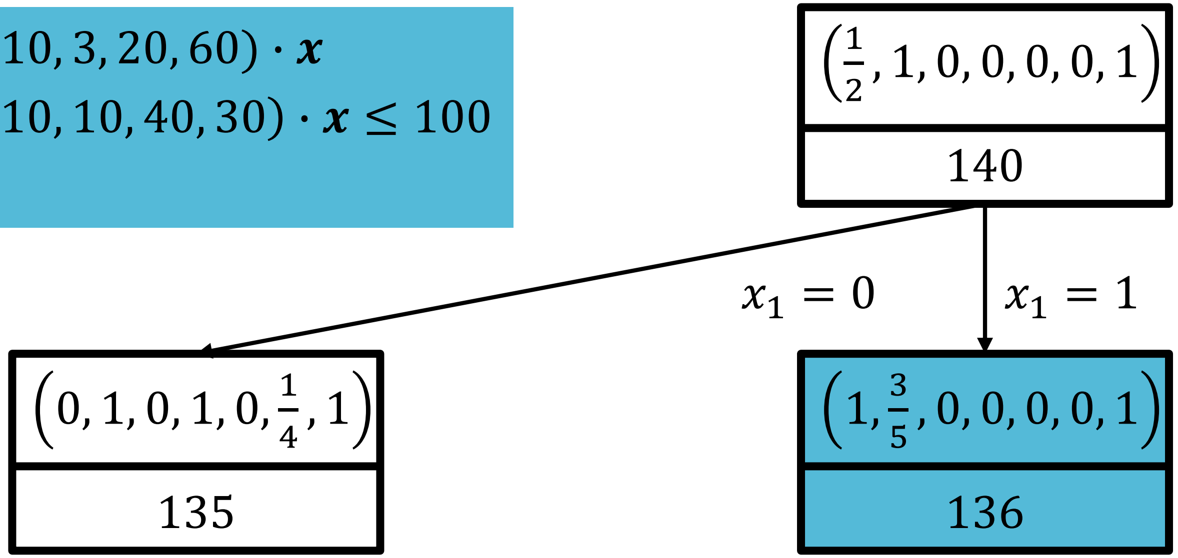


B&B

1. Choose leaf of tree

2. Branch on a variable

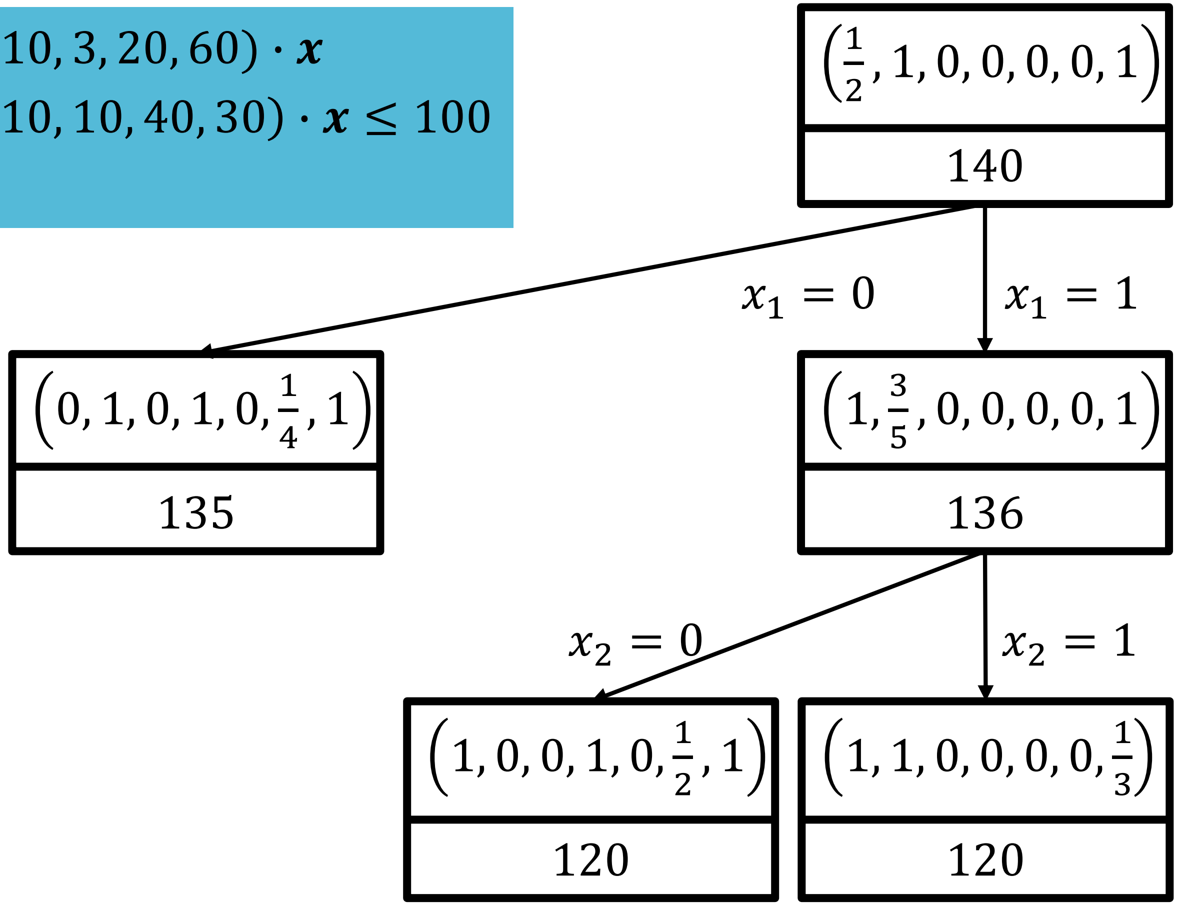
max	$(40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x}$
s.t.	$(40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100$
	$\boldsymbol{x} \in \{0,1\}^7$



B&B

1. Choose leaf of tree
2. **Branch** on a variable

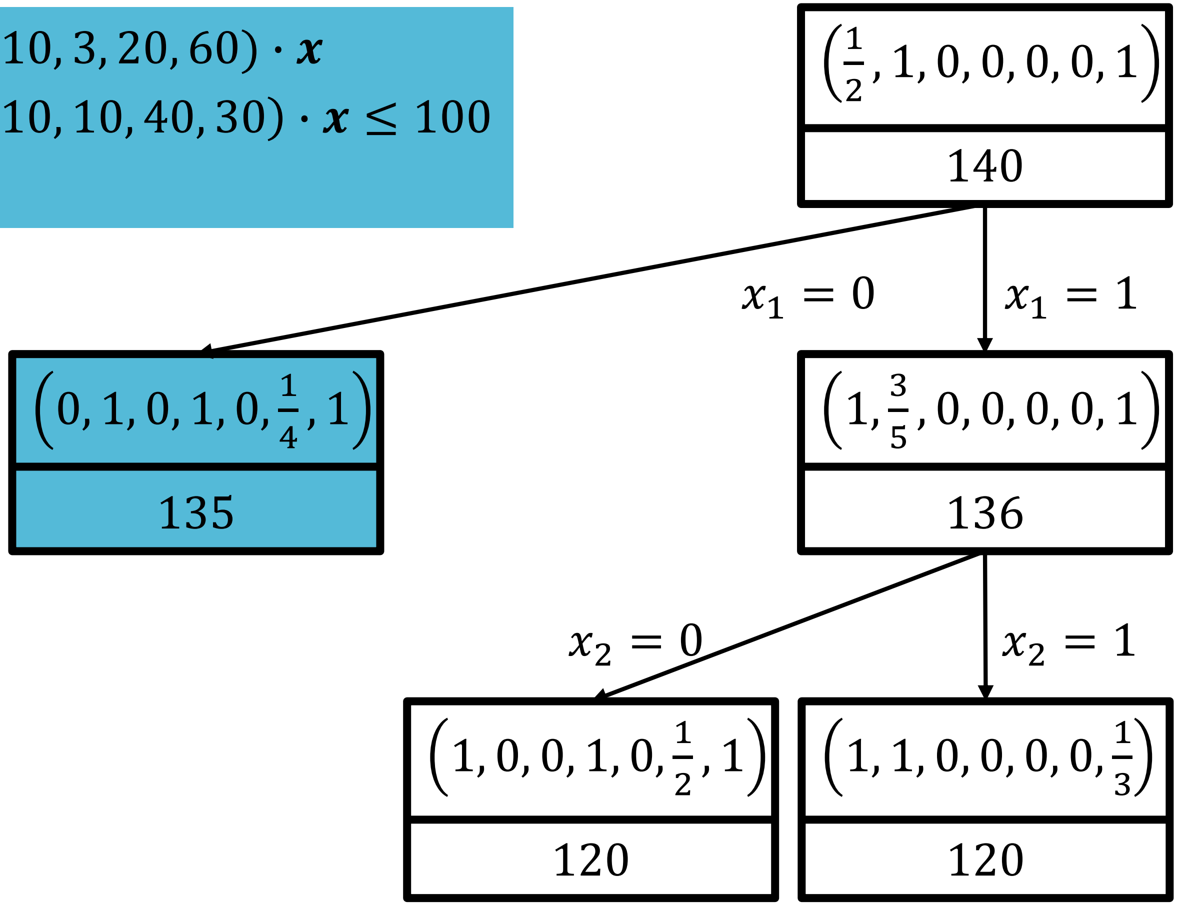
$$\begin{array}{ll} \text{max} & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x} \\ \text{s.t.} & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$



B&B

$$\begin{array}{ll} \text{max} & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x} \\ \text{s.t.} & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$

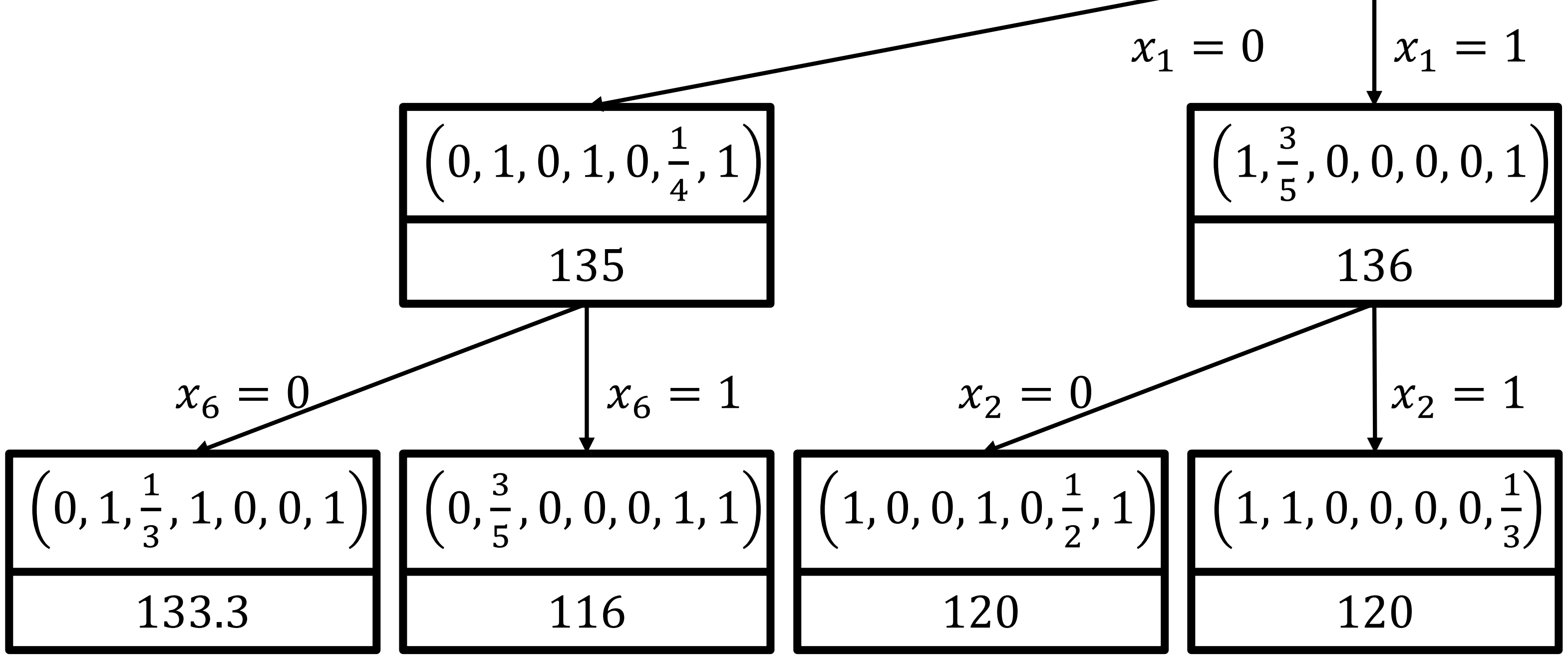
1. **Choose leaf of tree**
2. Branch on a variable



B&B

1. Choose leaf of tree
2. **Branch** on a variable

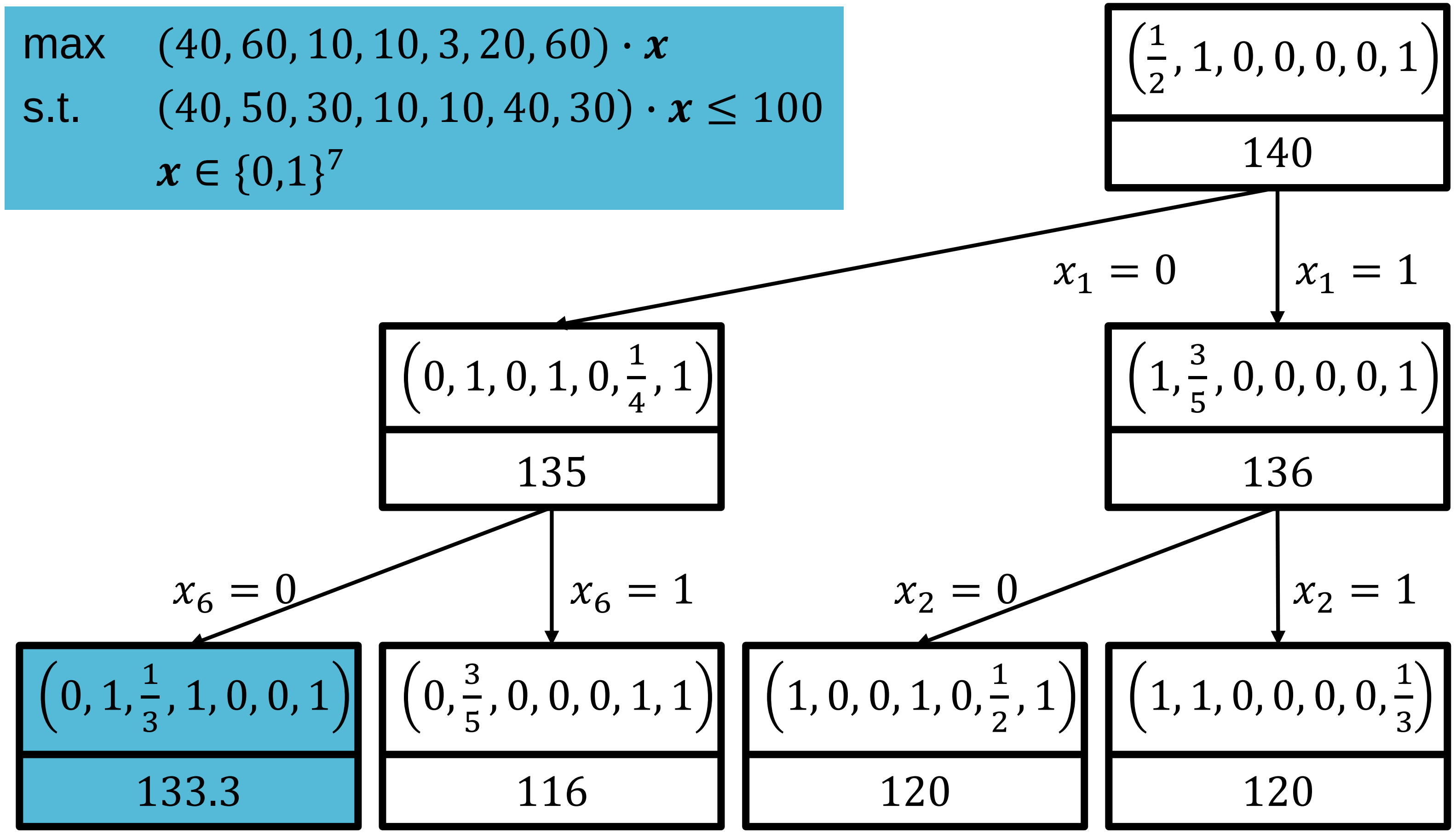
max	$(40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x}$
s.t.	$(40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100$
	$\boldsymbol{x} \in \{0,1\}^7$



B&B

1. **Choose leaf of tree**
2. Branch on a variable

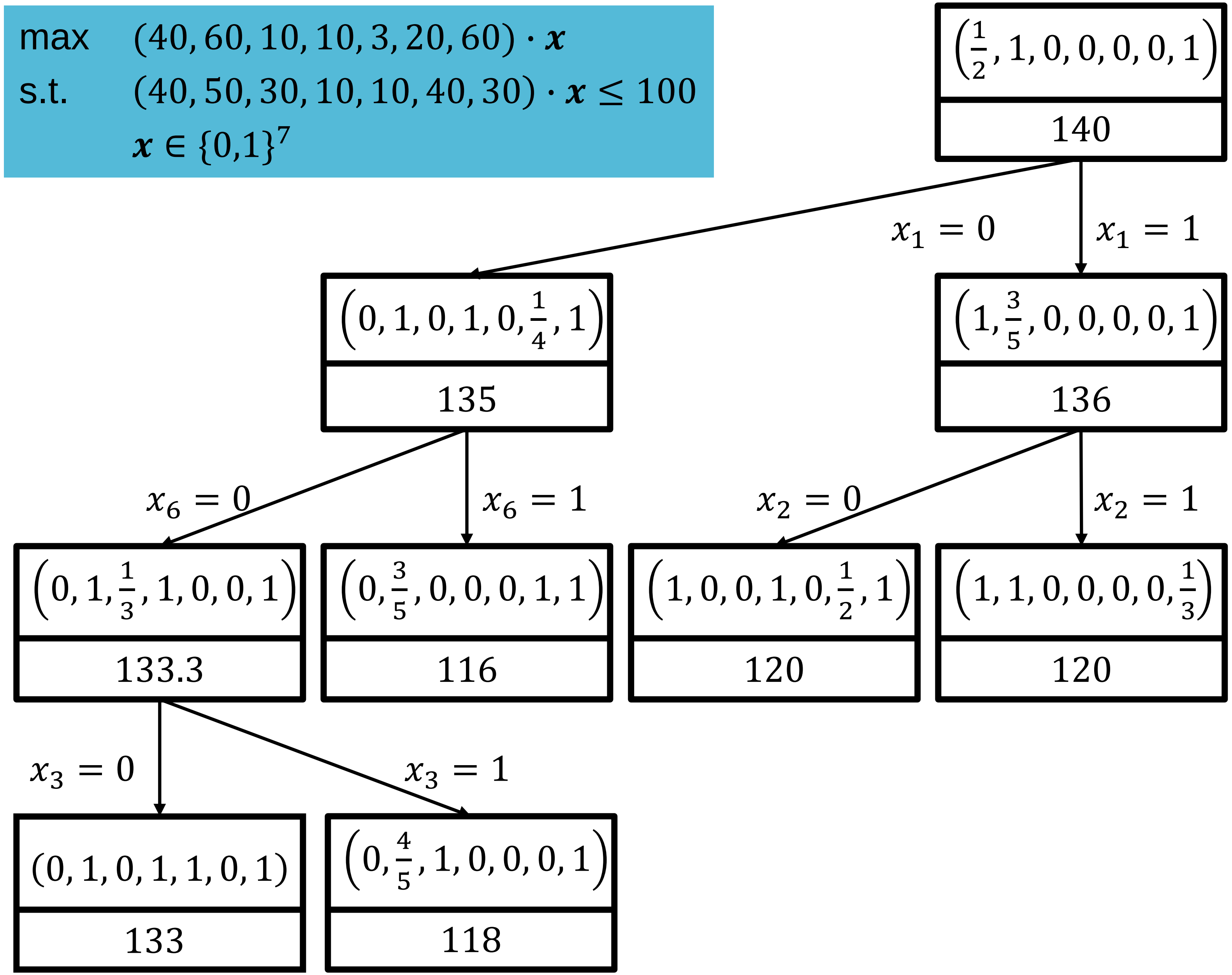
$$\begin{array}{ll} \text{max} & (40,60,10,10,3,20,60)\cdot \boldsymbol{x} \\ \text{s.t.} & (40,50,30,10,10,40,30)\cdot \boldsymbol{x} \le 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$



B&B

1. Choose leaf of tree
2. **Branch** on a variable

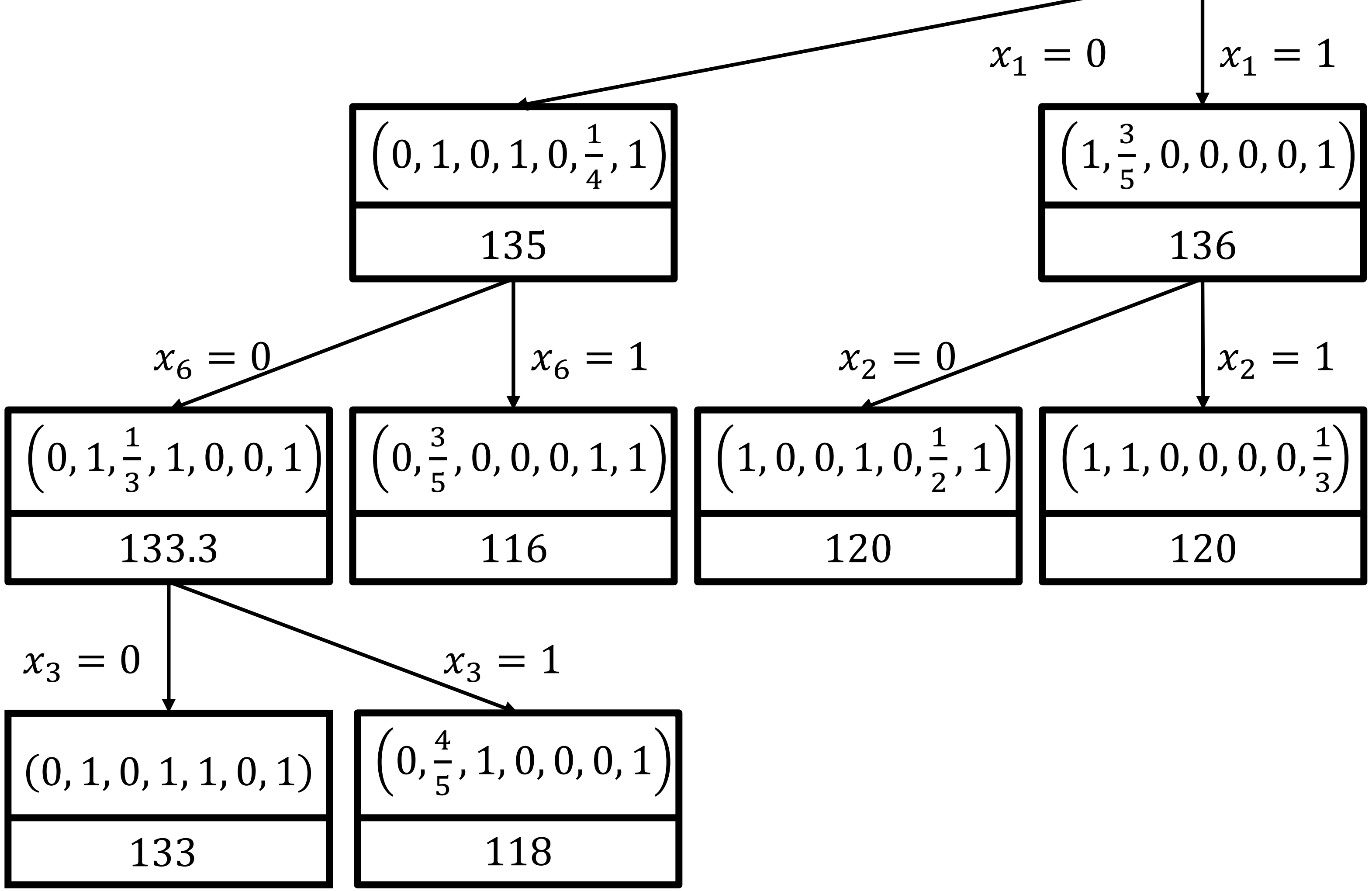
$$
\begin{aligned}
\text{max} \quad & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x} \\
\text{s.t.} \quad & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \le 100 \\
& \boldsymbol{x} \in \{0,1\}^7
\end{aligned}
$$



B&B

1. Choose leaf of tree
2. Branch on a variable
3. **Fathom** leaf if:
   i. LP relaxation solution is integral
   ii. LP relaxation is infeasible
   iii. LP relaxation solution isn't better than best-known integral solution

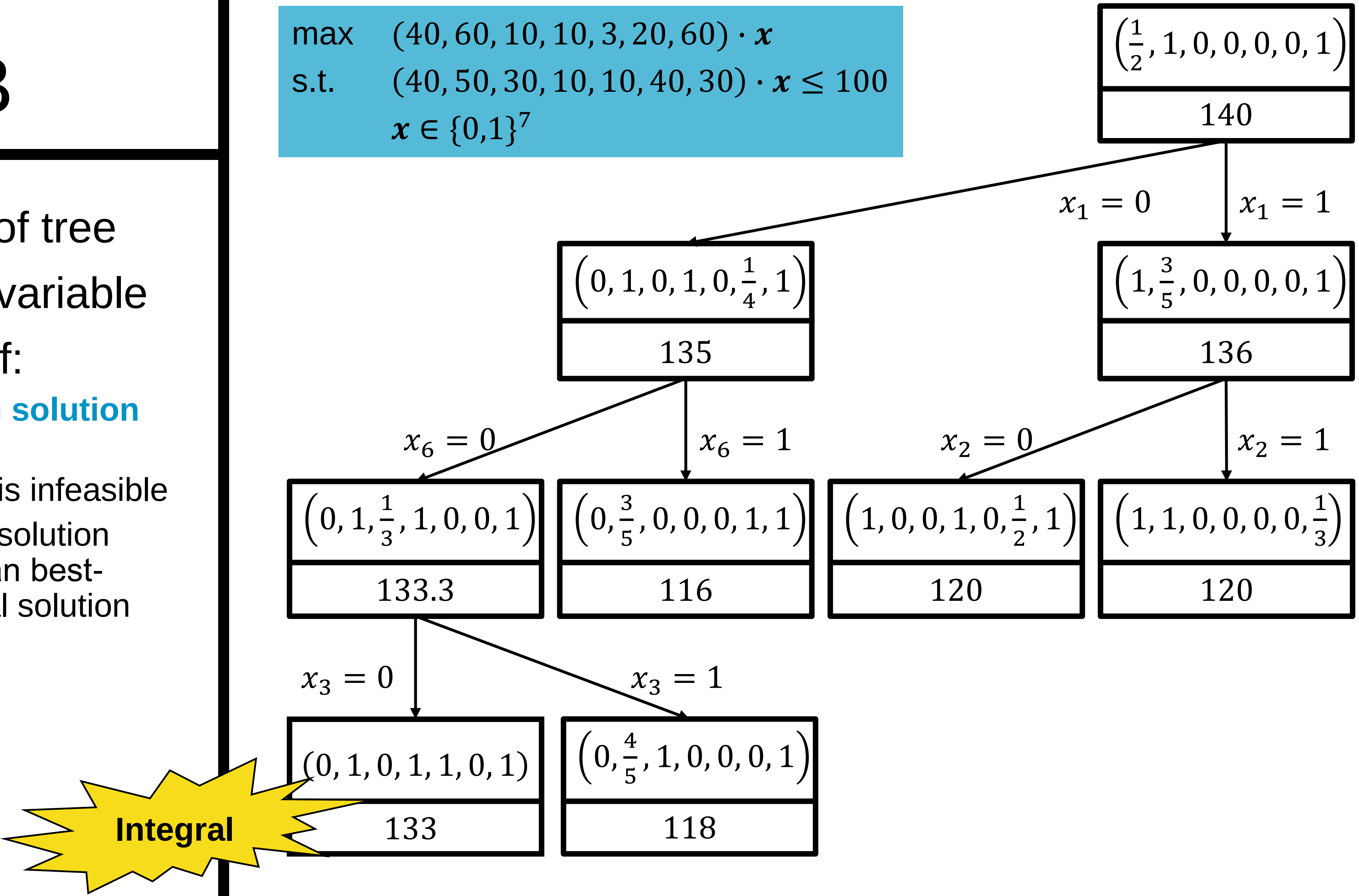
max	$(40,60,10,10,3,20,60)\cdot \boldsymbol{x}$
s.t.	$(40,50,30,10,10,40,30)\cdot \boldsymbol{x} \le 100$
	$\boldsymbol{x} \in \{0,1\}^7$



B&B

1. Choose leaf of tree
2. Branch on a variable
3. Fathom leaf if:
   i. **LP relaxation solution is integral**
   ii. LP relaxation is infeasible
   iii. LP relaxation solution isn't better than best-known integral solution

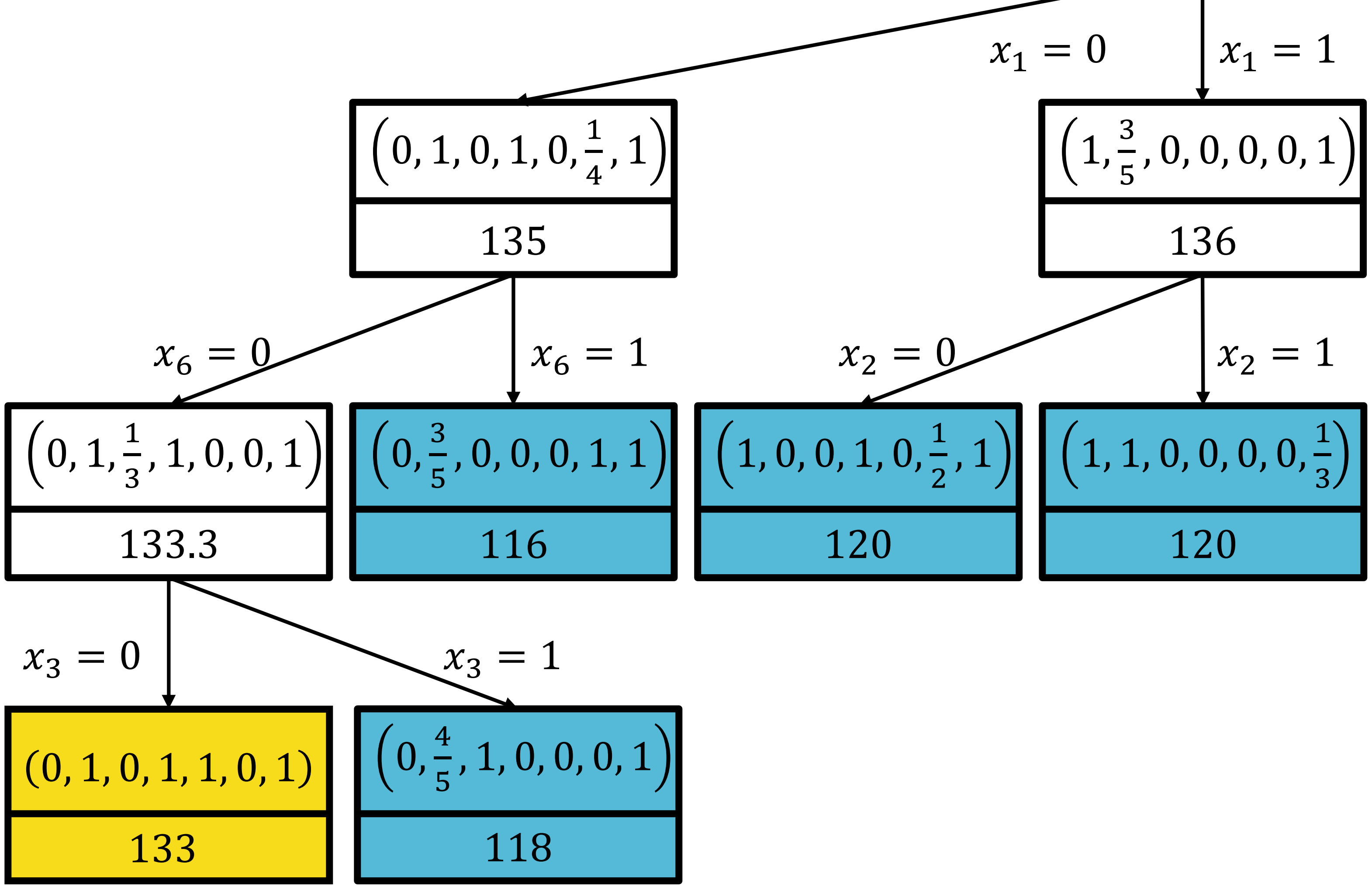
$$\begin{array}{ll} \text{max} & (40,60,10,10,3,20,60)\cdot \boldsymbol{x} \\ \text{s.t.} & (40,50,30,10,10,40,30)\cdot \boldsymbol{x} \le 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$



B&B

1. Choose leaf of tree
2. Branch on a variable
3. Fathom leaf if:
   i. LP relaxation solution is integral
   ii. LP relaxation is infeasible
   iii. **LP relaxation solution isn't better than best-known integral solution**

$$
\begin{array}{ll}
\text{max} & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x} \\
\text{s.t.} & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100 \\
 & \boldsymbol{x} \in \{0,1\}^7
\end{array}
$$

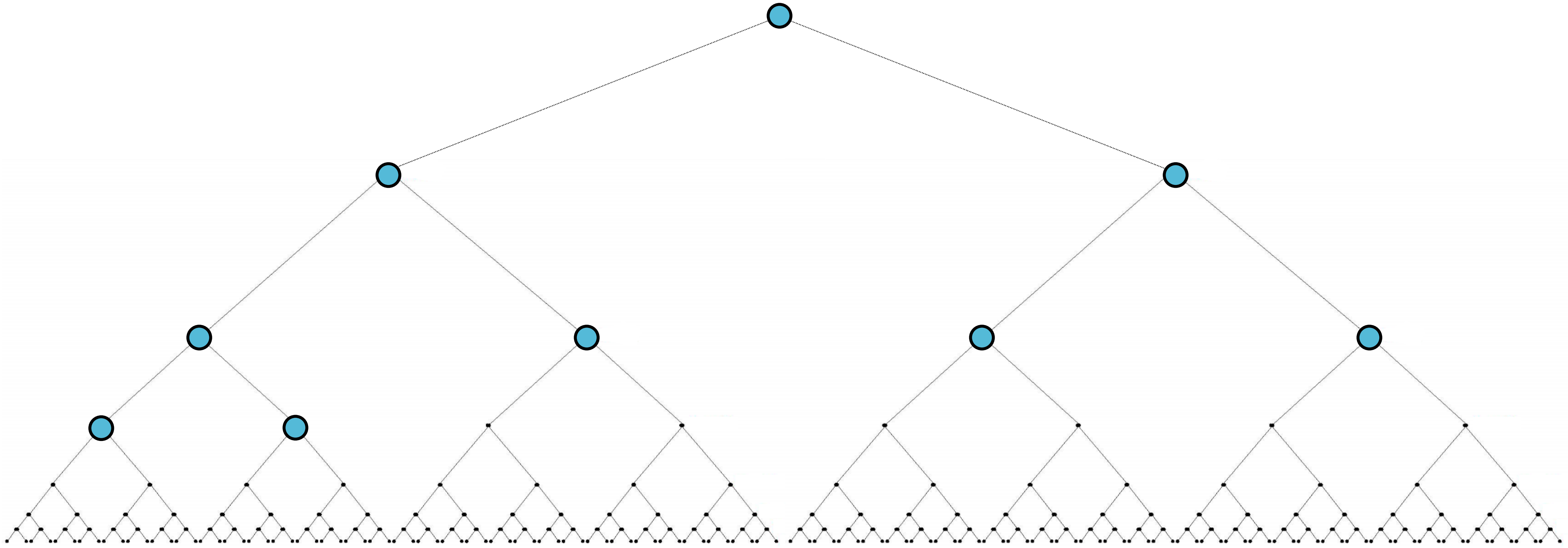


B&B

1. Choose leaf of tree
2. Branch on a variable
3. Fathom leaf if:
   i. LP relaxation solution is integral
   ii. LP relaxation is infeasible
   iii. LP relaxation solution isn't better than best-known integral solution

This talk: How to choose which variable?
(Assume every other aspect of B&B is fixed.)

Variable selection policies can have a huge effect on tree size



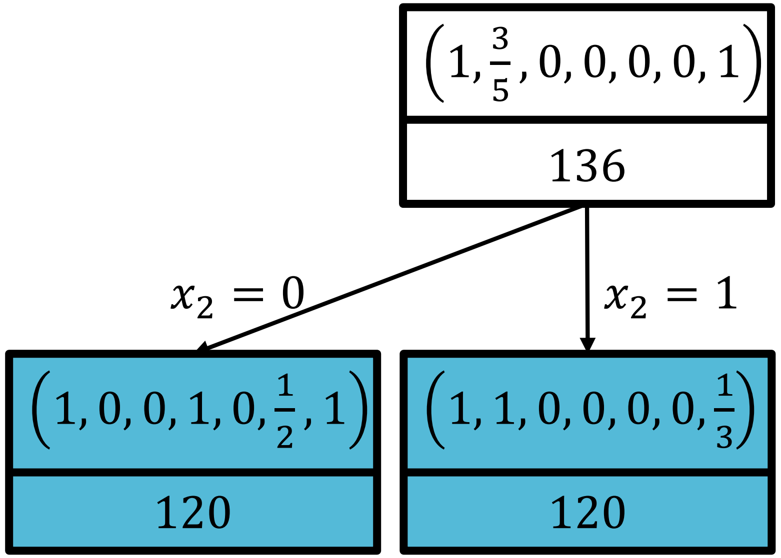
Outline

Variable selection policies (VSPs)

Score-based VSP:

At leaf $\boldsymbol{Q}$, branch on variable $\boldsymbol{x_i}$ maximizing

$$\textbf{score}(\boldsymbol{Q}, \boldsymbol{i})$$



Many options! Little known about which to use when

Variable selection policies

For an IP instance Q:

- Let c_Q be the objective value of its LP relaxation

Example.

$Q \rightarrow$

$$\begin{array}{ll} \text{max} & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x} \\ \text{s.t.} & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$

$$\left(\frac{1}{2}, 1, 0, 0, 0, 0, 1\right)$$

$140 \leftarrow c_Q$

Variable selection policies

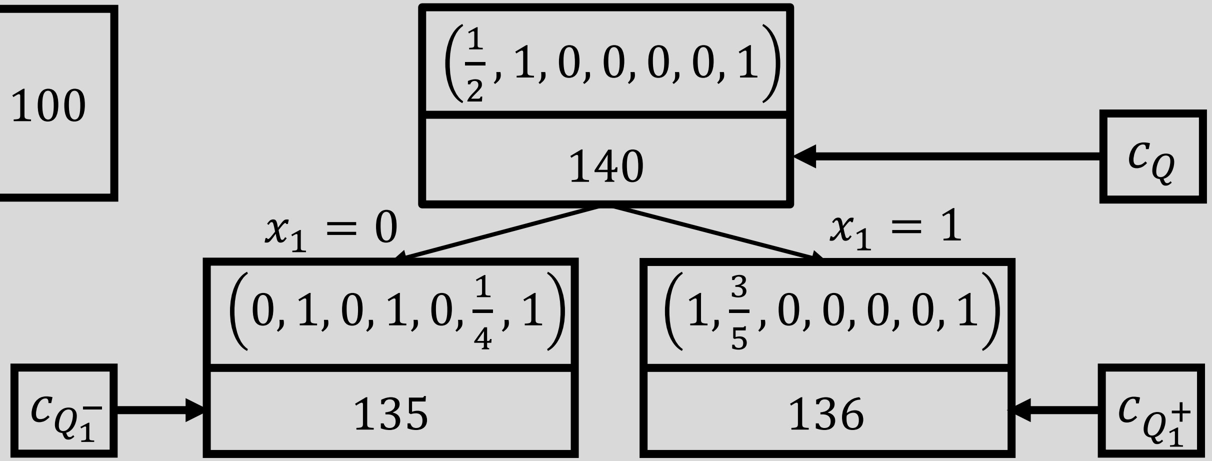
For an IP instance Q:

- Let c_Q be the objective value of its LP relaxation
- Let Q_i^- be Q with x_i set to 0, and let Q_i^+ be Q with x_i set to 1

Example.

Q

$$\begin{array}{ll} \text{max} & (40, 60, 10, 10, 3, 20, 60) \cdot \boldsymbol{x} \\ \text{s.t.} & (40, 50, 30, 10, 10, 40, 30) \cdot \boldsymbol{x} \leq 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$



Variable selection policies

The linear rule (parameterized by μ) [Linderoth & Savelsbergh, 1999]

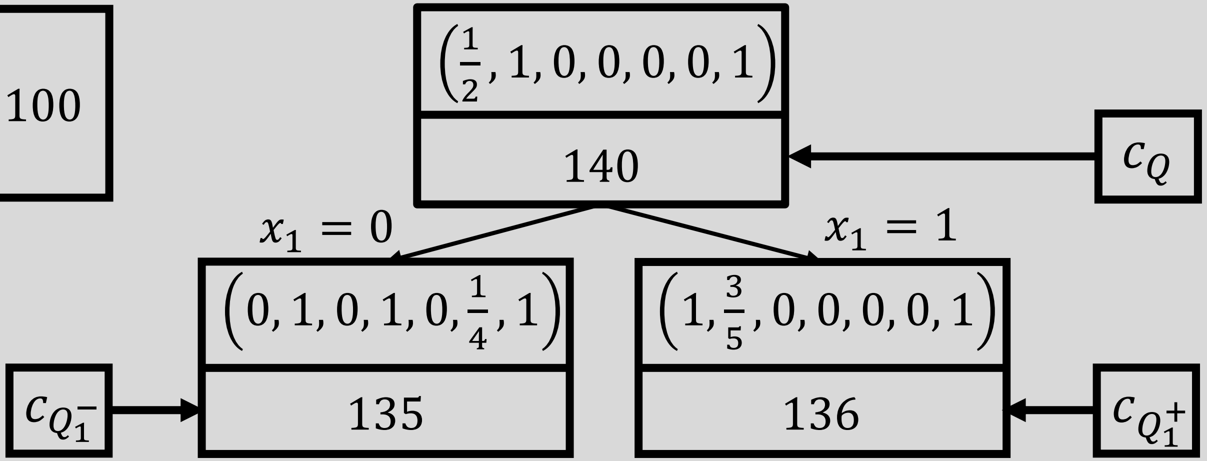
Branch on variable x_i maximizing:

$$\mathrm{score}(Q,i) = \mu \min\left\{c_Q - c_{Q_i^-}, c_Q - c_{Q_i^+}\right\} + (1-\mu)\max\left\{c_Q - c_{Q_i^-}, c_Q - c_{Q_i^+}\right\}$$

Example.

$Q \rightarrow$

$$\begin{array}{ll} \text{max} & (40,60,10,10,3,20,60)\cdot \boldsymbol{x} \\ \text{s.t.} & (40,50,30,10,10,40,30)\cdot \boldsymbol{x} \le 100 \\ & \boldsymbol{x} \in \{0,1\}^7 \end{array}$$



Variable selection policies

The linear rule (parameterized by μ) [Linderoth & Savelsbergh, 1999]
Branch on variable x_i maximizing: $\mathrm{score}(Q,i) = \mu \min\left\{c_Q - c_{Q_i^-}, c_Q - c_{Q_i^+}\right\} + (1-\mu)\max\left\{c_Q - c_{Q_i^-}, c_Q - c_{Q_i^+}\right\}$

The (simplified) **product rule** [Achterberg, 2009]
Branch on variable x_i maximizing: $\mathrm{score}(Q,i) = \left(c_Q - c_{Q_i^-}\right)\cdot\left(c_Q - c_{Q_i^+}\right)$

And many more…

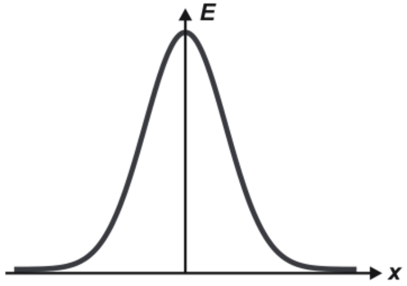
Variable selection policies

Given d scoring rules $\mathrm{score}_1, \ldots, \mathrm{score}_\mathrm{d}$.

Goal: Learn best convex combination $\mu_1 \mathrm{score}_1 + \cdots + \mu_d \mathrm{score}_\mathrm{d}$.

Our parameterized rule
Branch on variable x_i maximizing: $$\mathrm{score}(Q,i) = \mu_1 \mathrm{score}_1(Q,i) + \cdots + \mu_d \mathrm{score}_\mathrm{d}(Q,i)$$

Model



Application-Specific Distribution

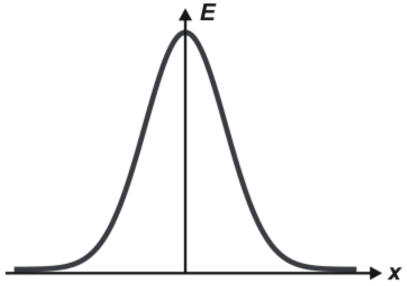
$$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}, \dots, \{A^{(m)}, \boldsymbol{b}^{(m)}, \boldsymbol{c}^{(m)}\}$$

Algorithm Designer

B&B parameters

How to use samples to find best B&B parameters for my domain?

Model



Application-Specific Distribution

$\{A^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{c}^{(1)}\}, \dots, \{A^{(m)}, \boldsymbol{b}^{(m)}, \boldsymbol{c}^{(m)}\}$

Algorithm Designer

$\mu_1, \dots, \mu_d$

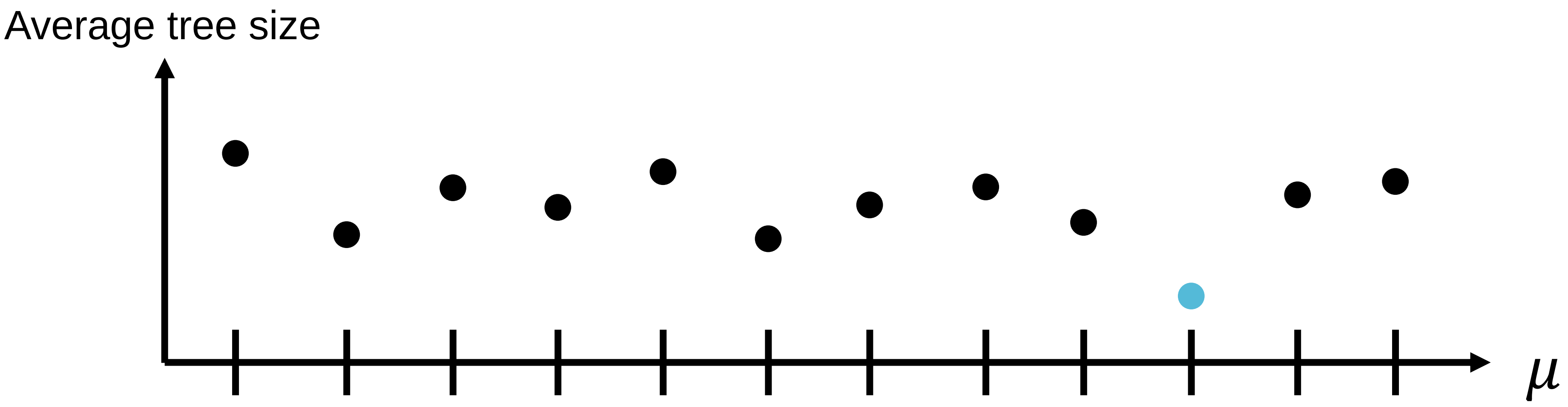
How to use samples to find best $\mu_1, \dots, \mu_d$ for my domain?

Outline

1. Introduction
2. Branch-and-Bound
3. Learning algorithms
   a. **First-try: Discretization**
   b. Our Approach
4. Experiments
5. Conclusion and Future Directions

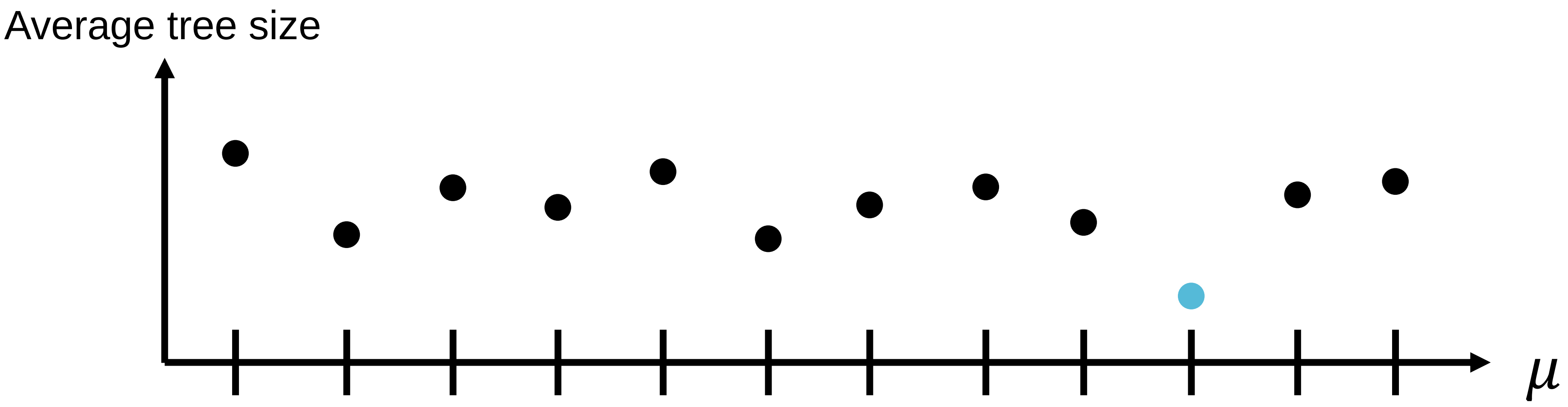
First try: Discretization

1. Discretize parameter space
2. Receive sample problems from unknown distribution
3. Find params in discretization with best average performance

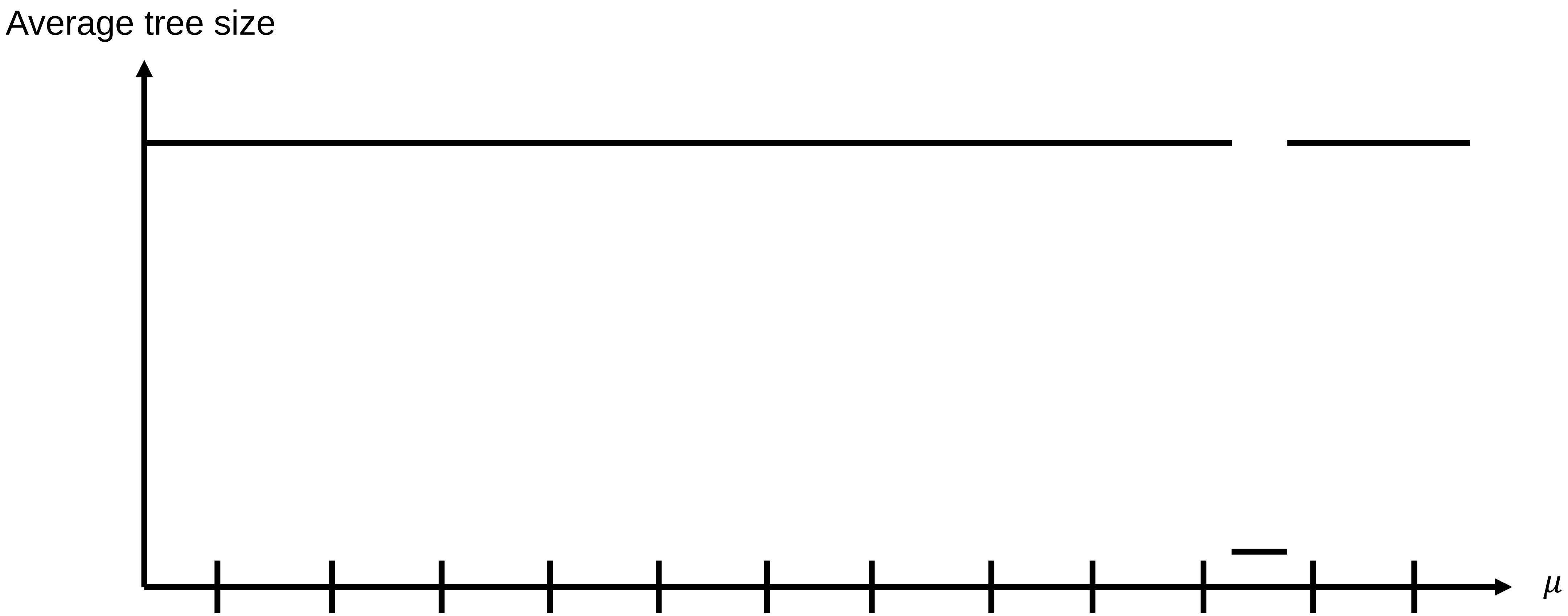


First try: Discretization

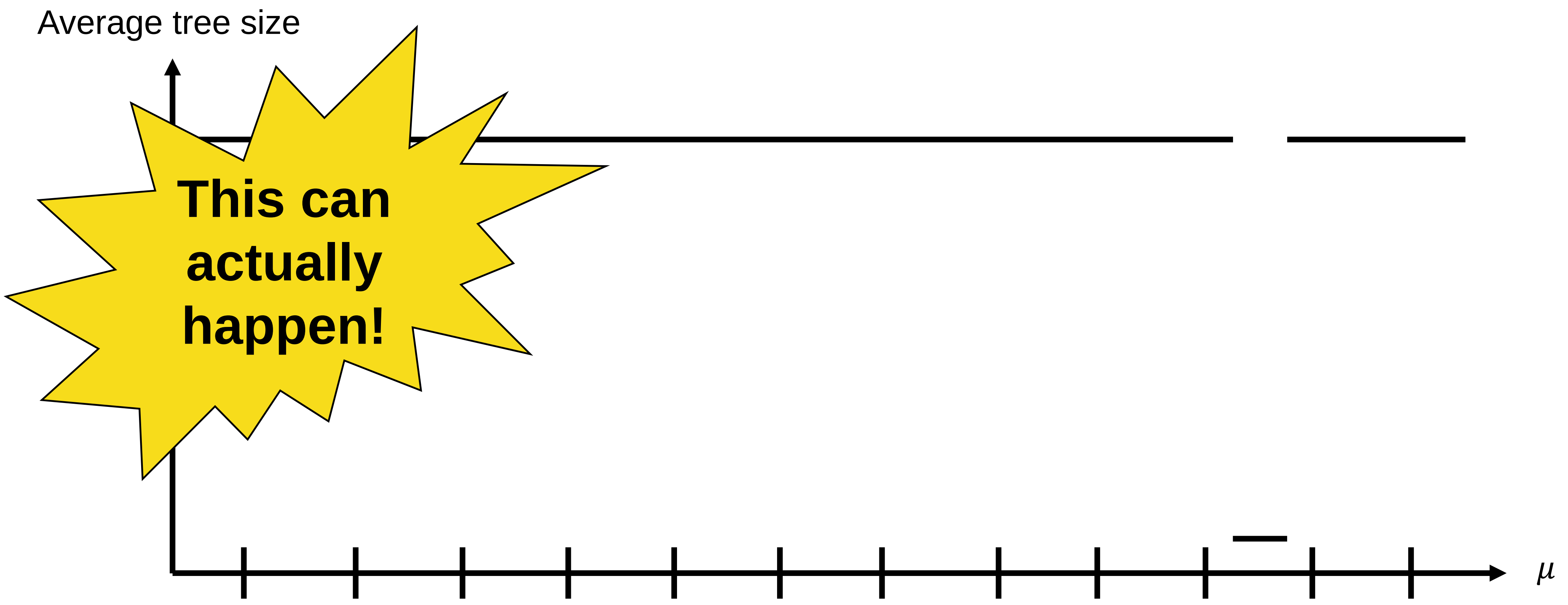
This has been prior work's approach [e.g., Achterberg (2009)].



Discretization gone wrong



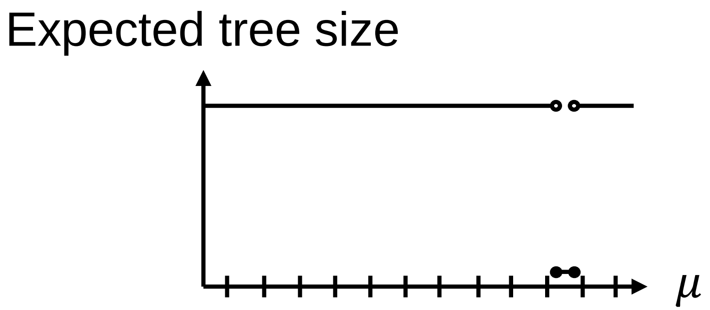
Discretization gone wrong



Discretization gone wrong

Theorem [informal]. For any discretization:

Exists problem instance distribution $\mathcal{D}$ inducing this behavior



Proof ideas:

$\mathcal{D}$'s support consists of infeasible IPs with “easy out” variables

B&B takes exponential time unless branches on “easy out” variables

B&B only finds “easy outs” if uses parameters from specific range

Outline

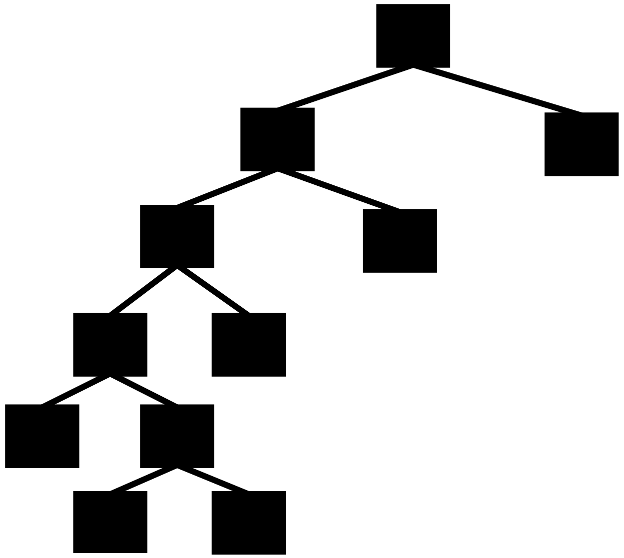
1. Introduction
2. Branch-and-Bound
3. Learning algorithms
   a. First-try: Discretization
   b. Our Approach
      i. **Single-parameter settings**
      ii. Multi-parameter settings
4. Experiments
5. Conclusion and Future Directions

Simple assumption

Exists κ upper bounding the size of largest tree willing to build

Common assumption, e.g.:

- Hutter, Hoos, Leyton-Brown, Stützle, JAIR’09
- Kleinberg, Leyton-Brown, Lucier, IJCAI’17



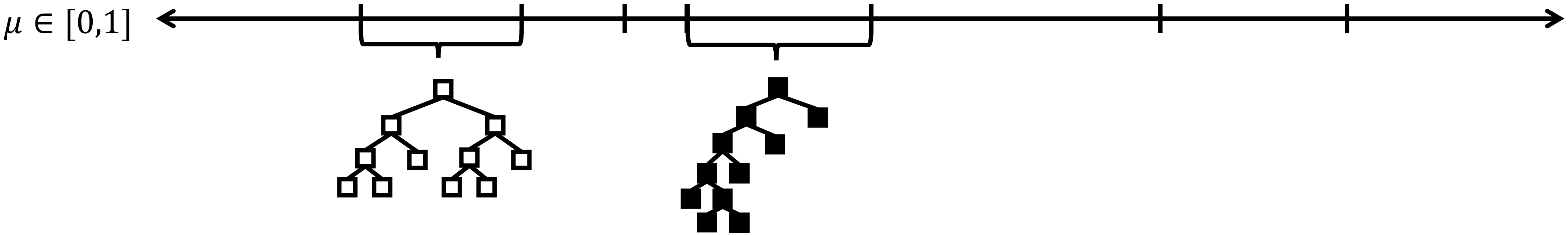
Useful lemma

Much smaller in our experiments!

Lemma: For any two scoring rules and any IP Q,

$O\left((\# \text{ variables})^{\kappa+2}\right)$ intervals partition $[0,1]$ such that:

For any interval $[a, b]$, B&B builds same tree across all $\mu \in [a, b]$

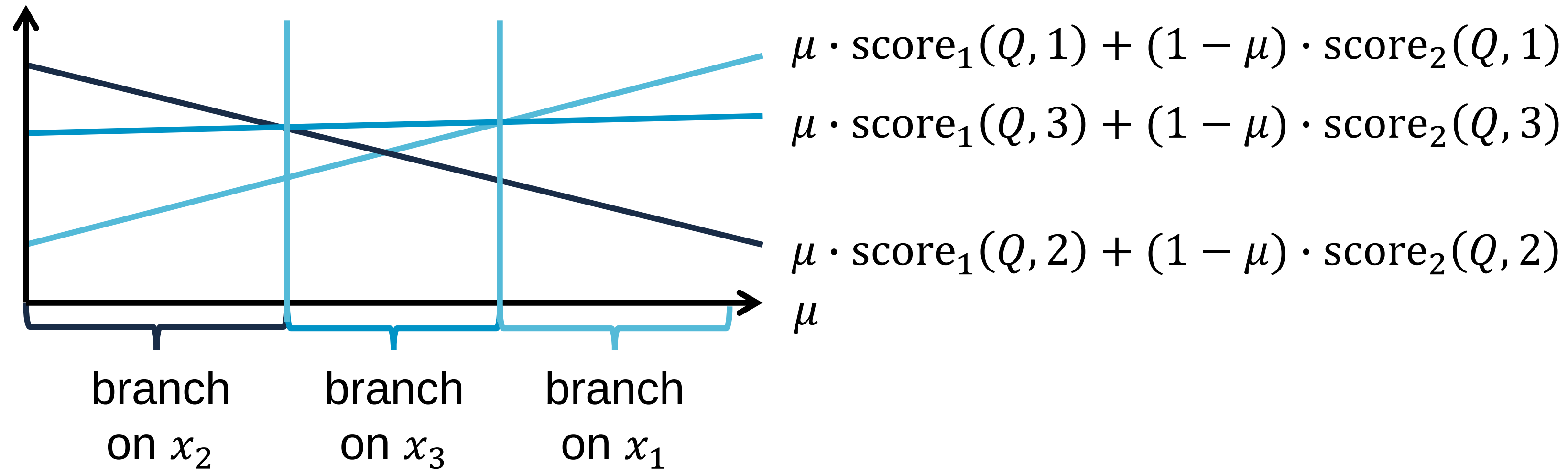


Useful lemma

Lemma: For any two scoring rules and any IP Q,

$O\big((\#\text{ variables})^{\kappa+2}\big)$ intervals partition $[0,1]$ such that:

For any interval $[a,b]$, B&B builds same tree across all $\mu \in [a,b]$

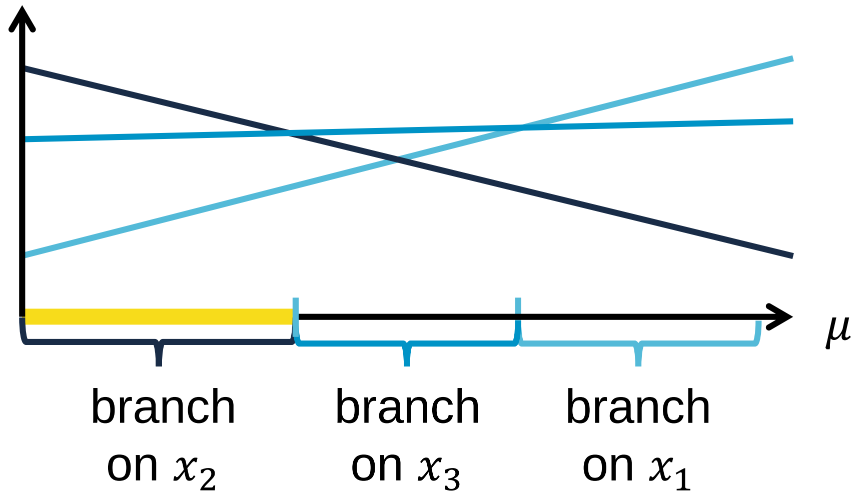


Useful lemma

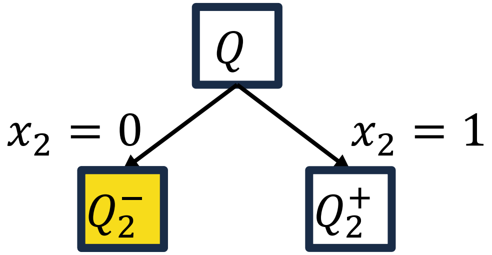
Lemma: For any two scoring rules and any IP Q,

$O\big((\#\text{ variables})^{\kappa+2}\big)$ intervals partition $[0,1]$ such that:

For any interval $[a, b]$, B&B builds same tree across all $\mu \in [a, b]$

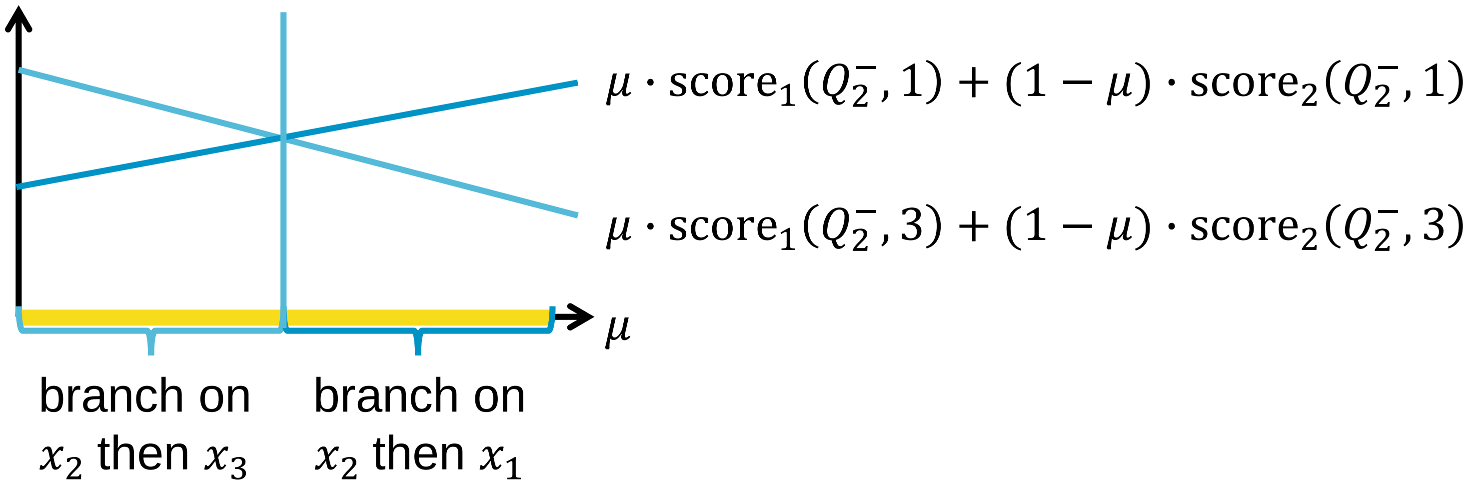


Any μ in yellow interval:

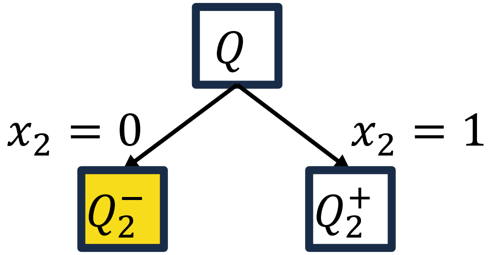


Useful lemma

Lemma: For any two scoring rules and any IP Q,
$O\big((\#\text{ variables})^{\kappa+2}\big)$ intervals partition $[0,1]$ such that:
For any interval $[a, b]$, B&B builds same tree across all $\mu \in [a, b]$

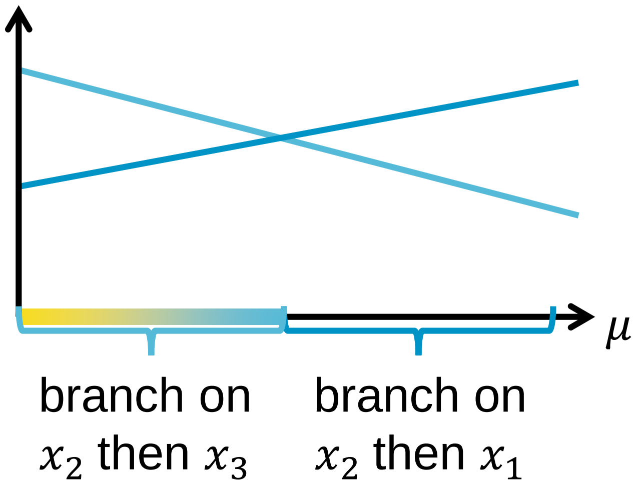


Any μ in yellow interval:

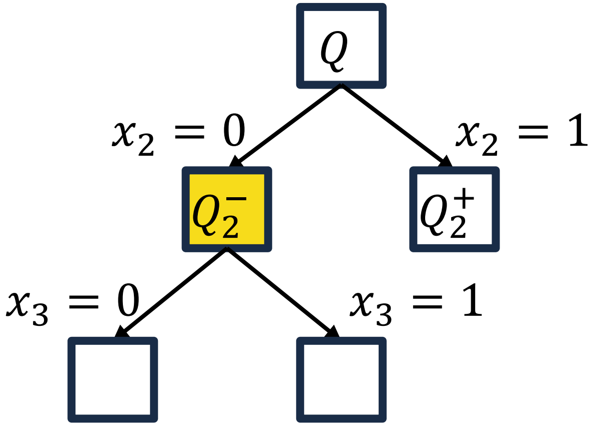


Useful lemma

Lemma: For any two scoring rules and any IP Q,
$O\left((\text{\# variables})^{\kappa+2}\right)$ intervals partition $[0,1]$ such that:
For any interval $[a, b]$, B&B builds same tree across all $\mu \in [a, b]$

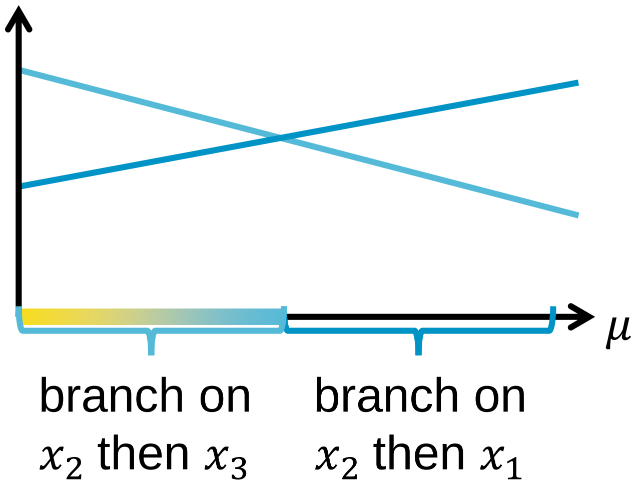


Any μ in blue-yellow interval:



Useful lemma

Lemma: For any two scoring rules and any IP Q,
$O\left((\#\ \text{variables})^{\kappa+2}\right)$ intervals partition $[0,1]$ such that:
For any interval $[a, b]$, B&B builds same tree across all $\mu \in [a, b]$



Proof idea.

- Continue dividing [0,1] into intervals s.t.:
  In each interval, var. selection order fixed
- Can subdivide only finite number of times
- Proof follows by induction on tree depth

Learning algorithm

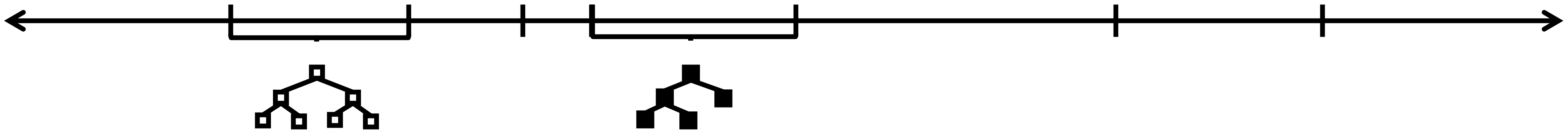
Input: Set of IPs sampled from a distribution $\mathcal{D}$

For each IP, set $\mu = 0$. While $\mu < 1$:

1. Run B&B using $\mu \cdot \mathrm{score}_1 + (1-\mu) \cdot \mathrm{score}_2$, resulting in tree $\mathcal{T}$
2. Find interval $[\mu, \mu')$ where if B&B is run using the scoring rule
$$\mu'' \cdot \mathrm{score}_1 + (1-\mu'') \cdot \mathrm{score}_2$$
for any $\mu'' \in [\mu, \mu')$, B&B will build tree $\mathcal{T}$ (takes a little bookkeeping)
3. Set $\mu = \mu'$

Return: Any $\hat{\mu}$ from the interval minimizing average tree size

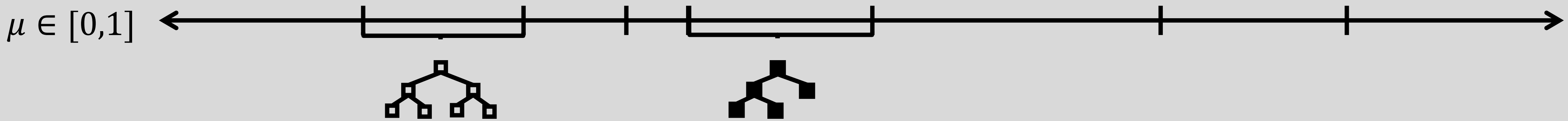
$\mu \in [0,1]$



Learning algorithm guarantees

Let $\hat{\mu}$ be algorithm's output given $\tilde{O}\left(\frac{\kappa^3}{\varepsilon^2}\ln(\#\text{variables})\right)$ samples.

W.h.p., $\mathbb{E}_{Q\sim\mathcal{D}}[\text{tree-size}(Q,\hat{\mu})] - \min_{\mu\in[0,1]} \mathbb{E}_{Q\sim\mathcal{D}}[\text{tree−size}(Q,\mu)] < \varepsilon$



Proof intuition: Bound algorithm class's intrinsic complexity (IC)

- Lemma bounds the number of "truly different" parameters
- Parameters that are "the same" come from a simple set

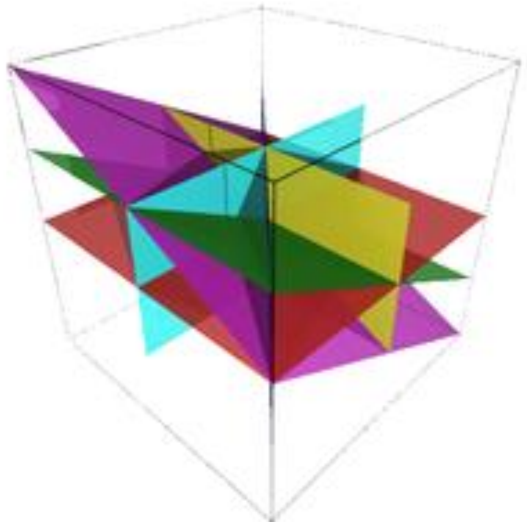
Learning theory allows us to translate IC to sample complexity

Outline

1. Introduction
2. Branch-and-Bound
3. Learning algorithms
    a. First-try: Discretization
    b. Our Approach
        i. Single-parameter settings
        ii. **Multi-parameter settings**
4. Experiments
5. Conclusion and Future Directions

Useful lemma: higher dimensions

Lemma: For any d scoring rules and any IP,
a set $\mathcal{H}$ of $O\left((\#\text{ variables})^{\kappa+2}\right)$ hyperplanes partitions $[0,1]^d$ s.t.:
For any connected component R of $[0,1]^d \setminus \mathcal{H}$,
B&B builds the same tree across all $\boldsymbol{\mu} \in R$



Learning-theoretic guarantees

Fix d scoring rules and draw samples $Q_1, \ldots, Q_N \sim \mathcal{D}$

If $N = \tilde{O}\left(\frac{\kappa^3}{\varepsilon^2} \ln(d \cdot \#\text{variables})\right)$, then w.h.p., for all $\boldsymbol{\mu} \in [0,1]^d$,

$$\left| \frac{1}{N} \sum_{i=1}^{N} \text{tree-size}(Q_i, \boldsymbol{\mu}) - \mathbb{E}_{Q \sim \mathcal{D}}[\text{tree-size}(Q, \boldsymbol{\mu})] \right| < \varepsilon$$

Average tree size generalizes to expected tree size

Outline

1. Introduction
2. Branch-and-Bound
3. Learning algorithms
4. **Experiments**
5. Conclusion and Future Directions

Experiments: Tuning the linear rule

Let: $\text{score}_1(Q,i) = \min\left\{c_Q - c_{Q_i^-}, c_Q - c_{Q_i^+}\right\}$

$\text{score}_2(Q,i) = \max\left\{c_Q - c_{Q_i^-}, c_Q - c_{Q_i^+}\right\}$

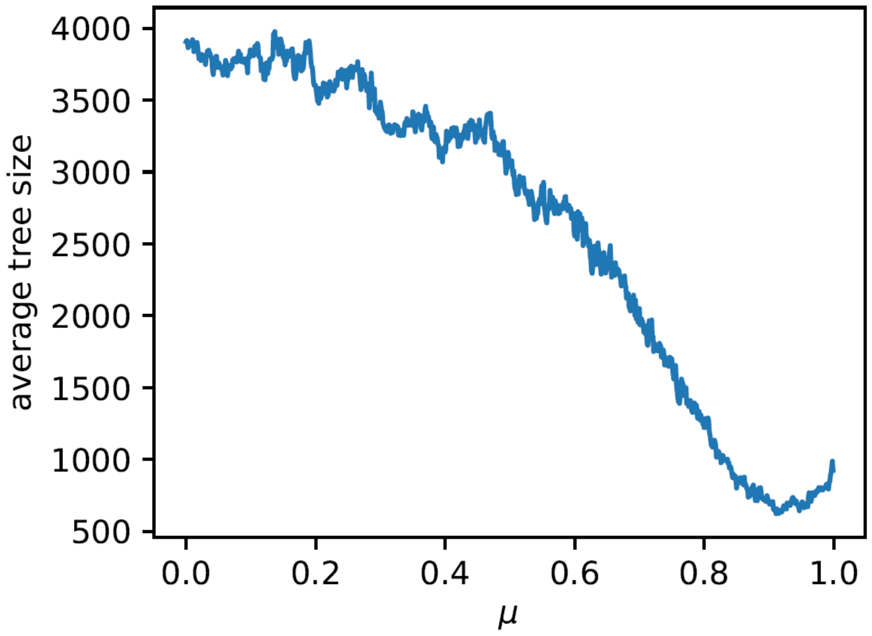
This is the linear rule [Linderoth & Savelsbergh, 1999]

Our parameterized rule

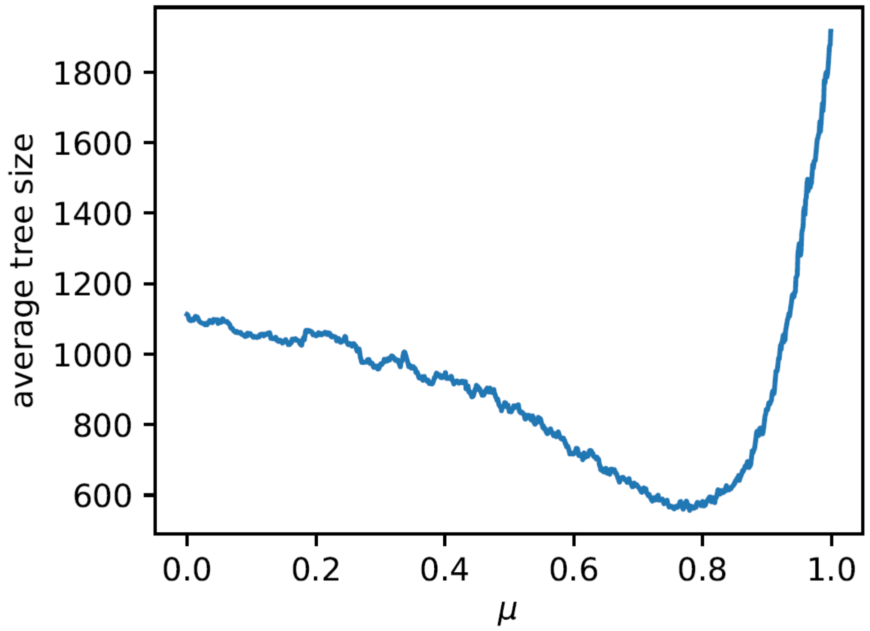
Branch on variable x_i maximizing:

$$\text{score}(Q,i) = \mu \cdot \text{score}_1(Q,i) + (1-\mu) \cdot \text{score}_2(Q,i)$$

Experiments: Combinatorial auctions



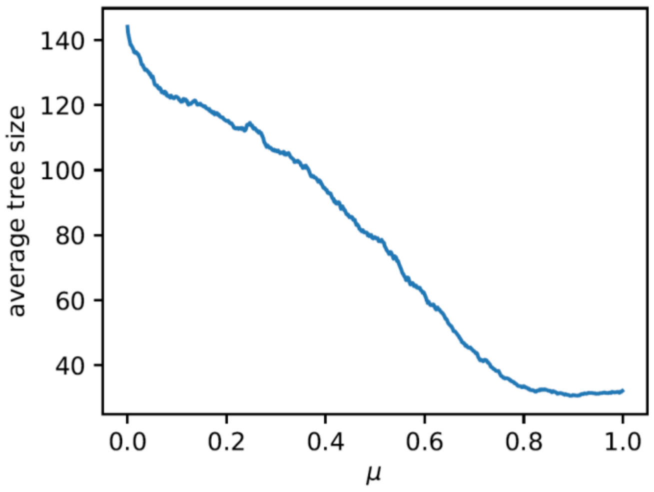
“Regions” generator:
400 bids, 200 goods, 100 instances



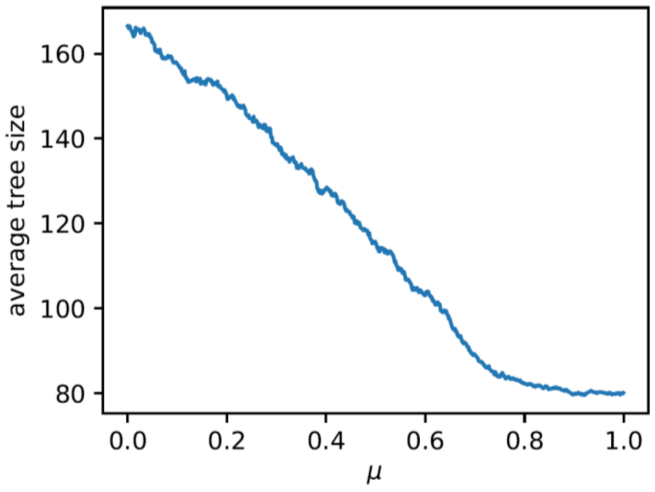
“Arbitrary” generator:
200 bids, 100 goods, 100 instances

Leyton-Brown, Pearson, and Shoham. Towards a universal test suite for combinatorial auction algorithms. In Proceedings of the Conference on Electronic Commerce (EC), 2000.

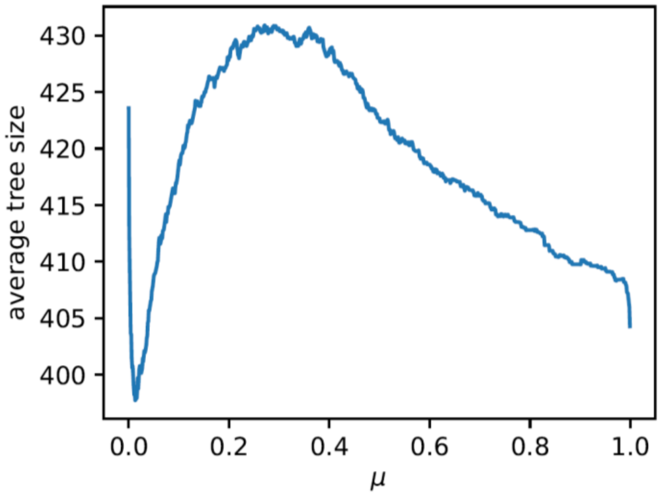
Additional experiments



Clustering:
5 clusters, 35 nodes,
500 instances



Facility location:
70 facilities, 70 customers,
500 instances



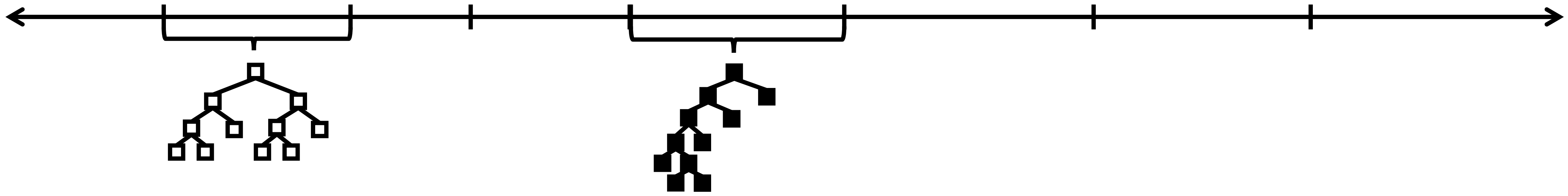
Agnostically learning linear separators:
50 points in $\mathbb{R}^2$,
500 instances

Outline

1. Introduction
2. Branch-and-Bound
3. Learning algorithms
4. Experiments
5. **Conclusion and Future Directions**

Conclusion

- Study B&B, a widely-used algorithm for combinatorial problems
- Show how to use ML to weight variable selection rules
  - First sample complexity bounds for tree search algorithm configuration
    - Unlike prior work [Khalil et al. '16; Alvarez et al. '17], which is purely empirical
- Empirically show our approach can dramatically shrink tree size
  - We prove this improvement can even be exponential
- Theory applies to other tree search algos, e.g., for solving CSPs



Future directions

How can we train faster?

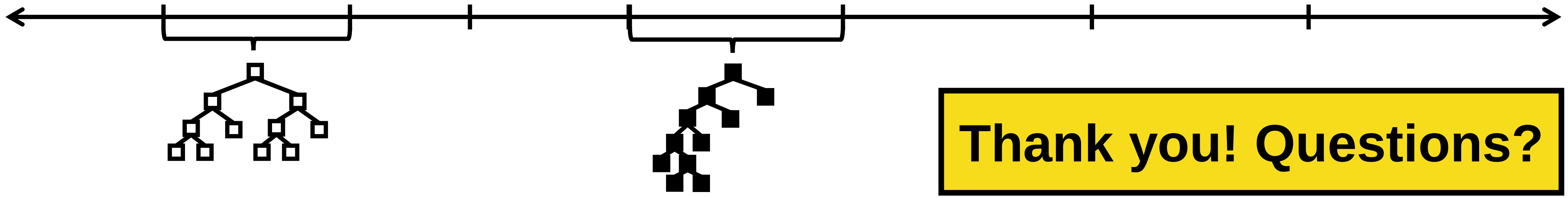
- Don’t want to build every tree B&B will make for every training instance
- Train on small IPs and then apply the learned policies on large IPs?

Other tree-building applications can we apply our techniques to?

- E.g., building decision trees and taxonomies

How can we attack other learning problems in B&B?

- E.g., node-selection policies



Thank you! Questions?