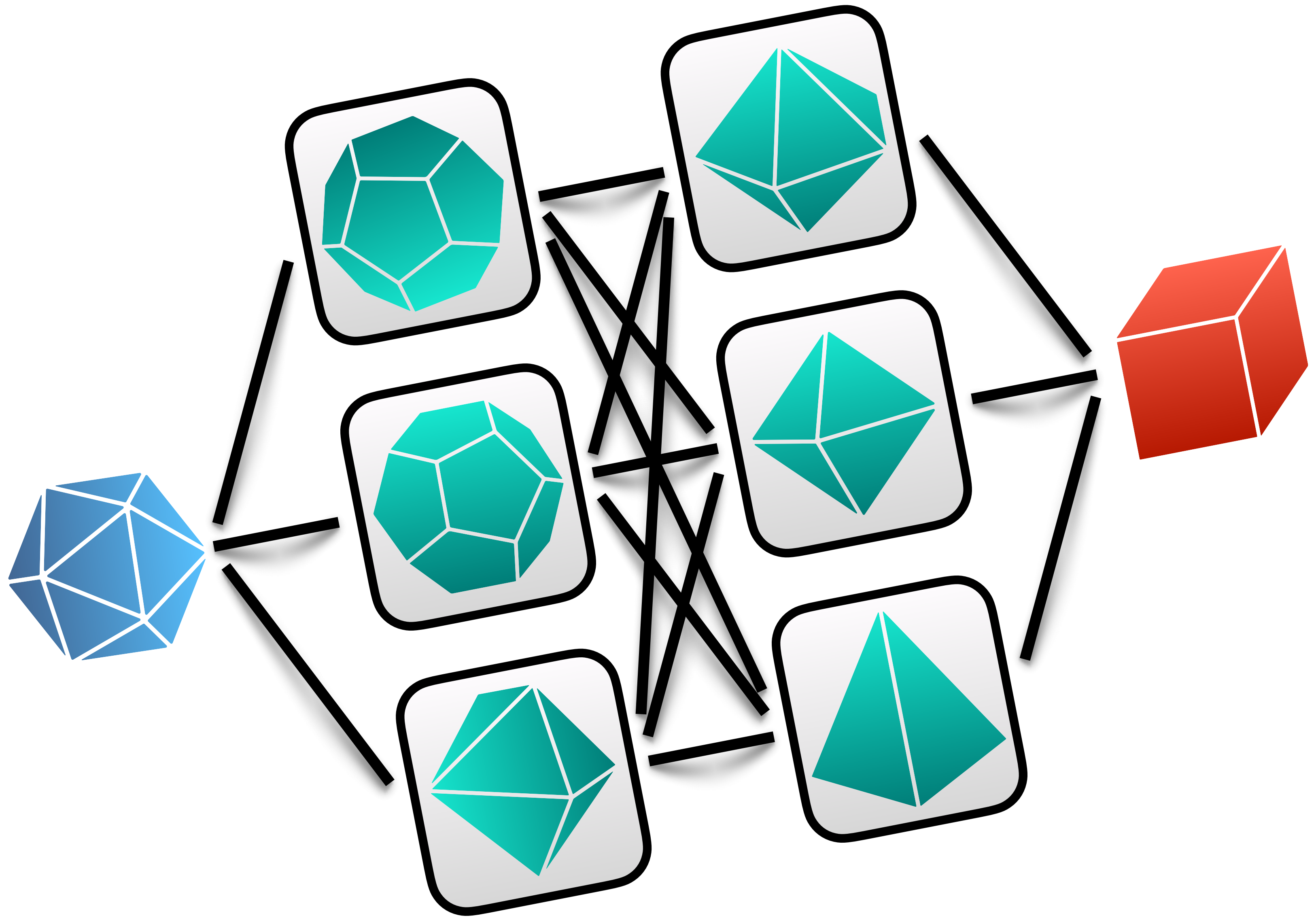
Did I forget to hit record? Please remind me!



Learning in Exact Solvers

MIE1666: Machine Learning for Mathematical Optimization

Elias B. Khalil — 18/10/21

UNIVERSITY OF TORONTO

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \text{ s.t. } Ax \le b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1. Select Node
2. Solve LP Relaxation
3. Prune?
4. Add Cuts
5. Run Heuristics
6. Branch

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 Select Node

2 Solve LP Relaxation

3 Prune?

4 Add Cuts

5 Run Heuristics

6 Branch

N_0

Branch & Bound for Integer Optimization

Land & Doig, 1960

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \cancel{\{0,1\}}^n \; [0,1]^n$$

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Solve LP Relaxation
→ Lower Bound on OPT

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \cancel{\{0,1\}}^n \quad [0,1]^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Solve LP Relaxation

→ Lower Bound on OPT

worse than best solution?

Prune!

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**

N_0

Add Cuts:
Tightening Constraints

Branch & Bound for Integer Optimization

LP-based

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

Land & Doig, 1960

Repeat:

1 Select Node

2 Solve LP Relaxation

3 Prune?

4 Add Cuts

5 **Run Heuristics**

6 Branch

N_0

Heuristic A
Heuristic B
Heuristic C

Feasible solution?

Update Best Solution

Branch & Bound for Integer Optimization

LP-based $\min_x c^T x$ **s.t.** $Ax \le b, x \in \{0,1\}^n$

Land & Doig, 1960

Repeat:

1 **Select Node**

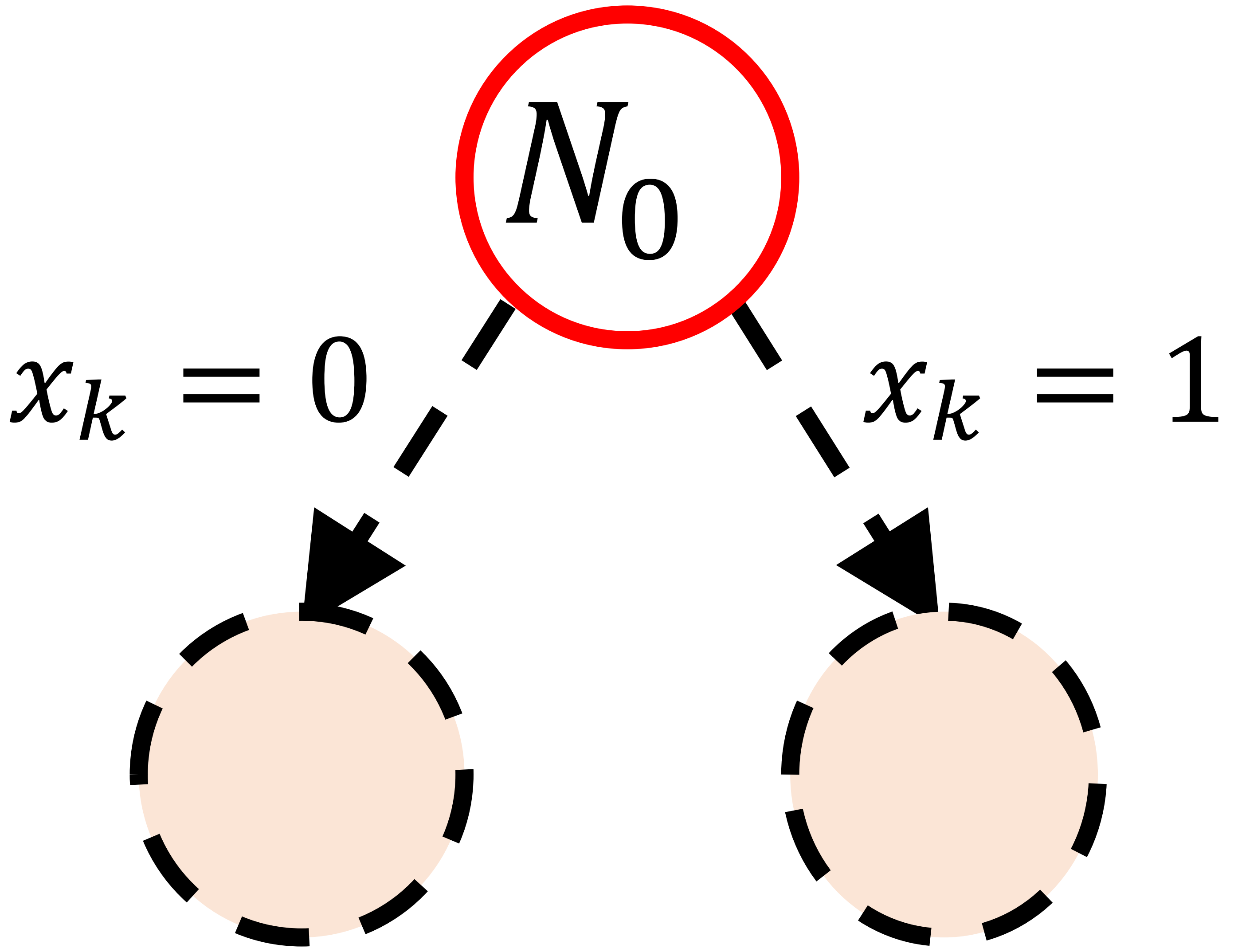
2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**



x_1?
x_2?
...
x_n?

Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

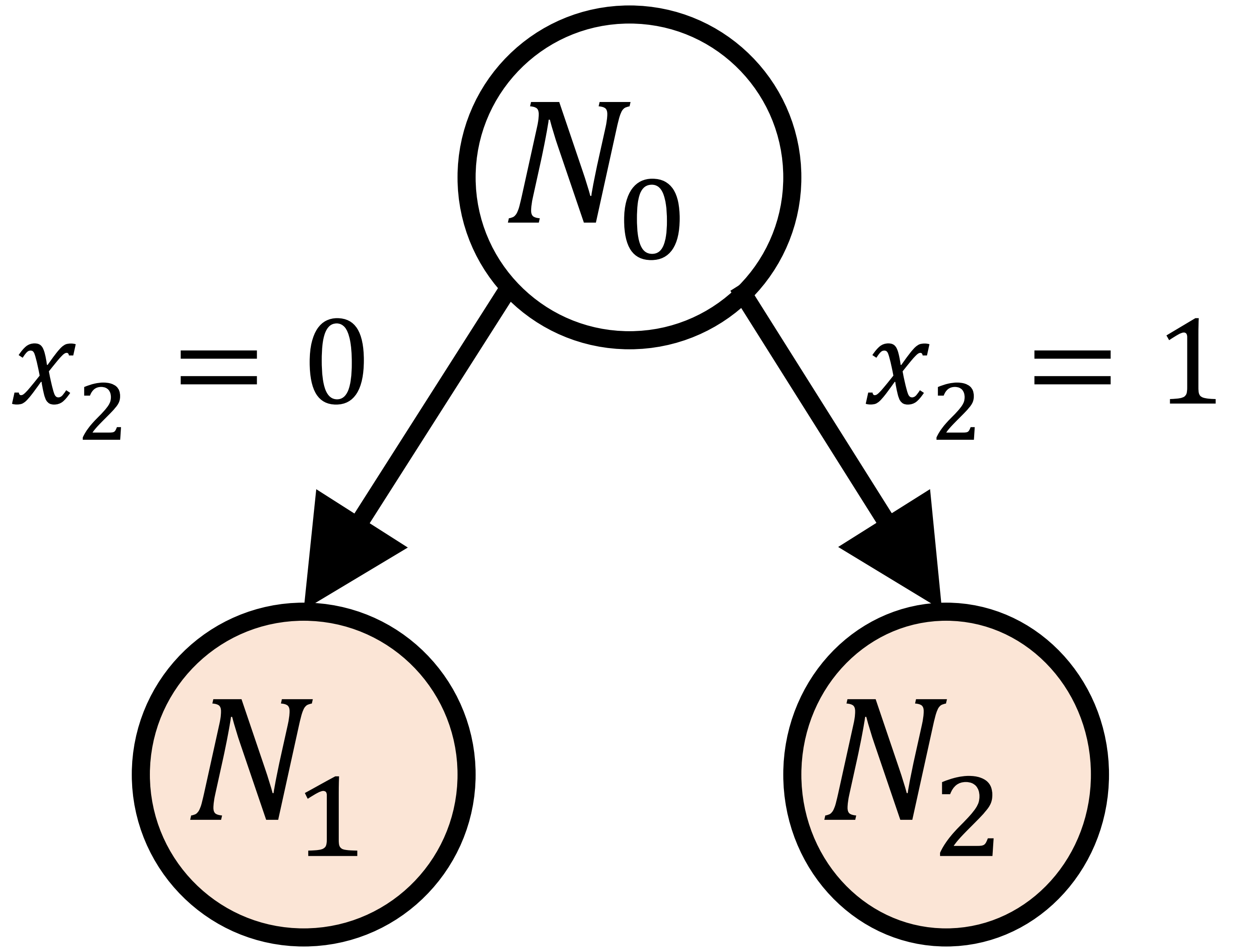
2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**



x_1?

$\boxed{x_2}$

...

x_n?

Branch & Bound for Integer Optimization

LP-based

$$\min_{x} c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

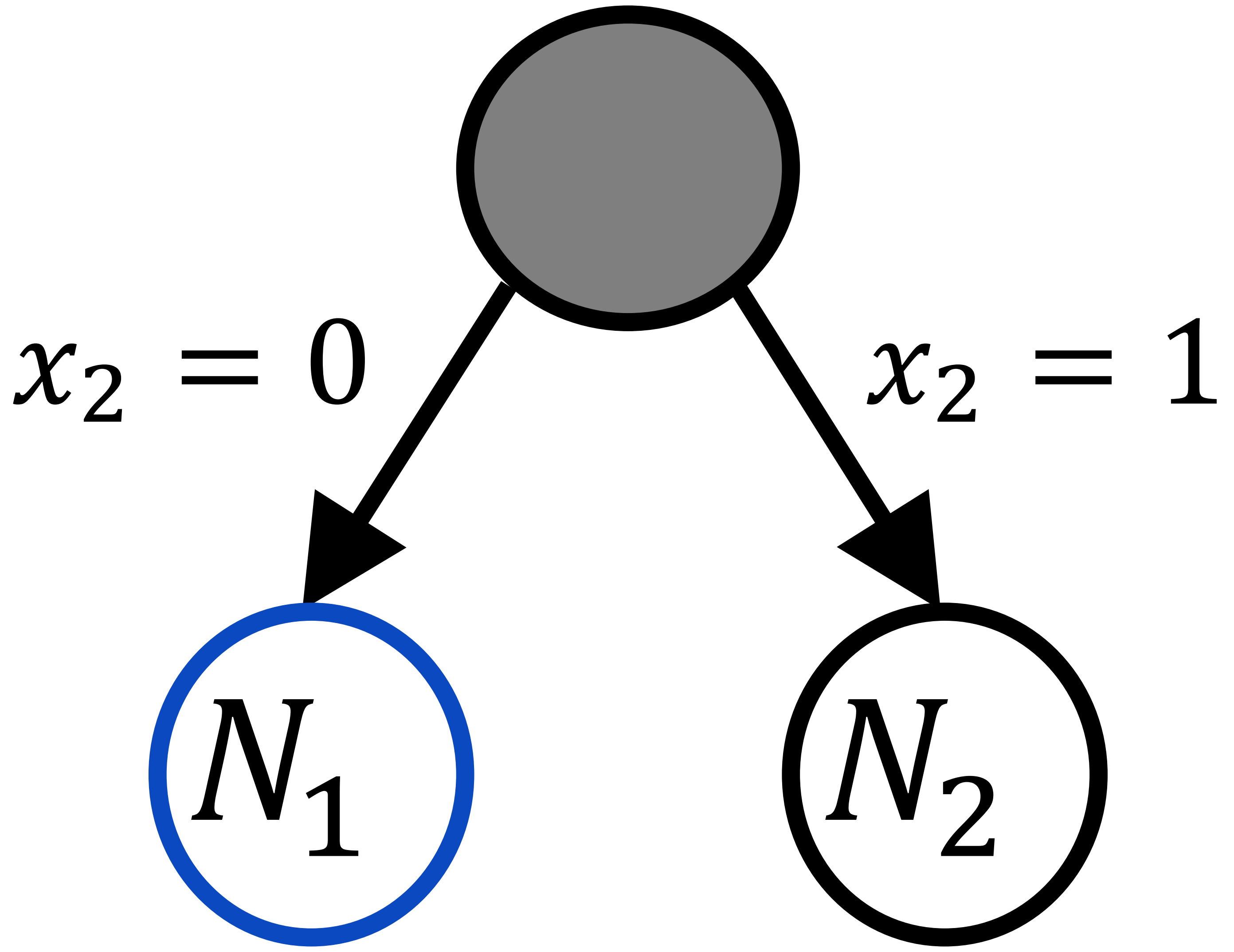
2 **Solve LP Relaxation**

3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**



Branch & Bound for Integer Optimization

LP-based

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \{0,1\}^n$$

Land & Doig, 1960

Repeat:

1 **Select Node**

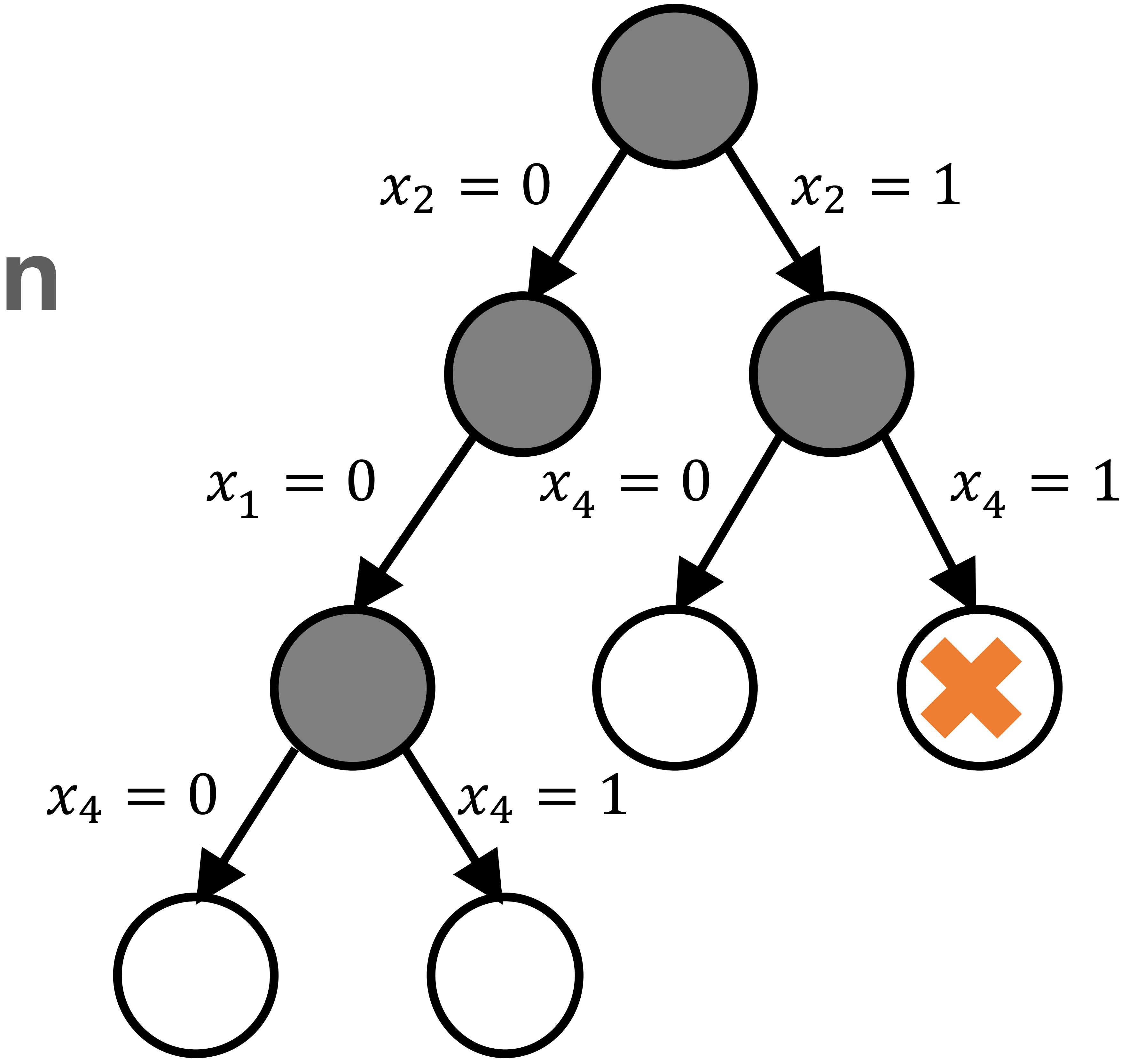
2 **Solve LP Relaxation**

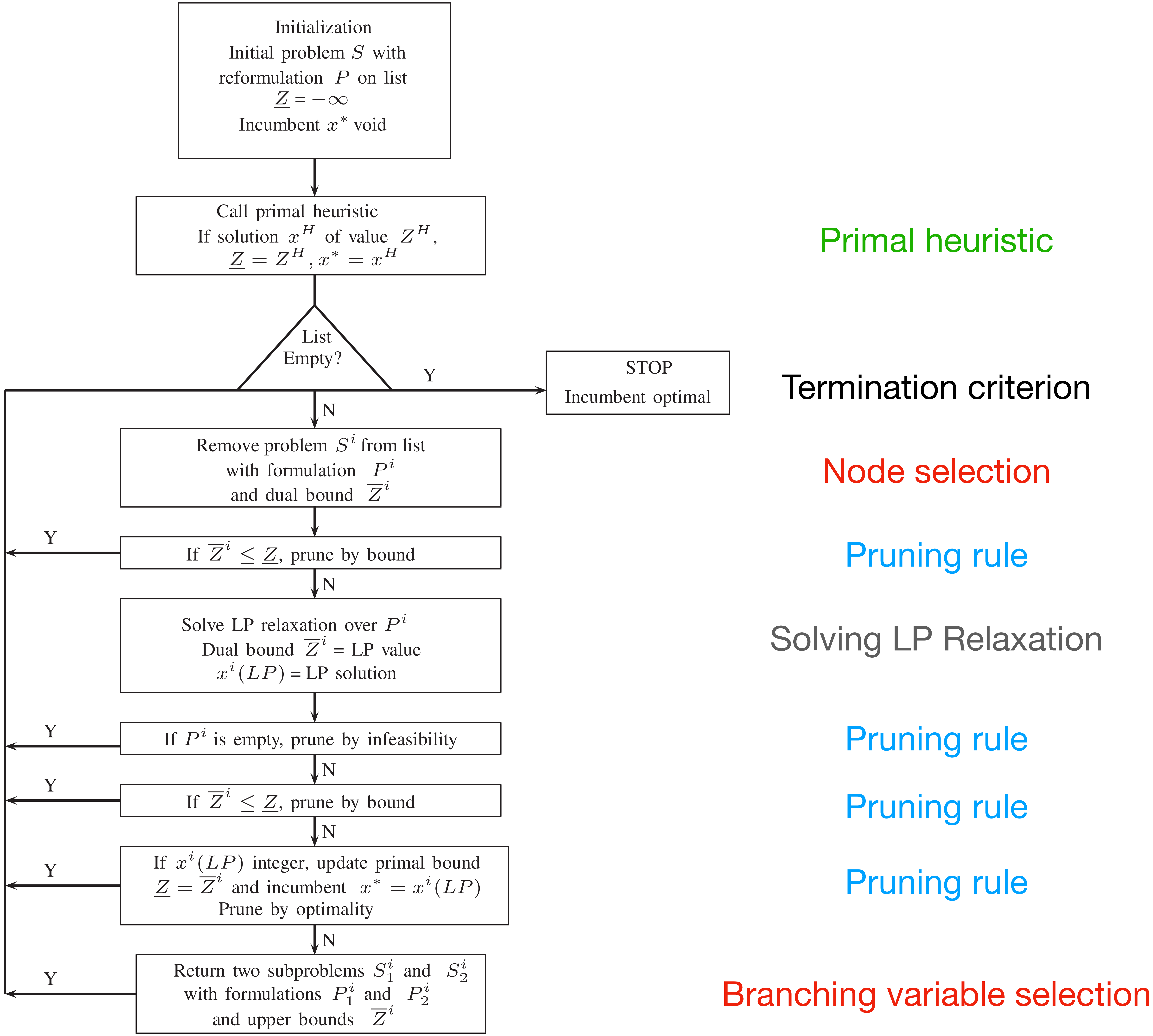
3 **Prune?**

4 **Add Cuts**

5 **Run Heuristics**

6 **Branch**





From Integer Programming by Wolsey

Figure 7.10 Branch-and-bound flow chart.

0-1 Knapsack Problem

There is a budget b available for investment in projects during the coming year and n projects are under consideration, where a_j is the outlay for project j and c_j is its expected return. The goal is to choose a set of projects so that the budget is not exceeded and the expected return is maximized.

Definition of the variables.

$x_j = 1$ if project j is selected, and $x_j = 0$ otherwise.

Definition of the constraints.

The budget cannot be exceeded:

$$\sum_{j=1}^{n} a_j x_j \leq b.$$

The variables are 0–1:

$$x_j \in \{0, 1\} \quad \text{for } j = 1, \ldots, n.$$

Definition of the objective function.

The expected return is maximized:

$$\max \sum_{j=1}^{n} c_j x_j.$$

Can you find a feasible solution greedily?

Hoos, Holger H. "Automated algorithm configuration and parameter tuning." Autonomous search. Springer, Berlin, Heidelberg, 2011. 37-71.

Greedy 0-1 knapsack

Exponent p_1 such that items are sorted w.r.t. $c_j / a_j^{p_1}$

- an algorithm A with parameters $p_1, \ldots, p_k$ that affect its behaviour,

$$p_1 \in (0,1]$$

- a space C of parameter settings (configurations), where $c \in C$ specifies values for $p_1, \ldots, p_k$,

- a set of problem instances I,

$$\sum_{i \in I} \sum_{j=1}^{n} c_j x_j$$

- a performance metric m that measures the performance of A on instance set I for a given configuration c,

find a configuration $c^* \in C$ such that running algorithm A on instance set I maximizes metric m

Why tune an EXACT solver?

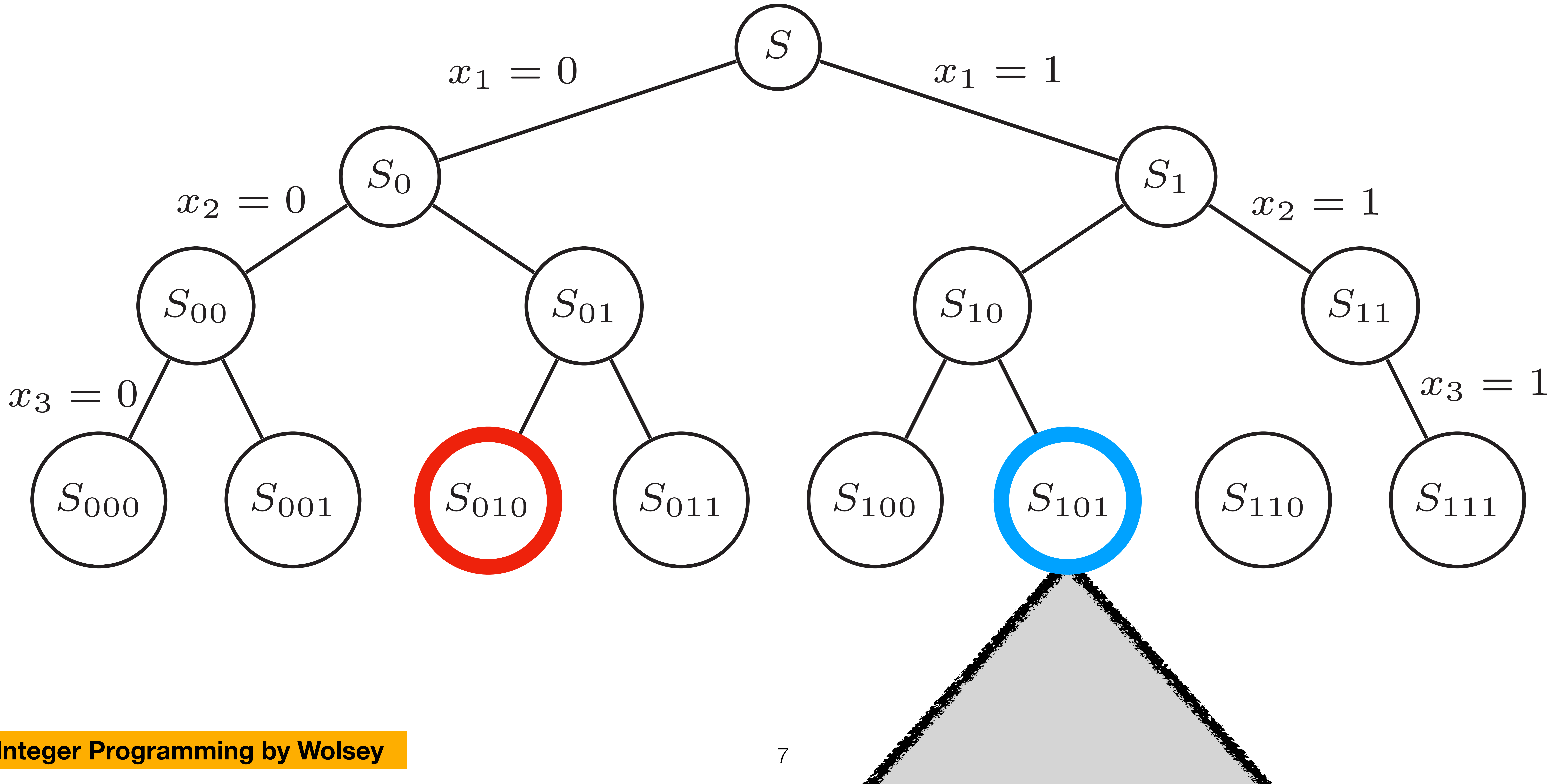
- **Given sufficient time**, solver is guaranteed to return global optimum (or declare instance infeasible).

A. Small changes to exact algorithm can dramatically influence its running time.

B. In practice, we have a limited time budget!

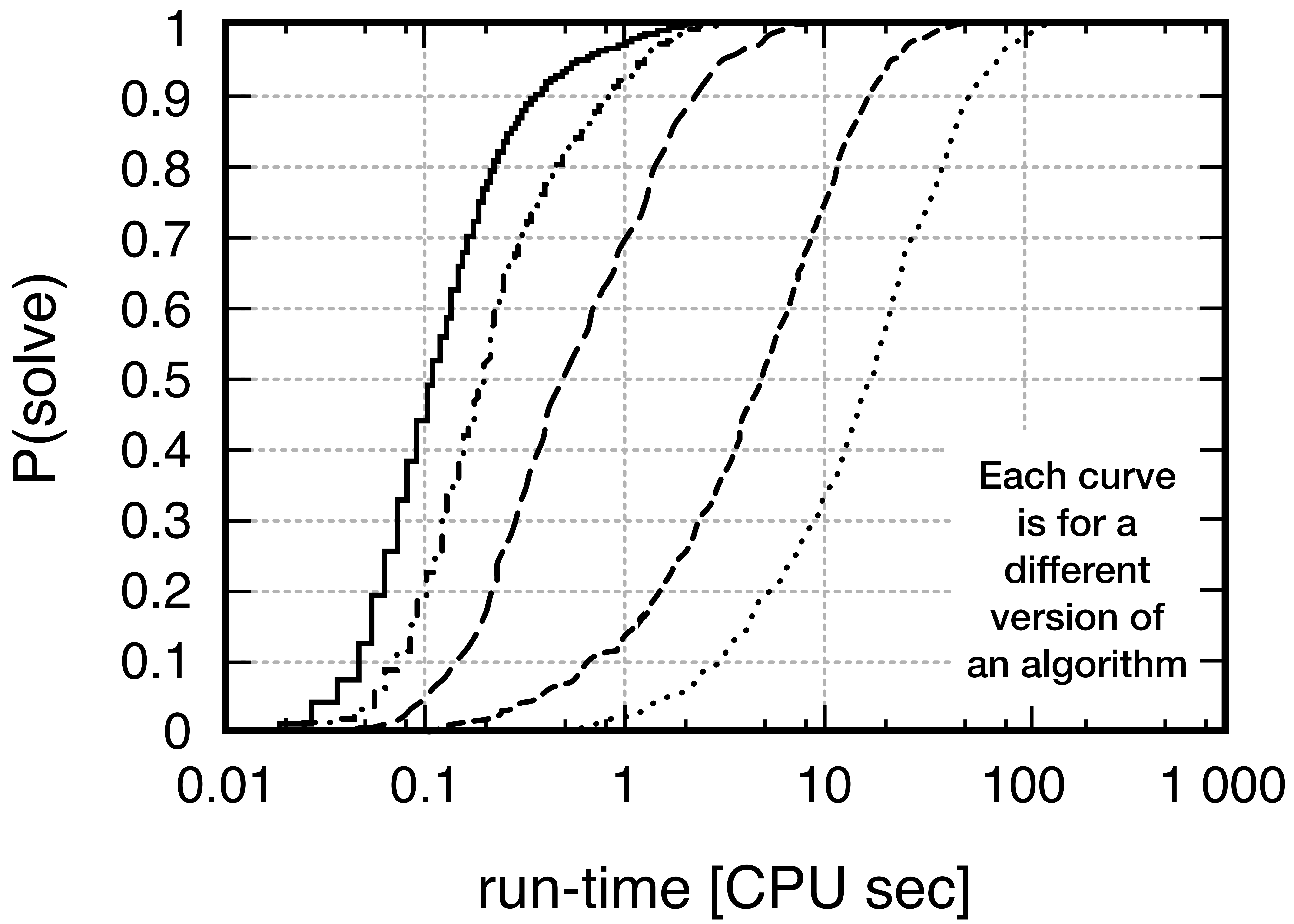
C. There is no universally superior (data-independent) “parameter setting”.

A. Small changes to exact algorithm can dramatically influence its running time.



From Integer Programming by Wolsey

B. In practice, we have a limited time budget!



Adapted from Holger Hoos, CPSC536H at UBC: http://www.cs.ubc.ca/labs/beta/Courses/CPSC536H-12/Slides/m5-all.pdf

C. There is no universally superior (data-independent) “parameter setting”.

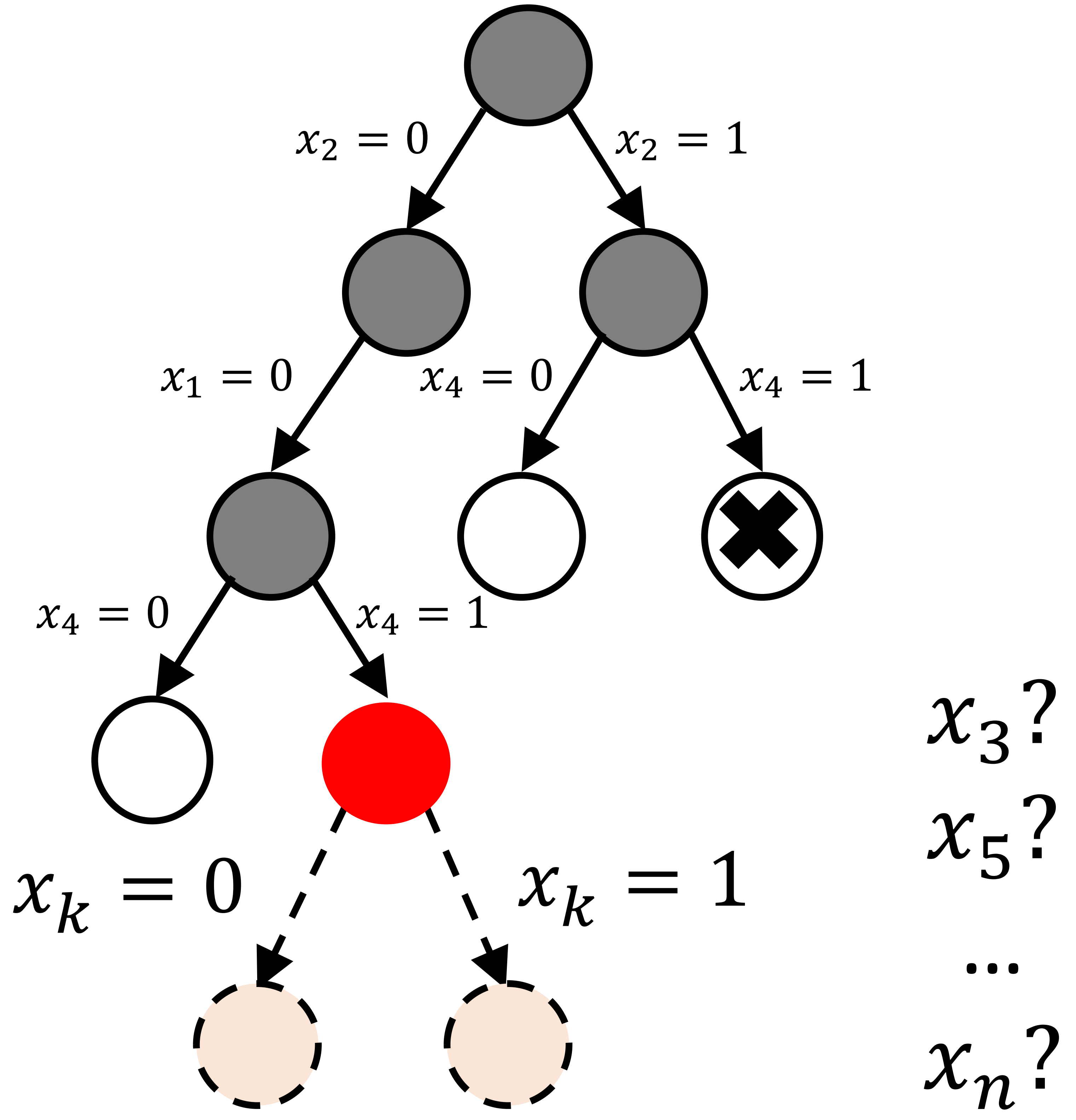
Algorithm 5.1 Generic variable selection

Input: Current subproblem Q with an optimal LP solution $\check{x} \notin X_{\text{MIP}}$.

Output: An index $j \in I$ of an integer variable x_j with fractional LP value $\check{x}_j \notin \mathbb{Z}$.

1. Let $F = \{j \in I \mid \check{x}_j \notin \mathbb{Z}\}$ be the set of branching candidates.
2. For all candidates $j \in F$, calculate a score value $s_j \in \mathbb{R}$.
3. Return an index $j \in F$ with $s_j = \max_{k \in F}\{s_k\}$.

$$\text{score}(q^-, q^+) = (1-\mu)\cdot\min\{q^-, q^+\} + \mu \cdot \max\{q^-, q^+\}$$



From Tobias Achterberg, doctoral dissertation (2007)

C. There is no universally superior (data-independent) "parameter setting".

$$\text{score}(q^-, q^+) = (1 - \mu) \cdot \min\{q^-, q^+\} + \mu \cdot \max\{q^-, q^+\}$$

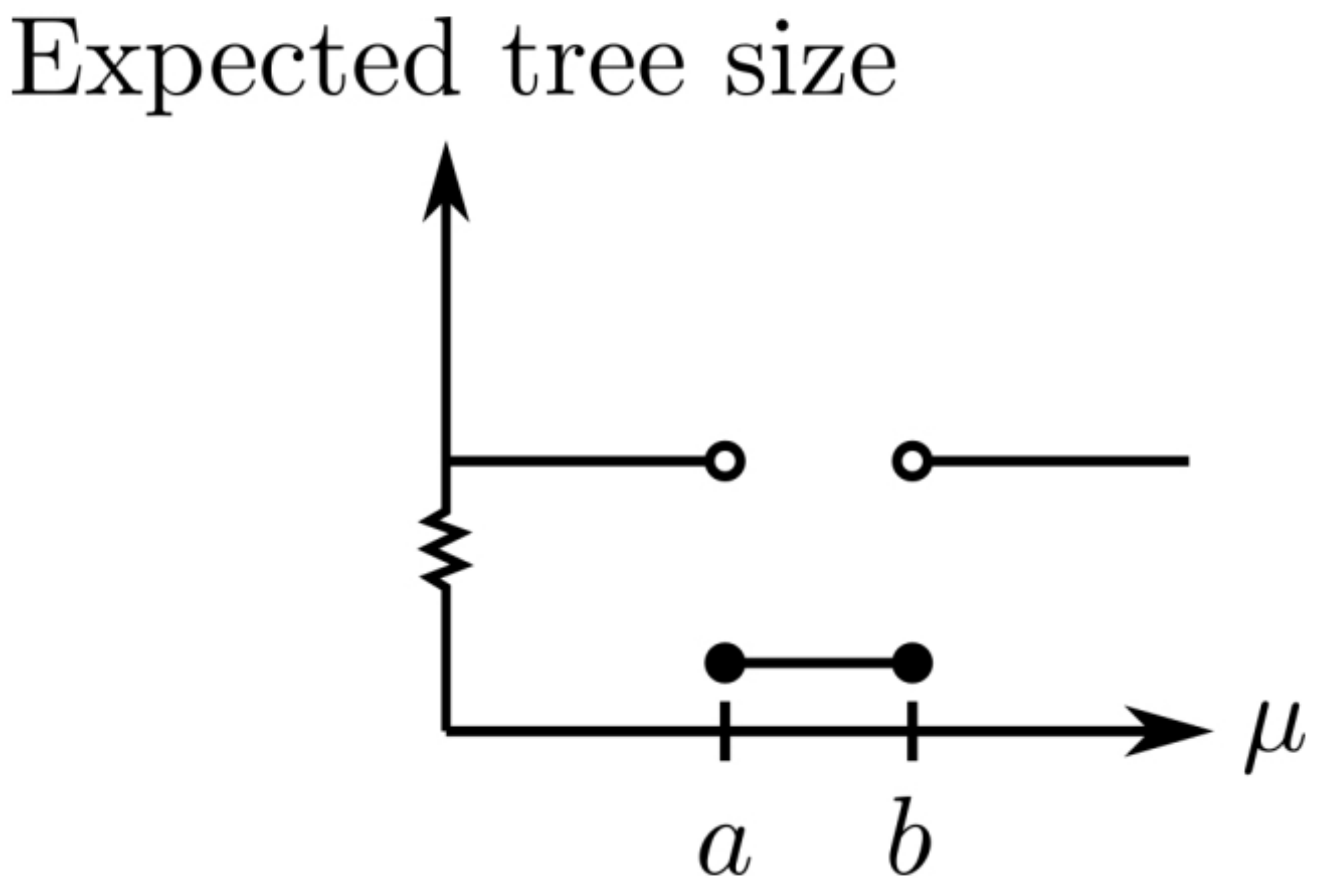
	test set	min ($\mu = 0$)		weighted ($\mu = \frac{1}{3}$)
time	MIPLIB	+26	<	+28
	CORAL	+25	<	+27
	MILP	+18	<	+36
	ENLIGHT	+34	>	−1
	ALU	+88	>	+85
	FCTP	+72	>	+6
	ACC	+43	>	+29
	FC	+58	>	−6
	ARCSET	+35	>	+22
	MIK	+134	>	+31
	total	**+29**	=	**+29**

Performance effect of different branching score functions for solving MIP instances. The values denote the percental changes in the shifted geometric mean of the runtime compared to the default score function. Positive values represent a deterioration, negative values an improvement.

From Tobias Achterberg, doctoral dissertation (2007)

C. There is no universally superior (data-independent) "parameter setting".

$$\text{score}(q^-, q^+) = (1-\mu)\cdot\min\{q^-, q^+\} + \mu\cdot\max\{q^-, q^+\}$$



(c) The expected tree size plot under the distribution $\mathcal{D}$ as a function of μ.

Theorem (informal): There exist distributions over MIP instances such that **setting μ between a and b gives small branch-and-bound trees**, but <u>other values give exponentially large trees</u>

Balcan, Maria-Florina, et al. "Learning to branch." *International conference on machine learning*. PMLR, 2018.

Why tune an EXACT solver?

- **Given sufficient time**, solver is guaranteed to return global optimum (or declare instance infeasible).

A. Small changes to exact algorithm can dramatically influence its running time.

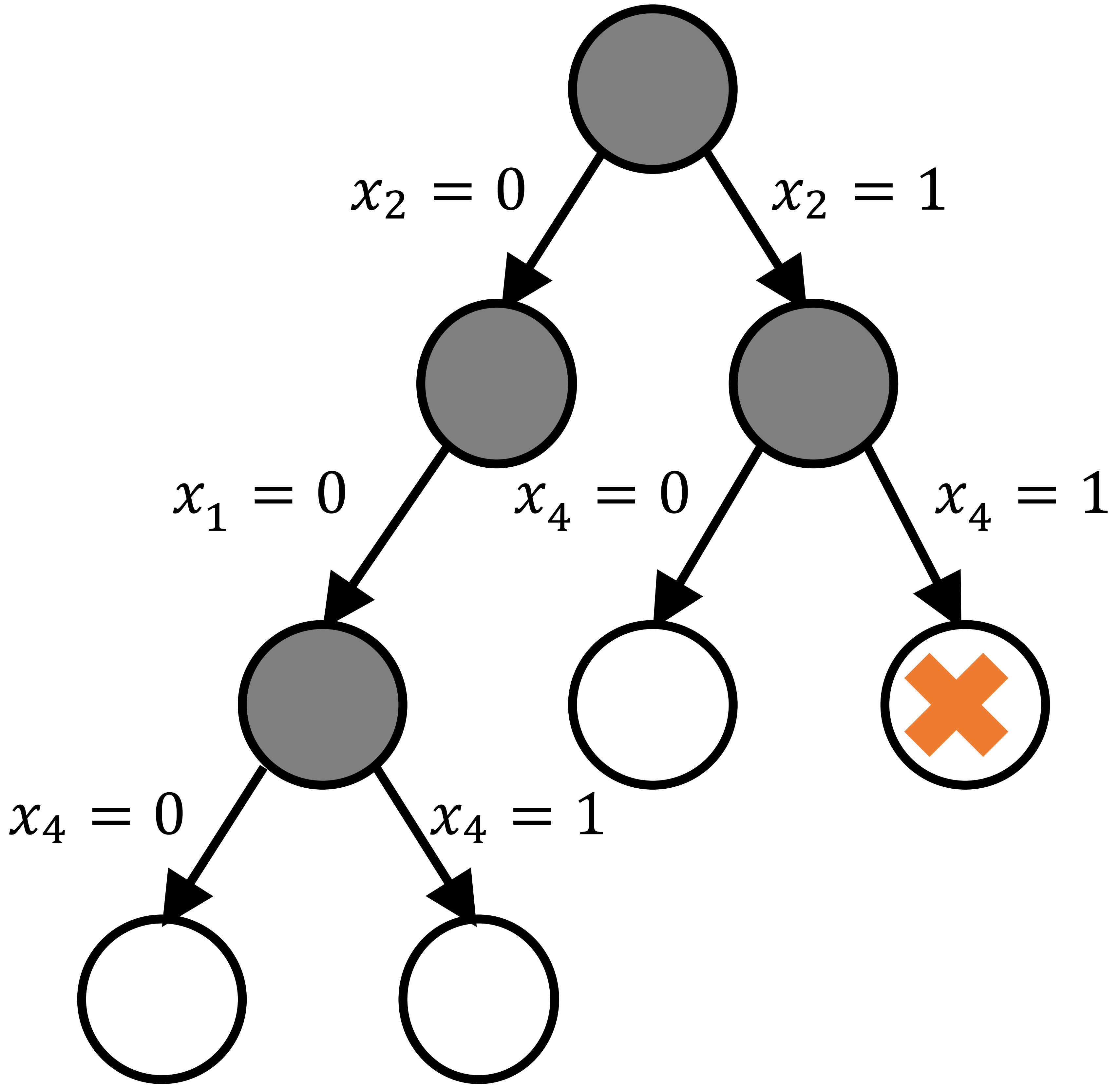
B. In practice, we have a limited time budget!

C. There is no universally superior (data-independent) “parameter setting”.

How do we evaluate solver performance?

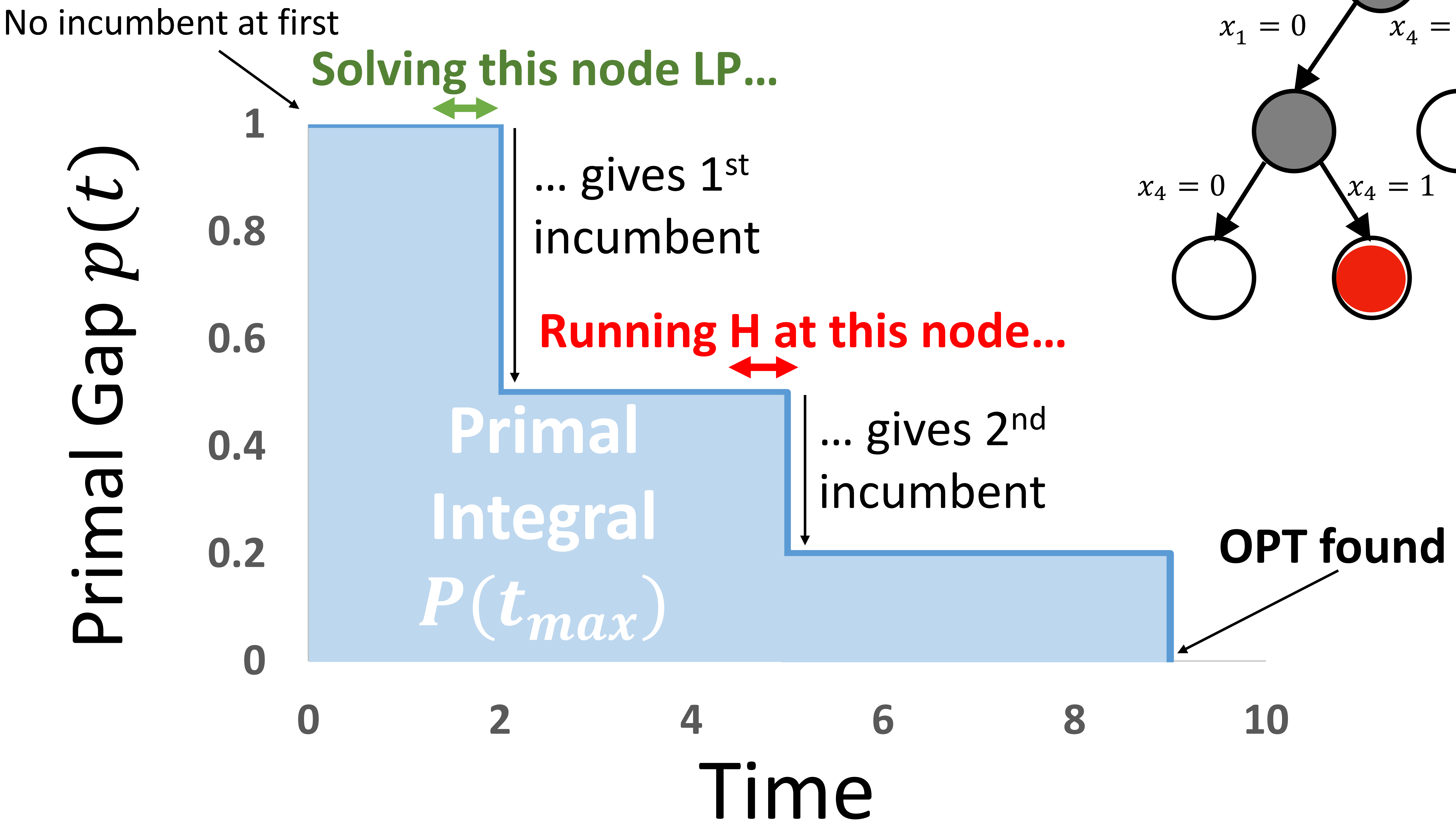
Evaluating a MIP solver on a set of instances

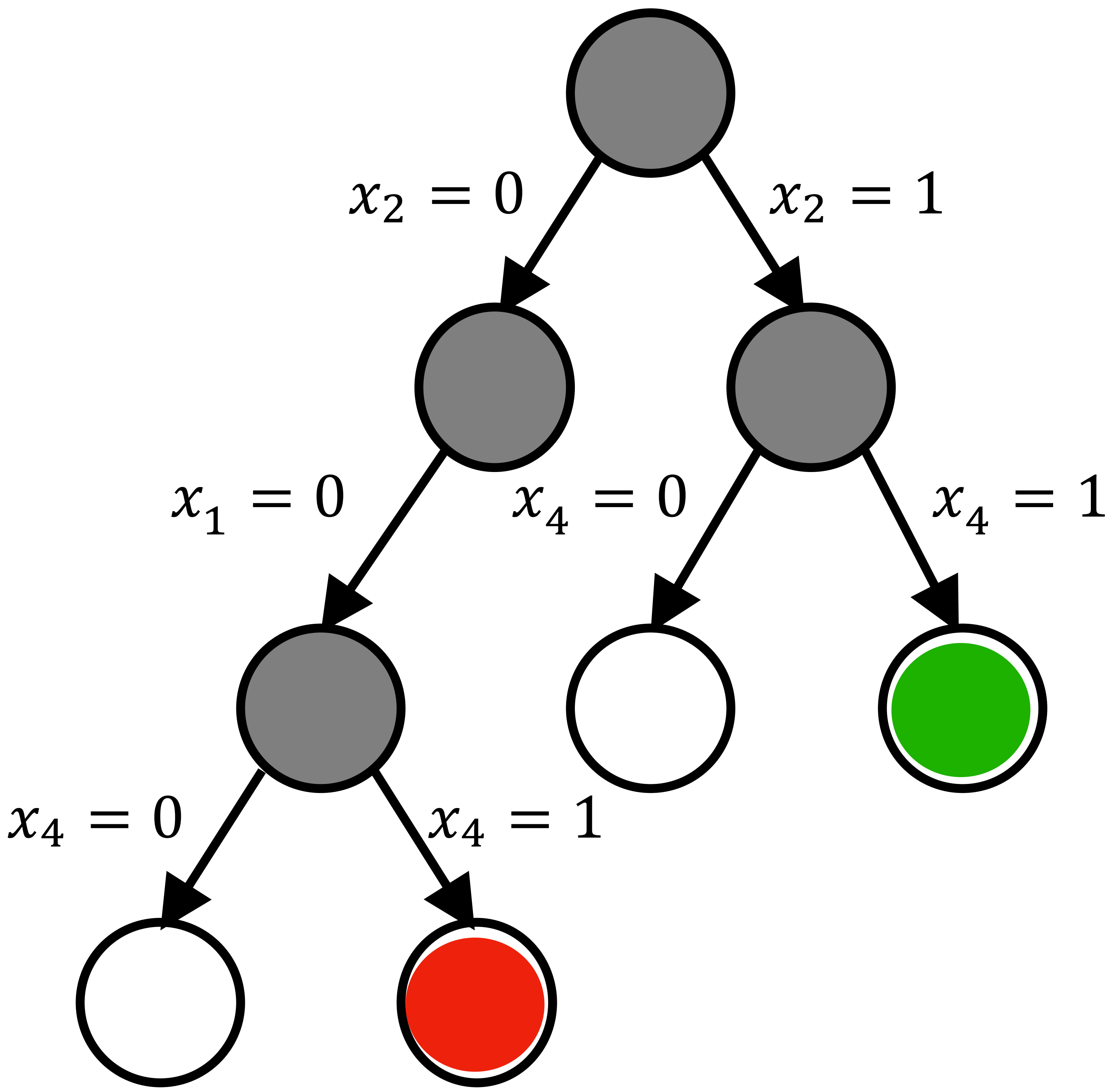
- **Total time**
- **Number of nodes in tree**
- What if solver did not terminate within time limit?
- Other limitations of these metrics?



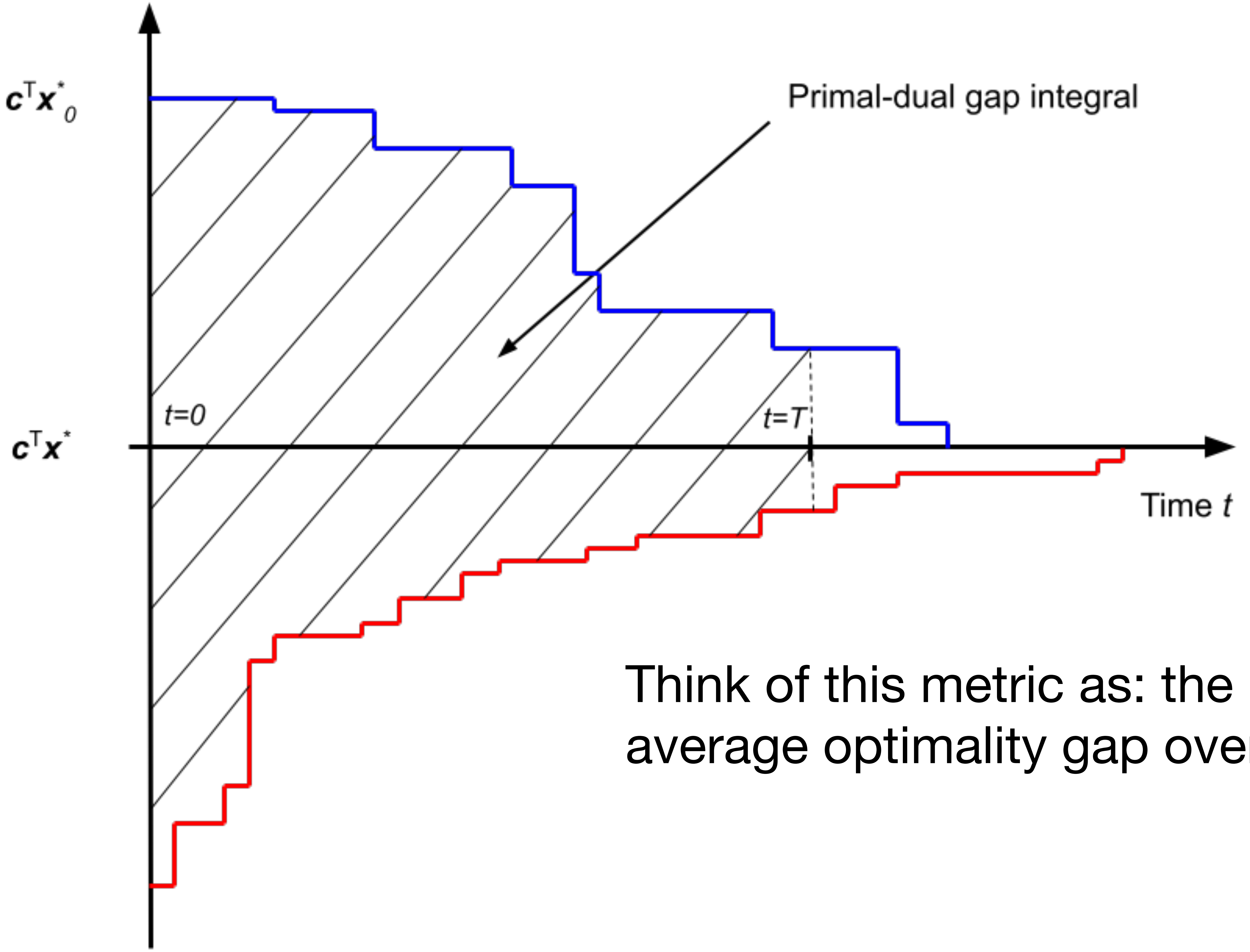
Primal Integral

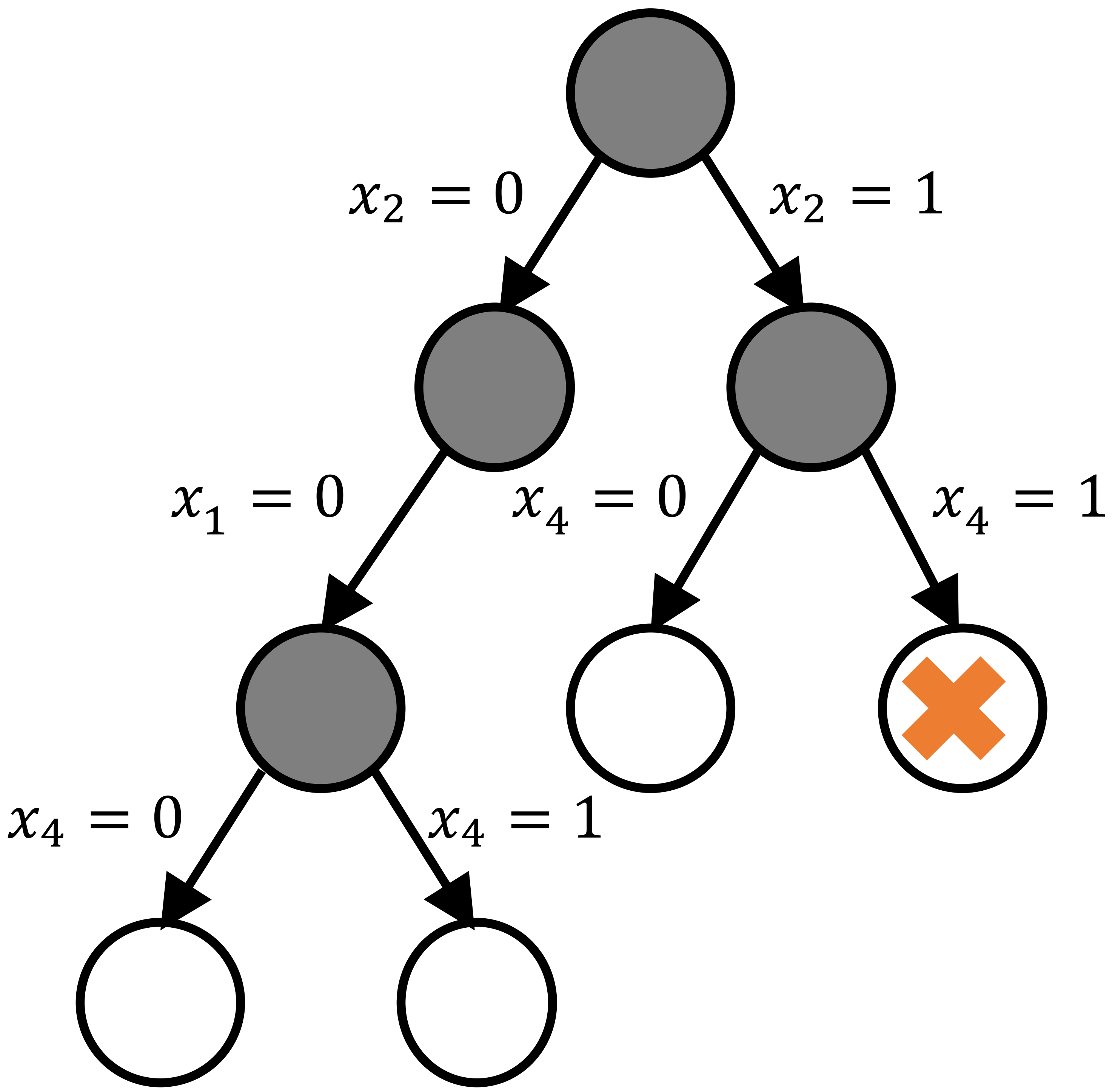
A more comprehensive metric?





Primal-Dual Integral





Think of this metric as: the average optimality gap over time

Illustration from https://www.ecole.ai/2021/ml4co-competition/

Instance Datasets for Learning in MIP

- MIPLIB2017: http://miplib2017.zib.de/
  - Community effort, 1000+ MILP instances
  - Wide variety of:
    - Applications
    - Sizes (10s of vars/cons to millions)
    - Mix of integer/binary/continuous vars.
    - Difficulties

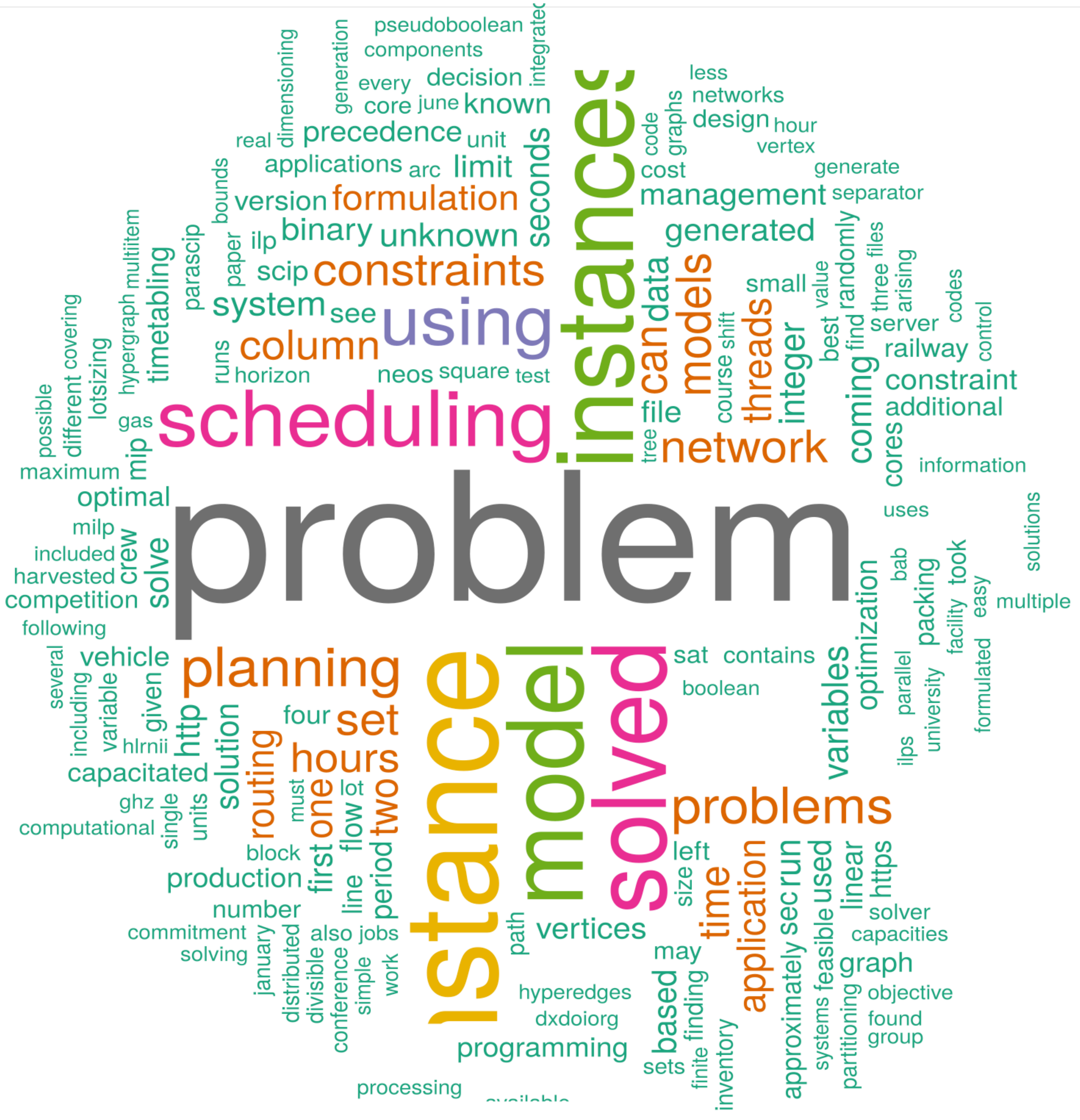
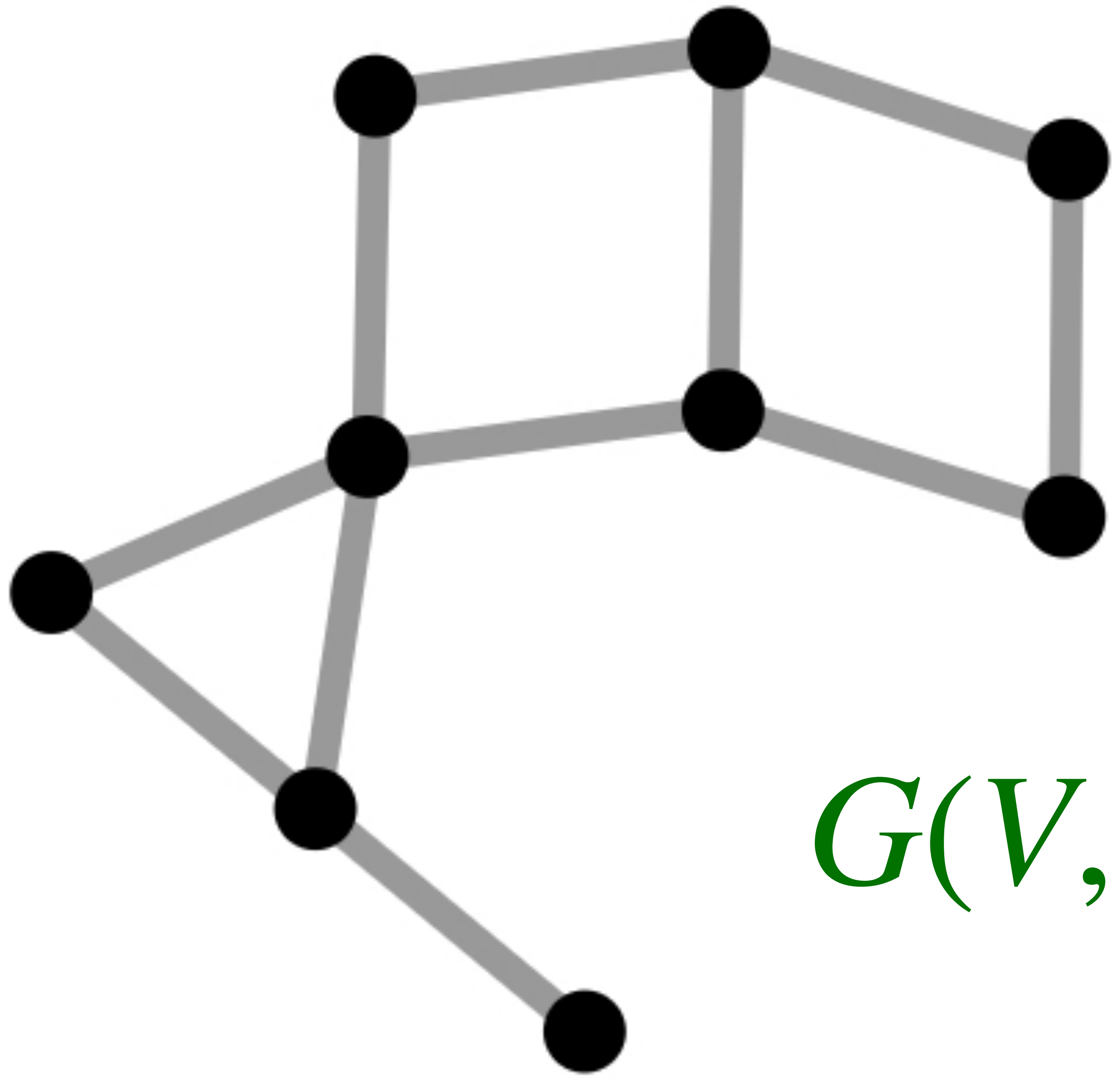


Image from http://miplib2017.zib.de/

Forest Harvesting

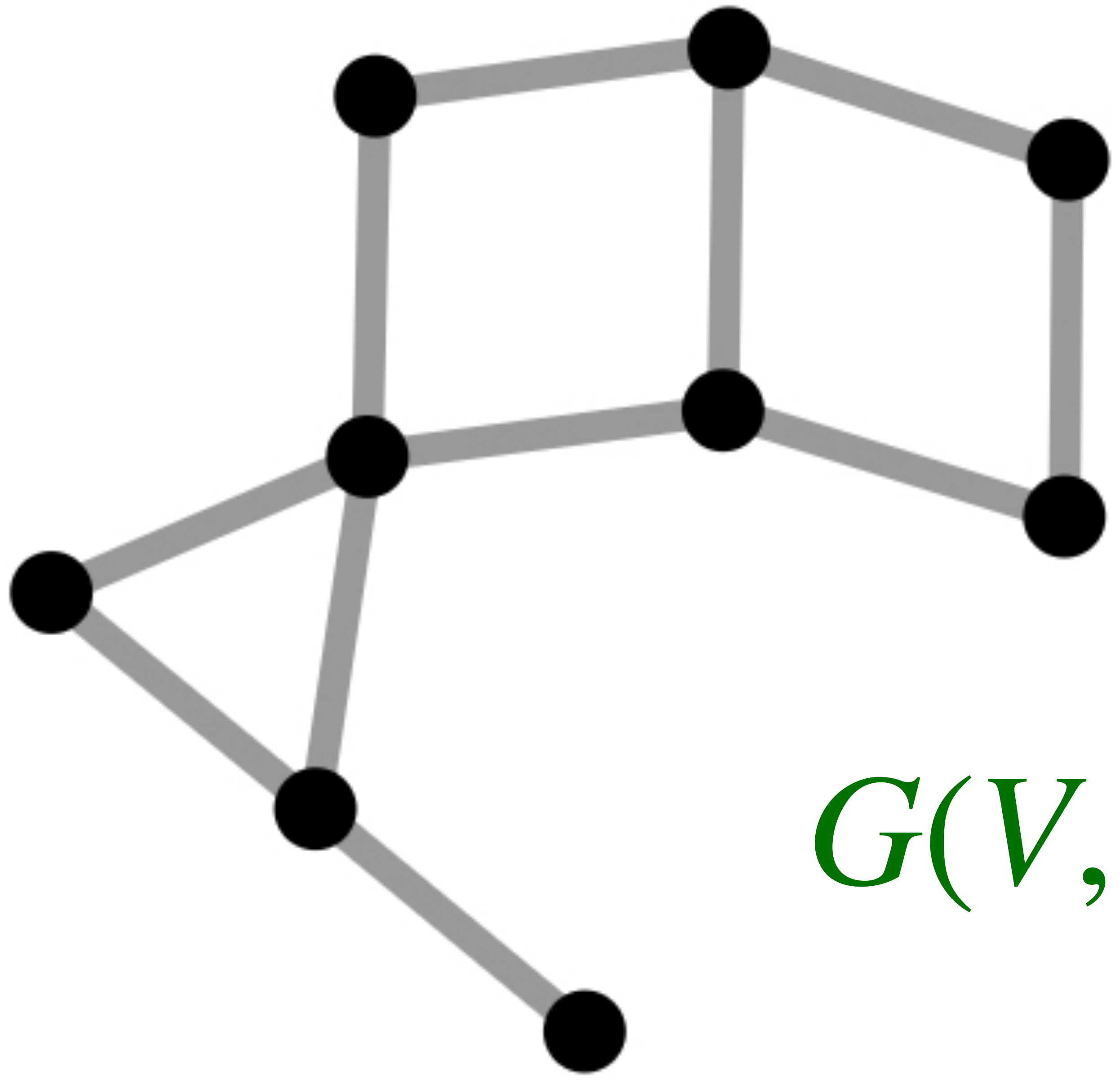
Forest Harvesting

Forest Harvesting



$G(V, E)$

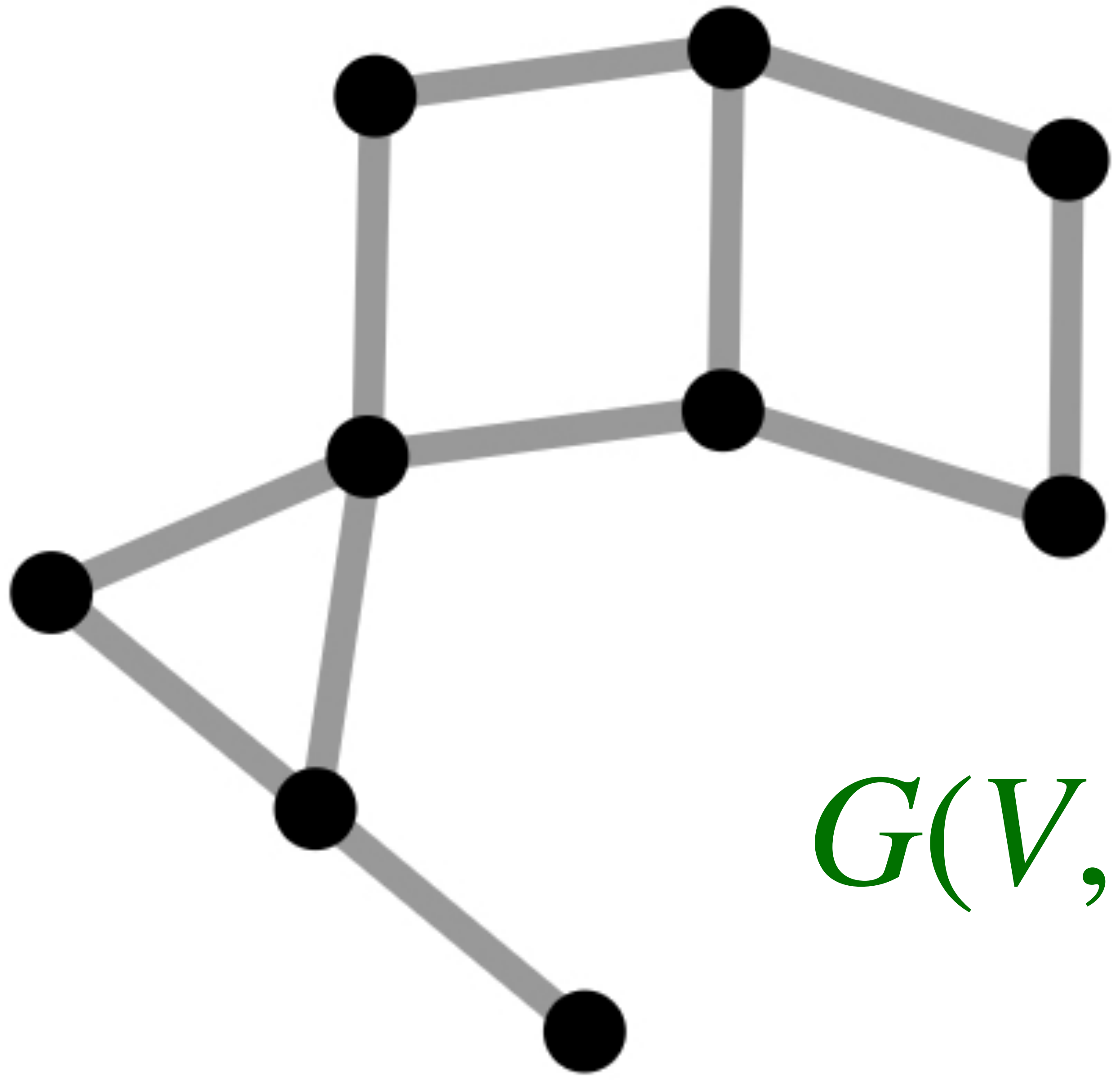
Forest Harvesting



$G(V, E)$

Forest Harvesting

Goal: Harvest subset of parcels to maximize **revenue**; pay **cost** for harvesting adjacent parcels



$G(V, E)$

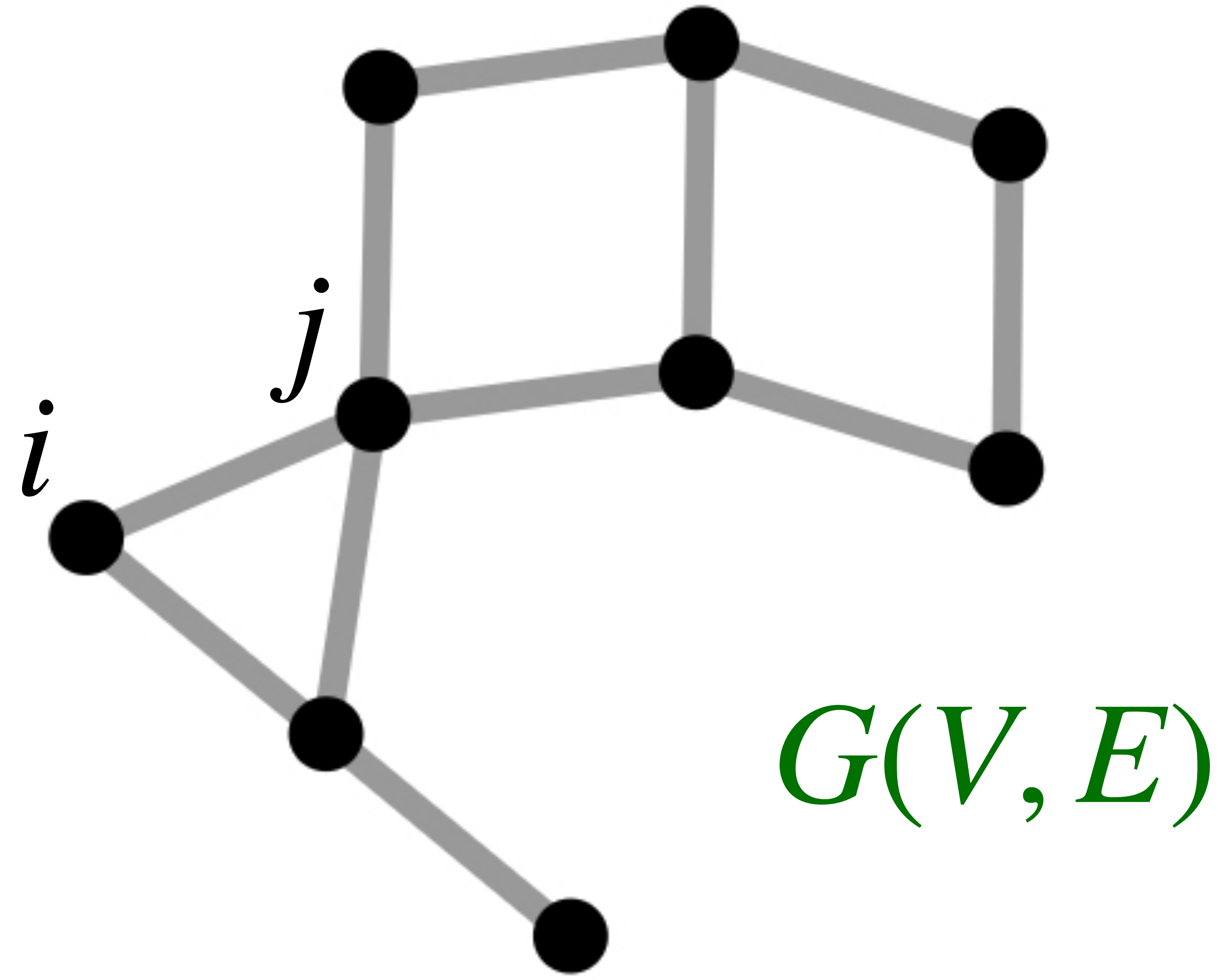
Forest Harvesting

Goal: Harvest subset of parcels to maximize **revenue**; pay **cost** for harvesting adjacent parcels

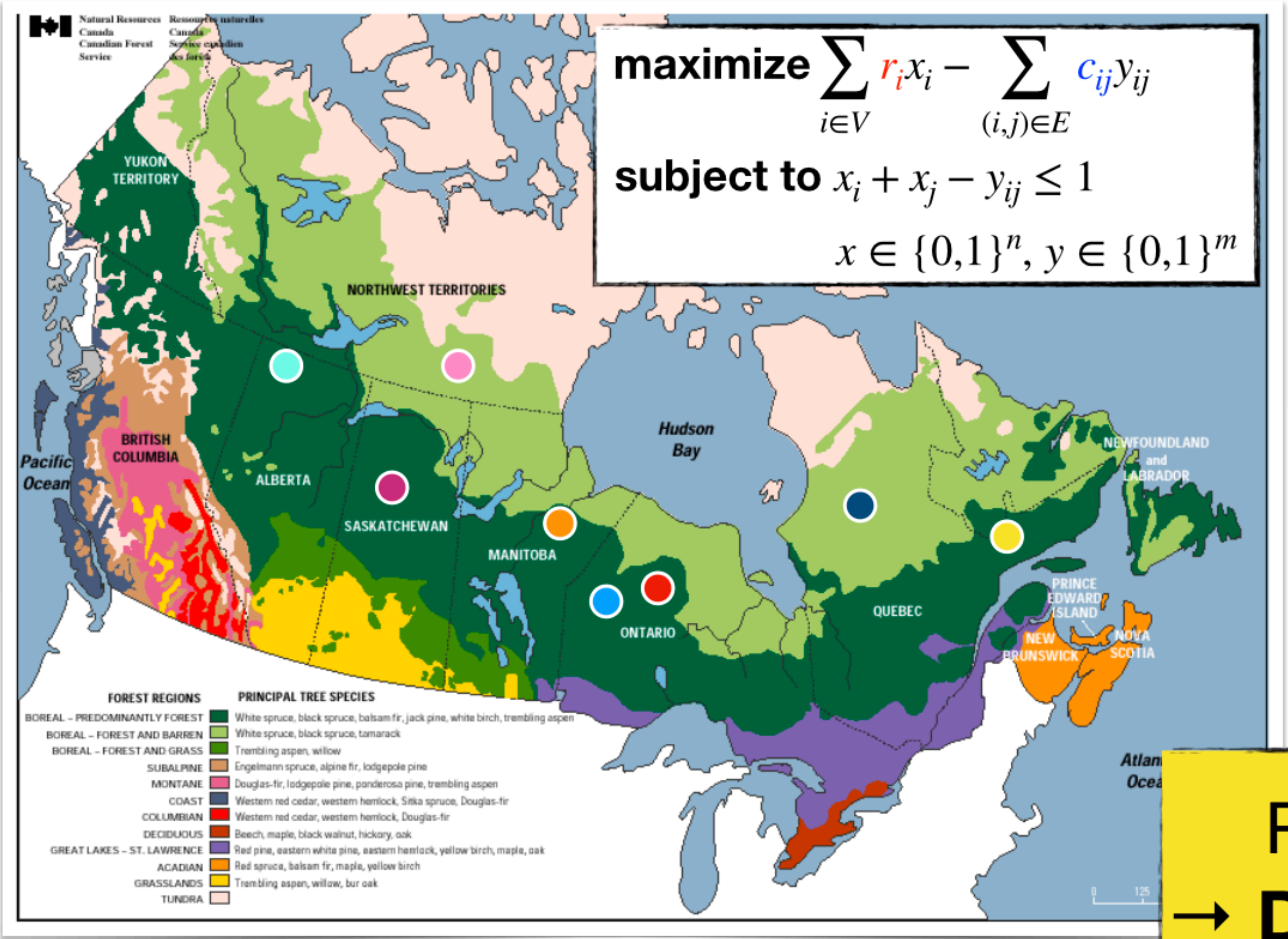
$$\textbf{maximize} \sum_{i \in V} r_i x_i - \sum_{(i,j) \in E} c_{ij} y_{ij}$$

$$\textbf{subject to } x_i + x_j - y_{ij} \le 1$$

$$x \in \{0,1\}^n,\ y \in \{0,1\}^m$$



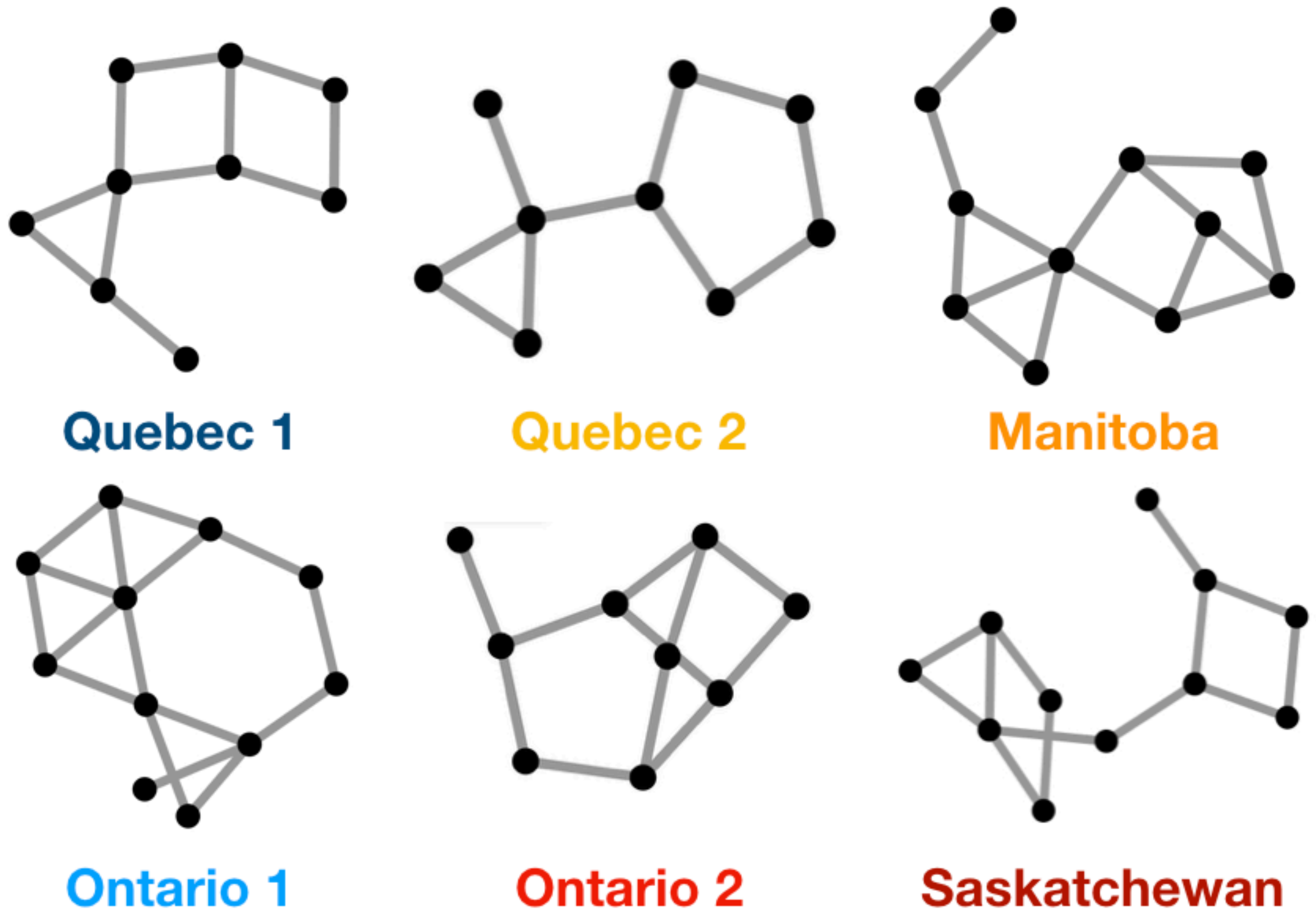
Forest Harvesting over Time/Space



maximize $\sum_{i \in V} r_i x_i - \sum_{(i,j) \in E} c_{ij} y_{ij}$

subject to $x_i + x_j - y_{ij} \leq 1$

$x \in \{0,1\}^n, \; y \in \{0,1\}^m$



Revenue varies **over time**

→ **Different** instances, **same** problem

Commonly used Problems

- Maximum Independent Set
- Generalized Independent Set
- Combinatorial Auctions
- Set Covering Problem
- Scheduling problems
- ….

ML4CO Competition

https://www.ecole.ai/2021/ml4co-competition/

Problem benchmark 1: Balanced Item Placement

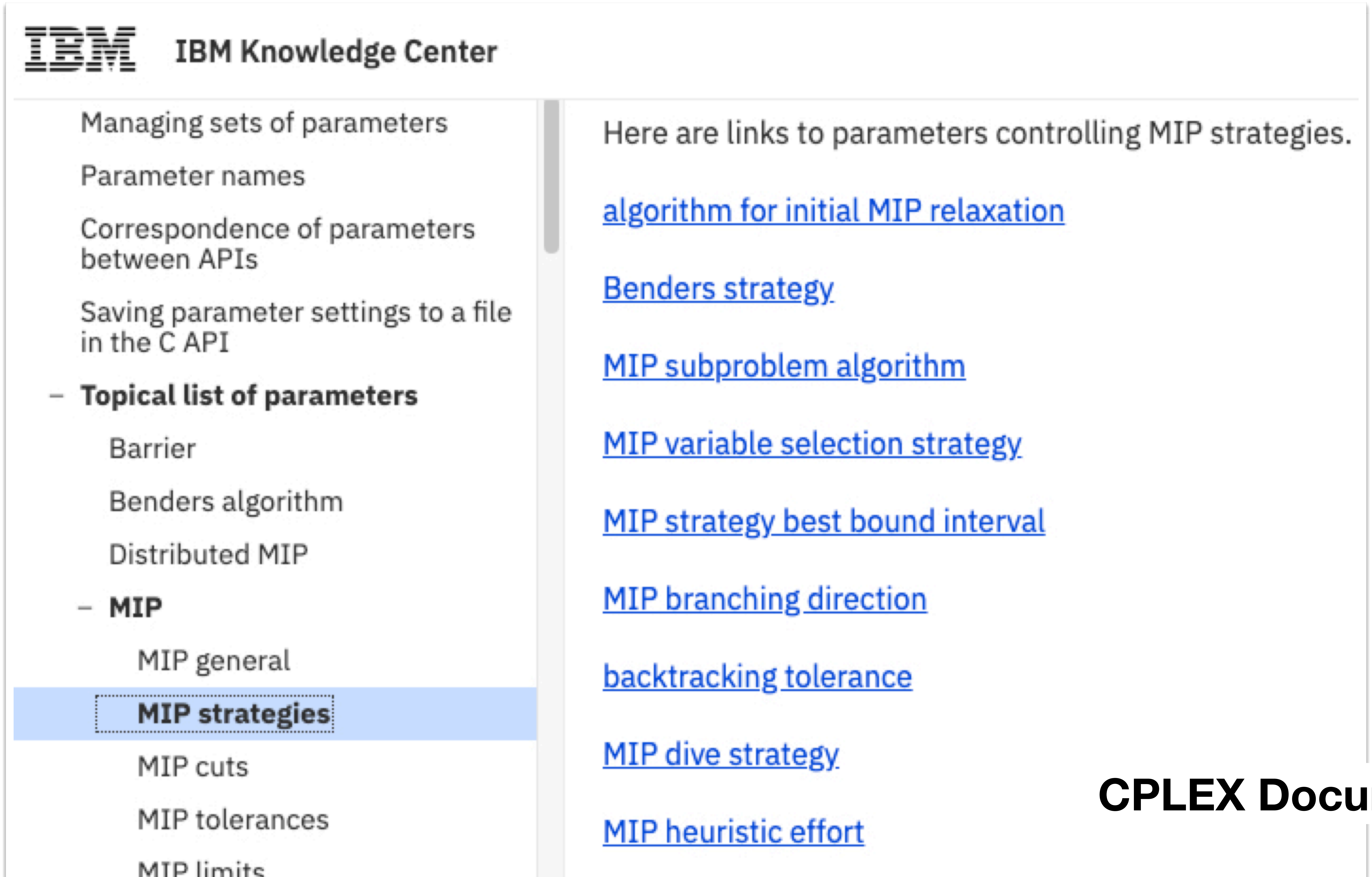
This problem deals with spreading items (e.g., files or processes) across containers (e.g., disks or machines) utilizing them *evenly*. Items can have multiple copies, but at most, one copy can be placed in a single bin. The number of items that can be moved is constrained, modeling the real-life situation of a live system for which some placement already exists. Each problem instance is modeled as a MILP, using a multi-dimensional multi-knapsack formulation. This dataset contains 10000 training instances (pre-split into 9900 train and 100 valid instances).

Problem benchmark 2: Workload Apportionment

This problem deals with apportioning workloads (e.g., data streams) across as few workers (e.g., servers) as possible. The apportionment is required to be robust to any one worker's failure. Each instance problem is modeled as a MILP, using a bin-packing with apportionment formulation. This dataset contains 10000 training instances (pre-split into 9900 train and 100 valid instances).

From https://www.ecole.ai/2021/ml4co-competition/

First Stop: Back to Configuration



CPLEX Documentation

First Stop: Back to Configuration

MIP variable selection strategy

Value	Symbol	Meaning
-1	`CPX_VARSEL_MININFEAS`	Branch on variable with minimum infeasibility
0	`CPX_VARSEL_DEFAULT`	Automatic: let CPLEX choose variable to branch on; **default**
1	`CPX_VARSEL_MAXINFEAS`	Branch on variable with maximum infeasibility
2	`CPX_VARSEL_PSEUDO`	Branch based on pseudo costs
3	`CPX_VARSEL_STRONG`	Strong branching
4	`CPX_VARSEL_PSEUDOREDUCED`	Branch based on pseudo reduced costs

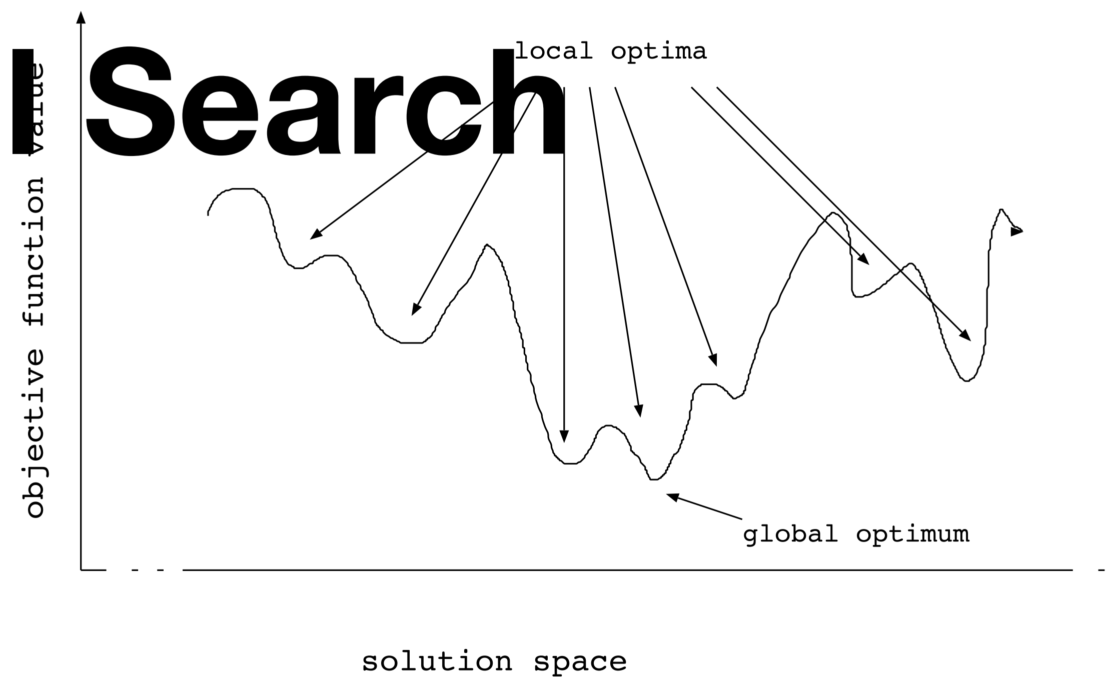
MIP heuristic frequency

Value	Meaning
-1	None
0	Automatic: let CPLEX choose; **default**
Any positive integer	Apply the periodic heuristic at this frequency

CPLEX Documentation

ParamILS

ILS: Iterated Local Search



procedure *ParamILS*
  input *target algorithm A, set of configurations C, set of problem instances I,*
    performance metric m;
  parameters *configuration* $c_0 \in C$*, integer r, integer s, probability pr*;
  output *configuration* c^*;

  $c^* := c_0$;
  for $i := 1$ **to** r **do**
    draw c from C uniformly at random;
    assess c against c^* based on performance of A on instances from I according to metric m;
    if c found to perform better than c^* **then**
      $c^* := c$;
    end if;
  end for;

  $c := c^*$;
  perform subsidiary local search on c;
  while termination condition not met **do**
    $c' := c$;
    perform s random perturbation steps on c'
    perform subsidiary local search on c';
    assess c' against c based on performance of A on instances from I according to metric m;
    if c' found to perform better than c **then**   *// acceptance criterion*
      update overall incumbent c^*;
      $c := c'$;
    end if;
    with probability pr **do**
      draw c from C uniformly at random;
    end with probability;
  end while;
  return c^*;
end *ParamILS*

Initial sampling phase

Random perturbation + local search

Evaluation

Update incumbent config.

Random restart!

Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." *Journal of Artificial Intelligence Research* 36 (2009): 267-306.

The first major configuration result for MIP

We present extensive evidence that ParamILS can find substantially improved parameter configurations of complex and highly optimized algorithms. In particular, we apply our automatic algorithm configuration procedures to the aforementioned commercial optimization tool CPLEX, one of the most powerful, widely used and complex optimization algorithms we are aware of. As stated in the CPLEX user manual (version 10.0, page 247), "A great deal of algorithmic development effort has been devoted to establishing default ILOG CPLEX parameter settings that achieve good performance on a wide variety of MIP models." We demonstrate consistent improvements over this default parameter configuration for a wide range of practically relevant instance distributions. In some cases, we were able to achieve an average speedup of over an order of magnitude on previously-unseen test instances (Section 7). We believe that these are the first results to be published on automatically configuring CPLEX or any other piece of software of comparable complexity.

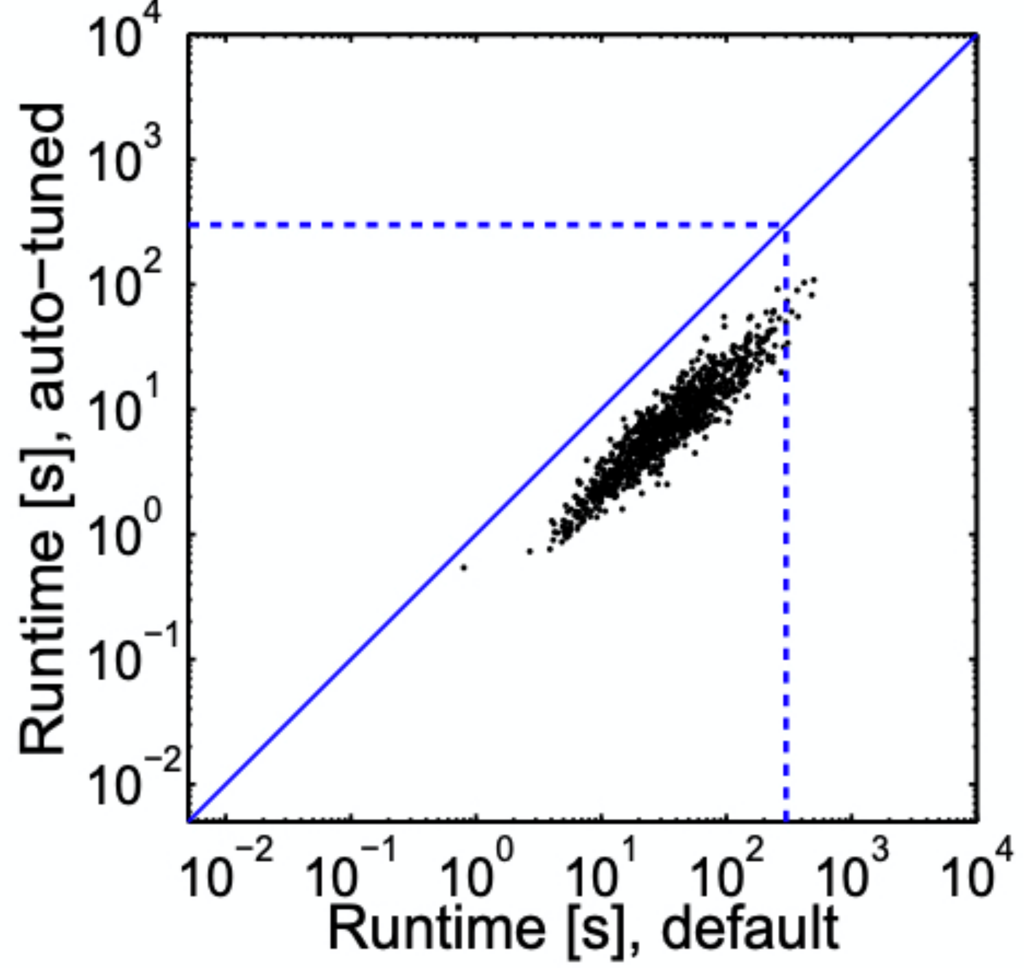
Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." *Journal of Artificial Intelligence Research* 36 (2009): 267-306.

The first major configuration result for MIP

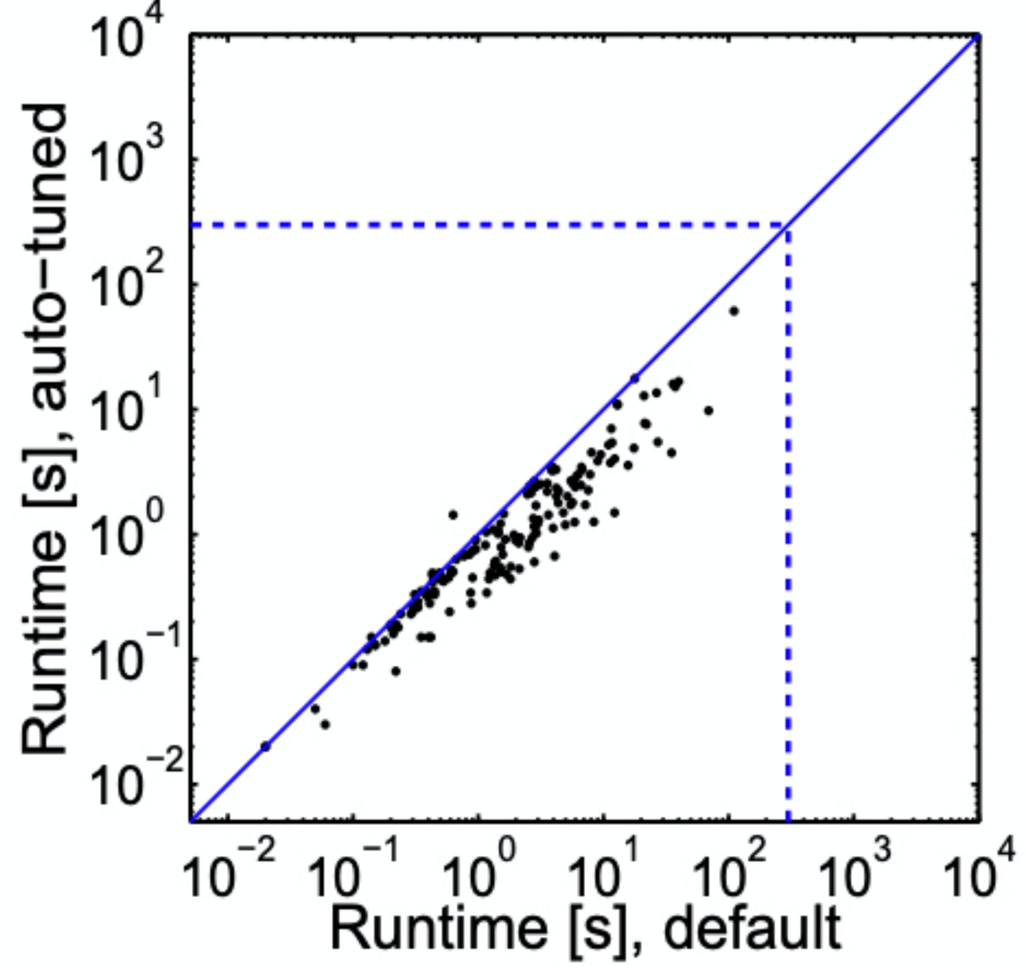
Algorithm	Parameter type	# parameters of type	# values considered	Total # configurations, $\|\Theta\|$
SAPS	Continuous	4	7	2 401
SPEAR	Categorical	10	2–20	$8.34 \cdot 10^{17}$
	Integer	4	5–8	
	Continuous	12	3–6	
CPLEX	Categorical	50	2–7	$1.38 \cdot 10^{37}$
	Integer	8	5–7	
	Continuous	5	3–5	

Table 2: Parameter overview for the algorithms we consider. More information on the parameters for each algorithm is given in the text. A detailed list of all parameters and the values we considered can be found in an online appendix at `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/algorithms.html`.

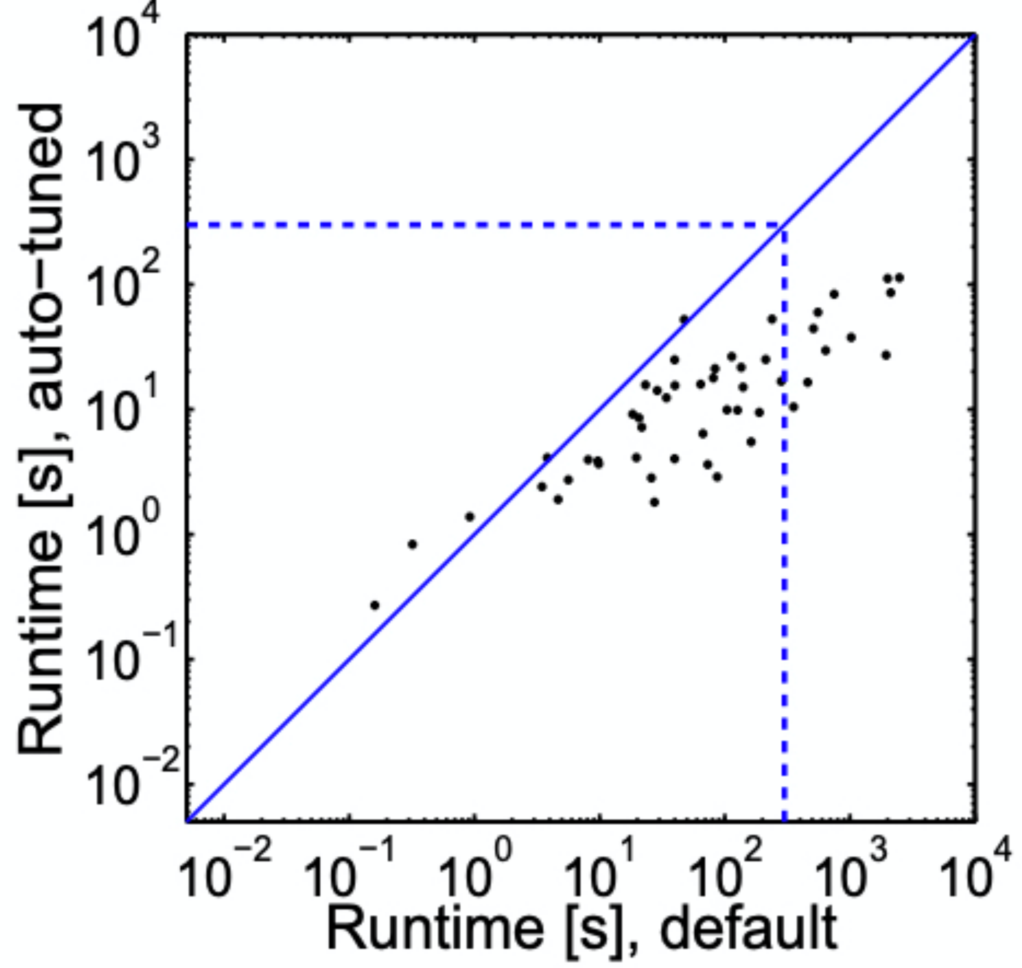
Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." *Journal of Artificial Intelligence Research* 36 (2009): 267-306.



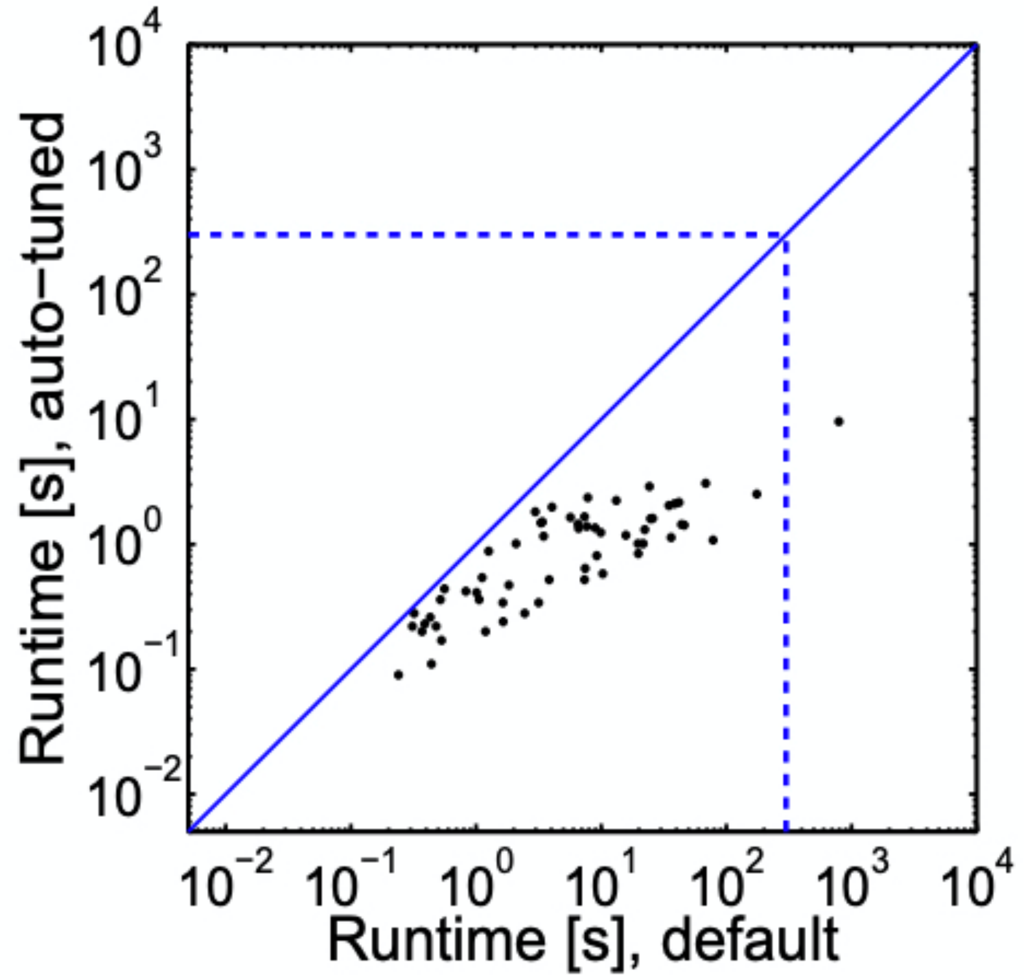
(a) CPLEX-REGIONS200.
72s *vs* 10.5s; no timeouts



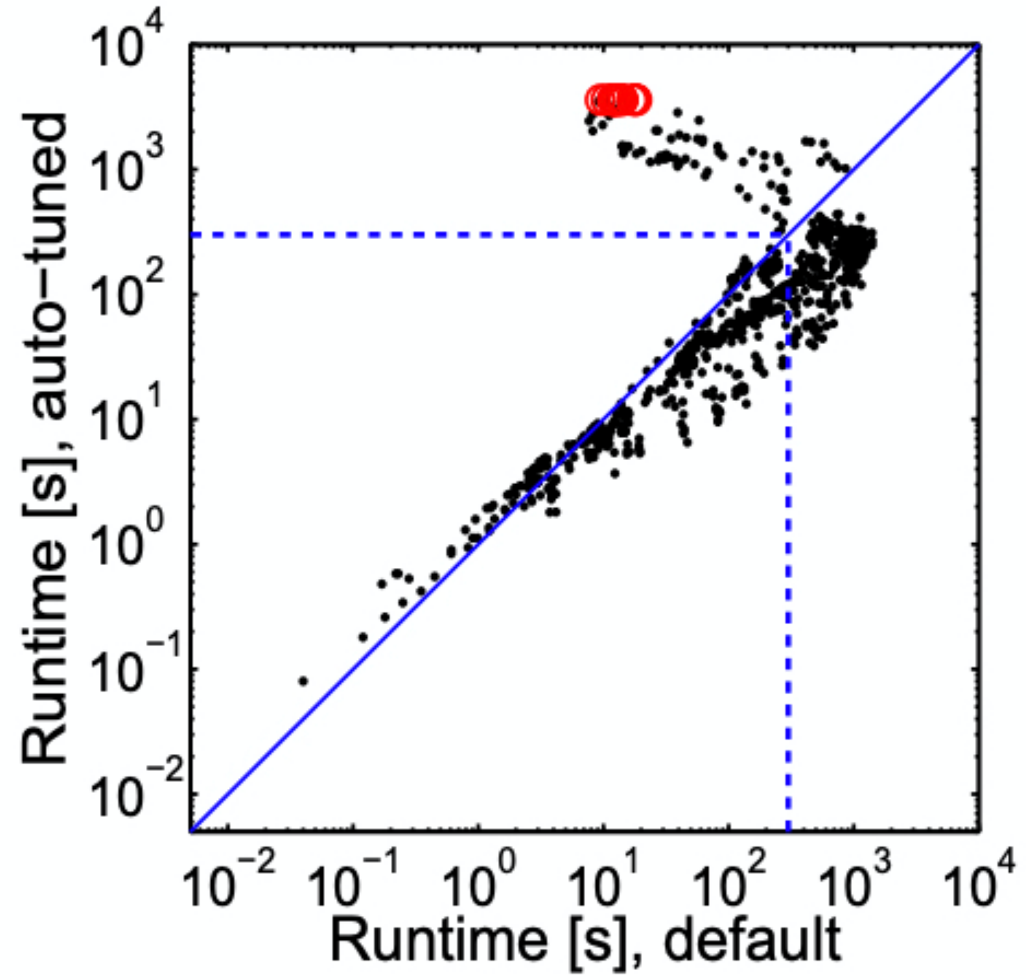
(b) CPLEX-CONIC.SCH.
5.37s *vs* 2.39.5s; no timeouts



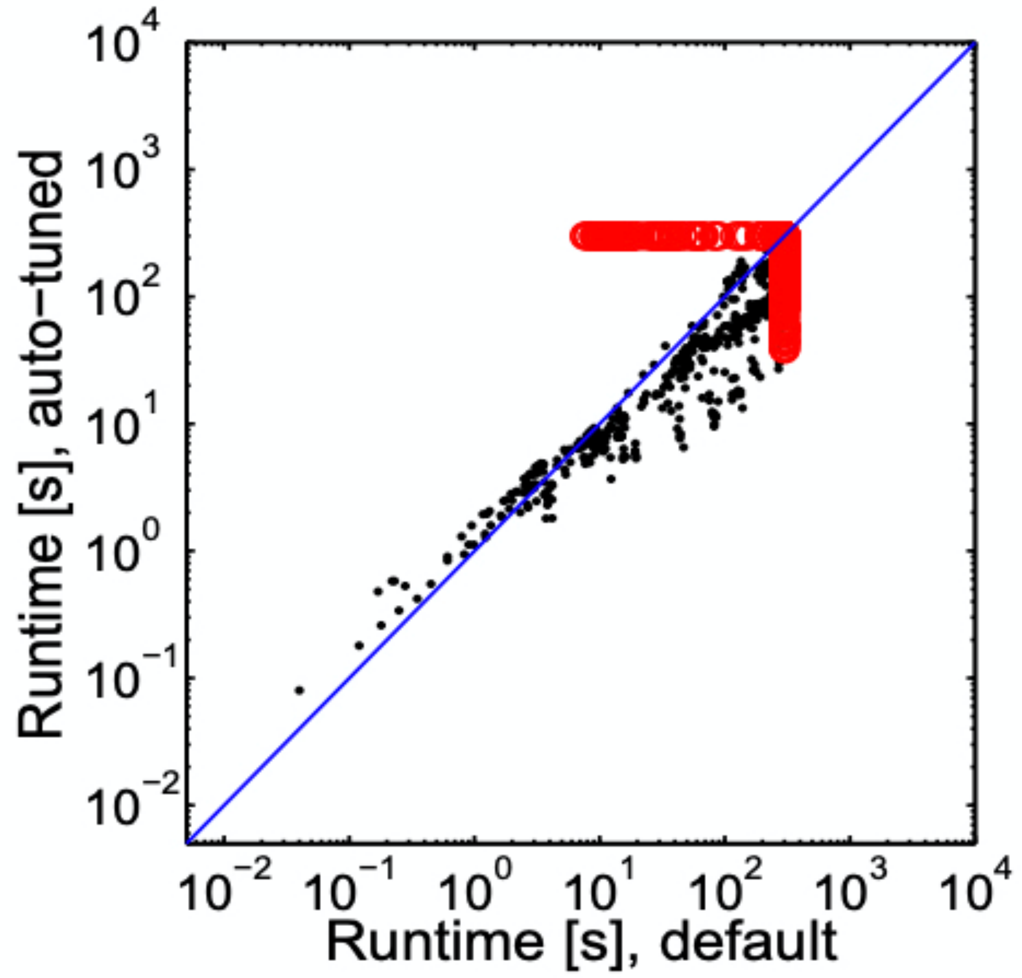
(c) CPLEX-CLS.
309s *vs* 21.5s; no timeouts



(d) CPLEX-MIK.
28s *vs* 1.2s; no timeouts



(e) CPLEX-QP.
296s *vs* 234s; 0 *vs* 21 timeouts



(f) CPLEX-QP, with test cutoff of 300 seconds.
81s *vs* 44s; 305 *vs* 150 timeouts

Hutter, Frank, et al. "ParamILS: an automatic algorithm configuration framework." *Journal of Artificial Intelligence Research* 36 (2009): 267-306.

Algorithm Configuration: Pros and Cons

See IJCAI-20 Tutorial: https://www.automl.org/tutorial_ac_ijcai20/

Automated Algorithm Configuration

- ParamILS [Hutter et al., JAIR 2009], SMAC [Hutter et al., LION 2011]
- Key Idea: search over parameter configurations
  - Stochastic Local Search or Bayesian Optimization

- Great for algorithms with many parameters
  - 2-52x speedups for CPLEX on some problem distributions [Hutter et al., CPAIOR 2010]

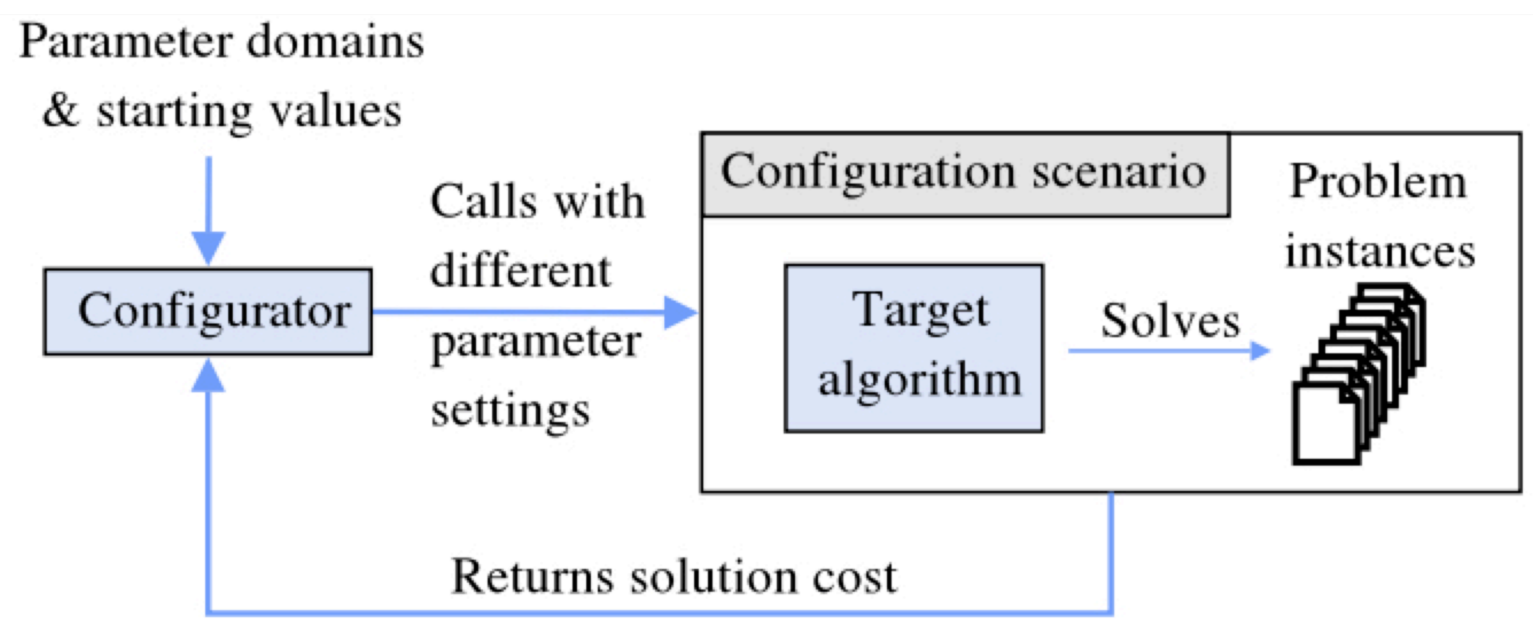


Fig. 1. A configuration procedure (short: configurator) executes the target algorithm with specified parameter settings on one or more problem instances, observes algorithm performance, and uses this information to decide which subsequent target algorithm runs to perform. A configuration scenario includes the target algorithm to be configured and a collection of instances.

Limitations

- Operates at the instance-level, not the **algorithm iteration-level**
- Assumes human-designed parameter space is rich enough

Algorithm: LP-based Branch-and-Bound

Input: a MIP $\min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}\ \forall j \in I\}$

Output: an optimal solution $x^*, z^* := c^T x^*$

1 Initialize: Queue of sub-problems (nodes) $\mathcal{L} := \{N_0\}$, Best value $z^* := \infty$, Best solution $x^* := \emptyset$

2 Terminate? If $\mathcal{L} = \emptyset$, **return** x^*

3 Select Node **[what selection rule?]**: Choose a node N_i to process from $\mathcal{L}$

4 Evaluate & Prune: Solve the LP relaxation of N_i and prune node if applicable.

5 Add Cuts **[which cuts to add?]**: new constraints that tighten the formulation.

6 Run Heuristics **[which heuristics to run?]**: try to find a better solution.

7 Select Branching Variable **[what selection rule?]**: Choose a variable that has fractional value in the LP solution of N_i. Create two new subproblems N_{i1} and N_{i2}. Go to line 2.

N_0

Algorithm: LP-based Branch-and-Bound

Input: a MIP $\min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}\ \forall j \in I\}$

Output: an optimal solution $x^*, z^* := c^T x^*$

1 Initialize: Queue of sub-problems (nodes) $\mathcal{L} := \{N_0\}$, Best value $z^* := \infty$, Best solution $x^* := \emptyset$

2 Terminate? If $\mathcal{L} = \emptyset$, **return** x^*

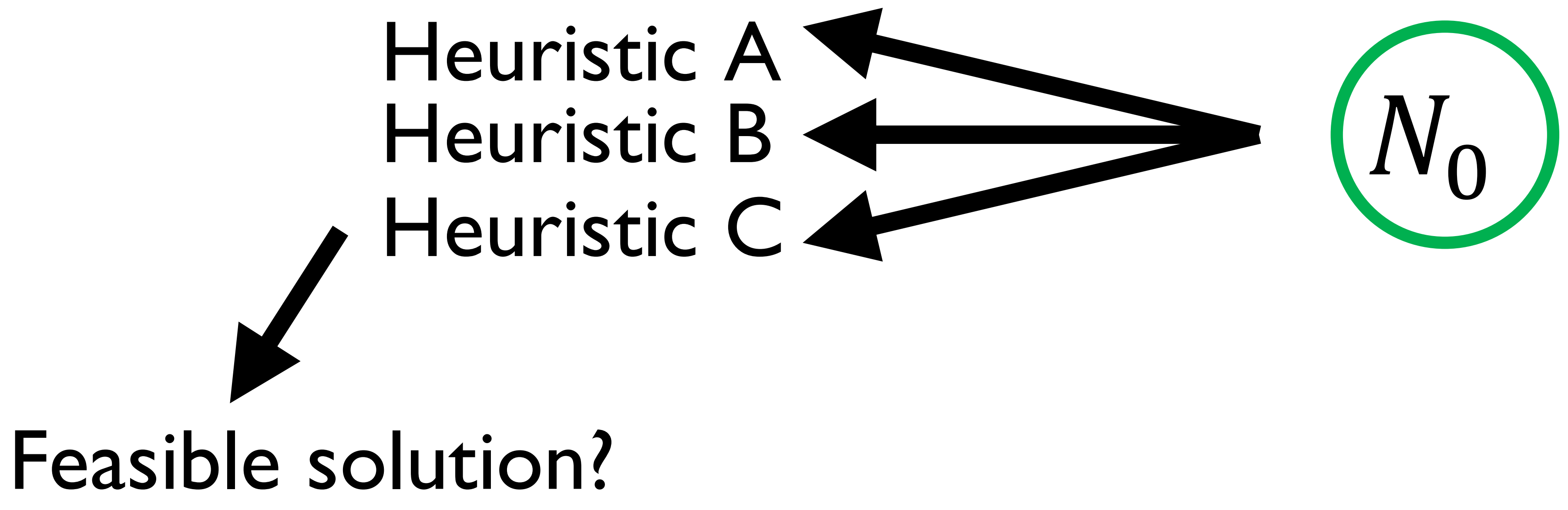
3 Select Node **[what selection rule?]**: Choose a node N_i to process from $\mathcal{L}$

4 Evaluate & Prune: Solve the LP relaxation of N_i and prune node if applicable.

5 Add Cuts **[which cuts to add?]**: new constraints that tighten the formulation.

6 Run Heuristics **[which heuristics to run?]**: try to find a better solution.

7 Select Branching Variable **[what selection rule?]**: Choose a variable that has fractional value in the LP solution of N_i. Create two new subproblems N_{i1} and N_{i2}. Go to line 2.



Algorithm: LP-based Branch-and-Bound

Input: a MIP $\min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}\ \forall j \in I\}$

Output: an optimal solution $x^*, z^* := c^T x^*$

1 Initialize: Queue of sub-problems (nodes) $\mathcal{L} := \{N_0\}$, Best value $z^* := \infty$, Best solution $x^* := \emptyset$

2 Terminate? If $\mathcal{L} = \emptyset$, **return** x^*

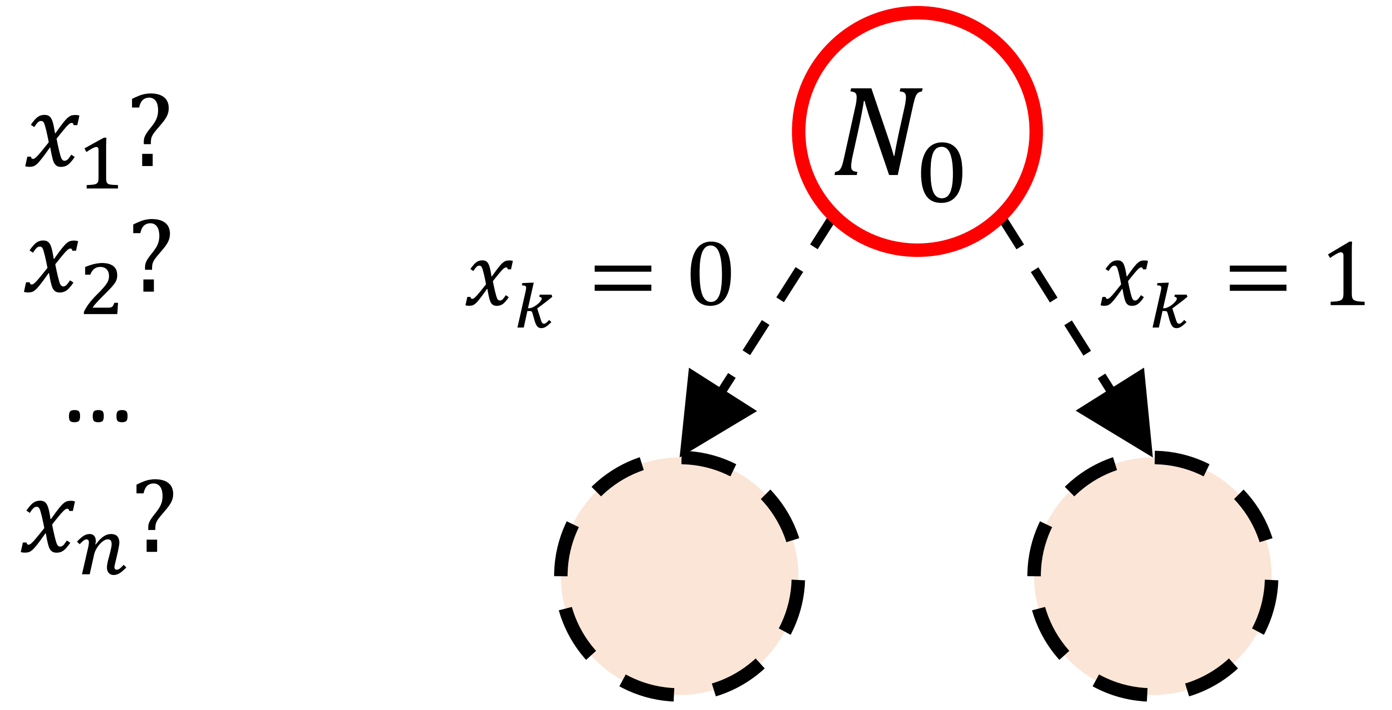
3 Select Node **[what selection rule?]**: Choose a node N_i to process from $\mathcal{L}$

4 Evaluate & Prune: Solve the LP relaxation of N_i and prune node if applicable.

5 Add Cuts **[which cuts to add?]**: new constraints that tighten the formulation.

6 Run Heuristics **[which heuristics to run?]**: try to find a better solution.

7 Select Branching Variable **[what selection rule?]**: Choose a variable that has fractional value in the LP solution of N_i. Create two new subproblems N_{i1} and N_{i2}. Go to line 2.



Algorithm: LP-based Branch-and-Bound

Input: a MIP $\min\{c^T x \mid Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z}\ \forall j \in I\}$

Output: an optimal solution $x^*, z^* := c^T x^*$

1 Initialize: Queue of sub-problems (nodes) $\mathcal{L} := \{N_0\}$, Best value $z^* := \infty$, Best solution $x^* := \emptyset$

2 Terminate? If $\mathcal{L} = \emptyset$, **return** x^*

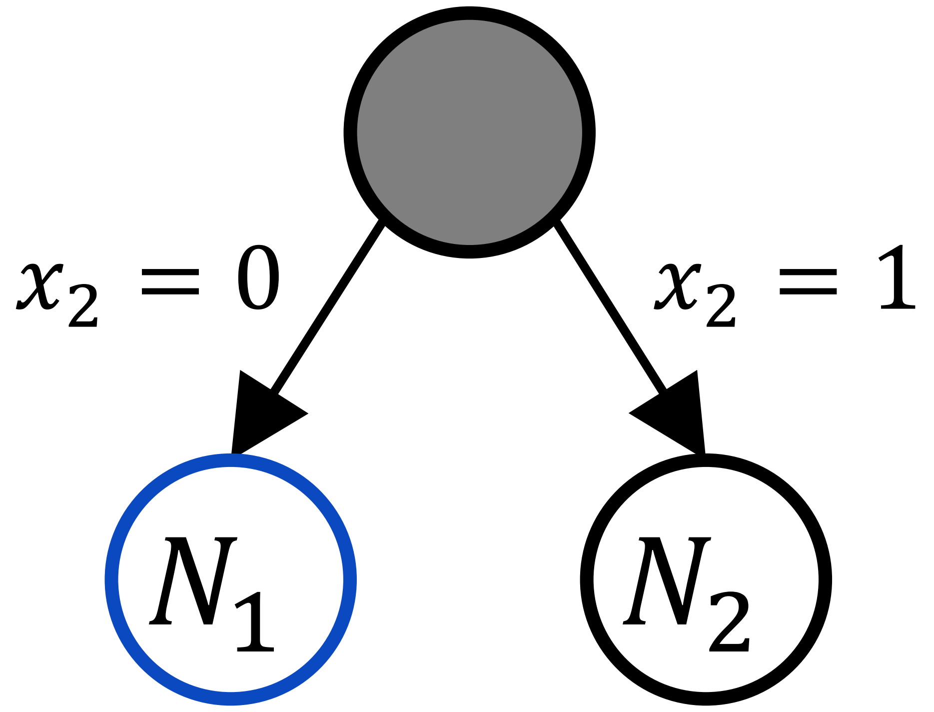
3 Select Node **[what selection rule?]**: Choose a node N_i to process from $\mathcal{L}$

4 Evaluate & Prune: Solve the LP relaxation of N_i and prune node if applicable.

5 Add Cuts **[which cuts to add?]**: new constraints that tighten the formulation.

6 Run Heuristics **[which heuristics to run?]**: try to find a better solution.

7 Select Branching Variable **[what selection rule?]**: Choose a variable that has fractional value in the LP solution of N_i. Create two new subproblems N_{i1} and N_{i2}. Go to line 2.



ML Paradigms

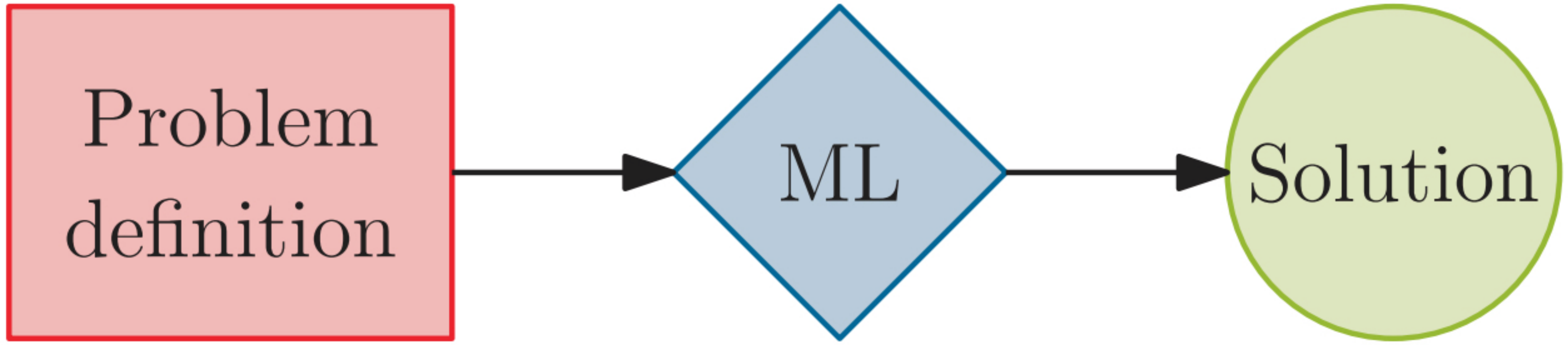


Fig. 7. Machine learning acts alone to provide a solution to the problem.

This is only viable for heuristics

Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost. “Machine learning for combinatorial optimization: a methodological tour d’horizon.” European Journal of Operational Research

ML Paradigms

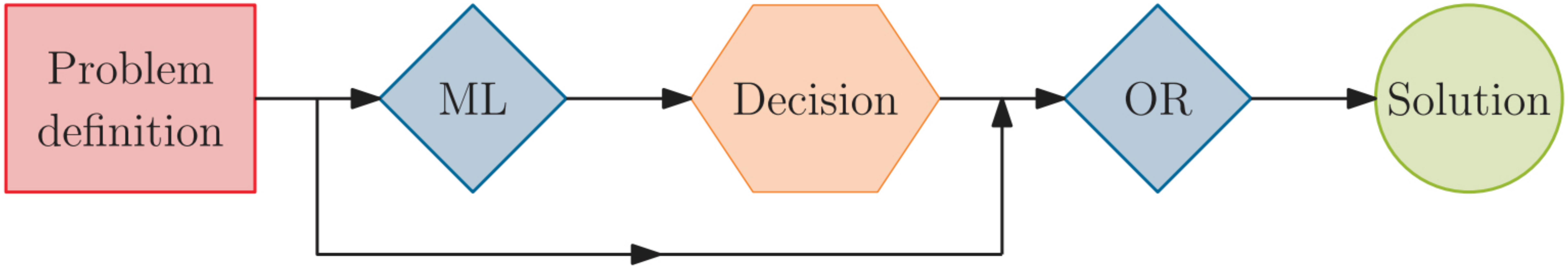


Fig. 8. The machine learning model is used to augment an operation research algorithm with valuable pieces of information.

e.g., Algorithm Configuration

Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d'horizon." European Journal of Operational Research

ML Paradigms

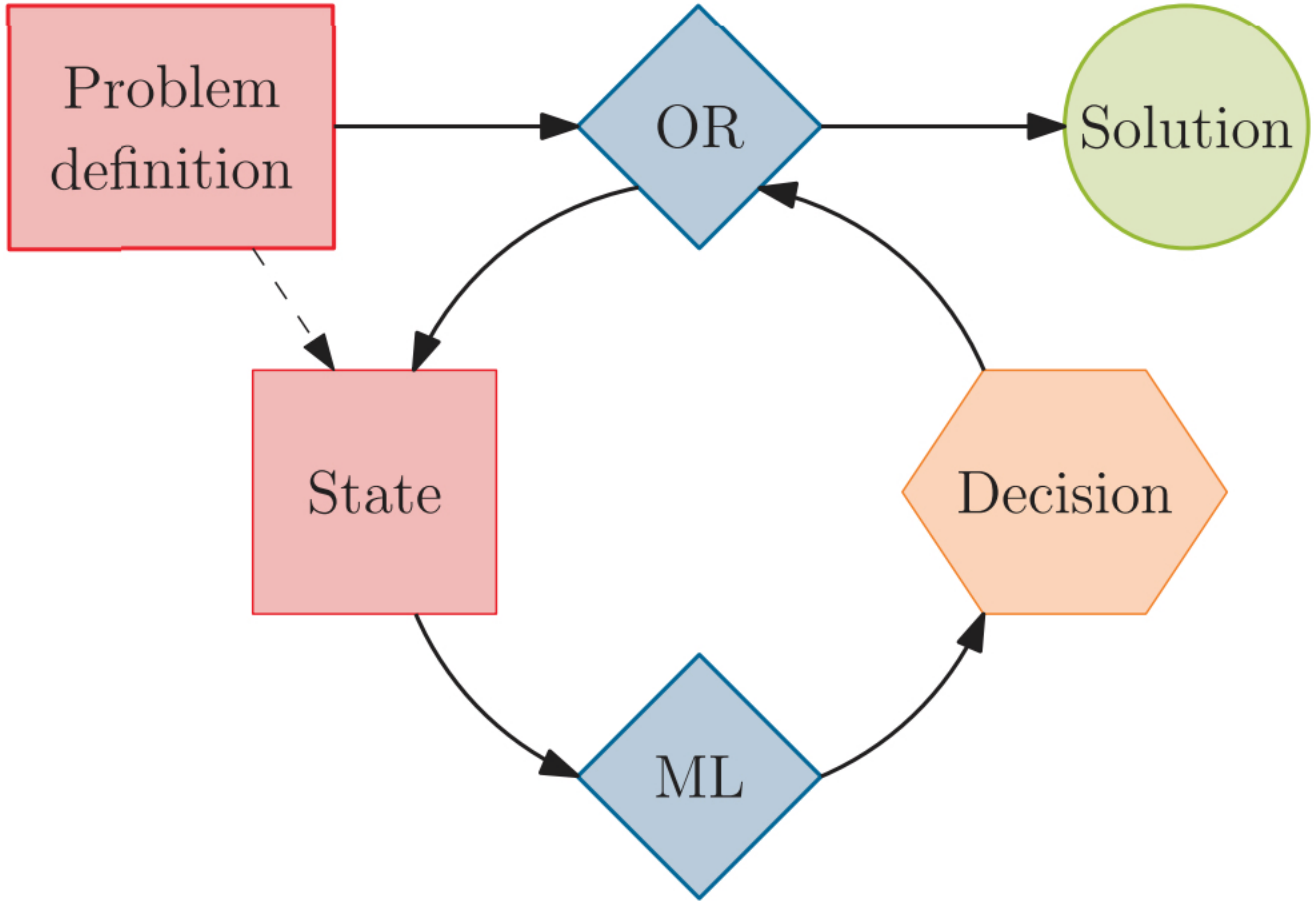


Fig. 9. The combinatorial optimization algorithm repeatedly queries the same machine learning model to make decisions. The machine learning model takes as input the current state of the algorithm, which may include the problem definition.

ML is infused within a bigger optimization algorithm

Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d'horizon." European Journal of Operational Research