Applied Mixed Integer Programming: Beyond 'The Optimum'

14 Nov 2016, Simons Institute, Berkeley
Pawel Lichocki
Operations Research Team, Google
https://developers.google.com/optimization/

Applied Mixed Integer Programming: Before 'The Optimum'

14 Nov 2016, Simons Institute, Berkeley
Pawel Lichocki
Operations Research Team, Google
https://developers.google.com/optimization/

Outline

- **Why do we use MIP?**
  - **Engineering**
  - **Efficiency**
- **Why are MIP solvers efficient?**
  - **Solver**
  - **Model**
- **Self-doubt**

Outline

Why do we use MIP?

- **Engineering**
- **Efficiency**

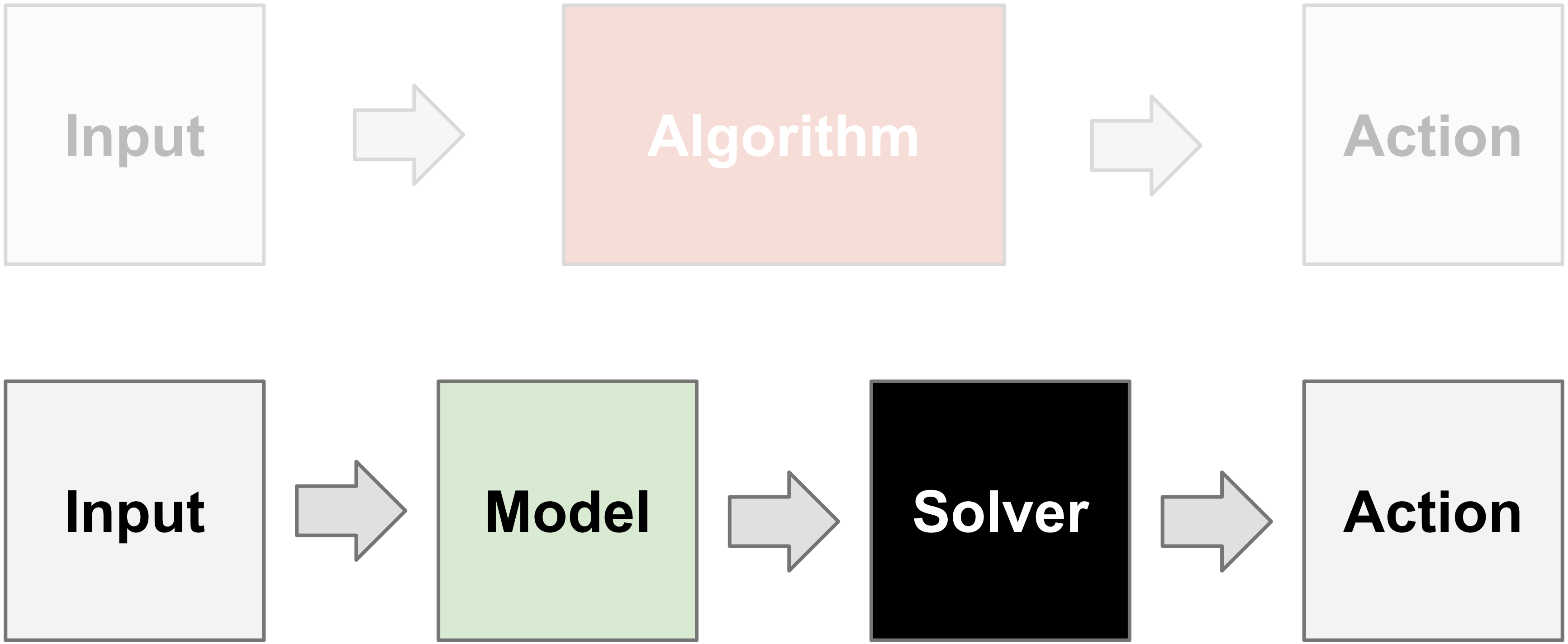
Why are MIP solvers efficient?

- **Solver**
- **Model**

Self-doubt

Imperative programming

Declarative programming



Mixed integer programming

$$\min c^T x$$
$$Ax \leq b$$
$$x \geq 0$$
$$x_j \in Z,\ j \in J$$

Mixed integer programming

$$\min c^T x$$
$$Ax \le b$$
$$x \ge 0$$
$$x_j \in Z,\ j \in J$$

⟺

$$\min/\max\ c_0 + c^T x$$
$$lb_{ct} \le Ax \le ub_{ct}$$
$$lb_{var} \le x \le ub_{var}$$
$$x_j \in Z,\ j \in J$$

Indices

```
Item i = 1..I
Bin b = 1..B
Resource r = 1..R
```

Variables

```
place(i, b) in {0, 1}
```

Constants

```
int Value(i)
double Required(i, r)
double Available(b, r)
```

Constraints

Objective

Indices

Item i = 1..I
Bin b = 1..B
Resource r = 1..R

Variables

place(i, b) in {0, 1}

Constants

int Value(i)
double Required(i, r)
double Available(b, r)

Constraints

for item i = 1..I:
$\sum_{b = 1..B}$ place(i, b) ≤ 1

for resource r = 1..R:
for bin b = 1..B:
$\sum_{i = 1..I}$ Required(i, r) * place(i, b) ≤ Available(b, r)

Objective

Indices

Item i = 1..I
Bin b = 1..B
Resource r = 1..R

Variables

place(i, b) in {0, 1}

Constants

int Value(i)
double Required(i, r)
double Available(b, r)

Constraints

for item i = 1..I:

$\sum_{b = 1..B}$ place(i, b) ≤ 1

for resource r = 1..R:

for bin b = 1..B:

$\sum_{i = 1..I}$ Required(i, r) * place(i, b) ≤ Available(b, r)

Objective

maximize $\sum_{i = 1..I}$ Value(i) * place(i, b)

Indices

Item i = 1..I
Bin b = 1..B
Resource r = 1..R

Variables

place(i, b) in **[0..Copies(i)]**

Constants

int **Copies(i)**
double Required(i, r)
double Available(b, r)

Constraints

for item i = 1..I:
$\sum_{b = 1..B}$ place(i, b) = **Copies(i)**

for resource r = 1..R:
for bin b = 1..B:
$\sum_{i = 1..I}$ Required(i, r) * place(i, b) ≤ Available(b, r)

Objective

Indices

Item $i = 1..I$
Bin $b = 1..B$
Resource $r = 1..R$

Variables

place(i, b) in [0..Copies(i)]
surplus(b) in [0, +inf)

Constants

int Copies(i)
double Required(i, r)
double Available(b, r)

Constraints

for item $i = 1..I$:

$$\sum_{b = 1..B} \text{place}(i, b) = \text{Copies}(i)$$

for resource $r = 1..R$:
for bin $b = 1..B$:

$$\sum_{i = 1..I} \text{Required}(i, r) * \text{place}(i, b) \leq \text{Available}(b, r)$$

for bin $b = 1..B$:

$$\sum_{i = 1..I} \text{Copies}(i) / B - \sum_{i = 1..I} \text{place}(i, b) \leq \text{surplus}(b)$$

Objective

$$\min \sum_{b = 1..B} \text{surplus}(b)$$

Indices

```
Item i = 1..I
Bin b = 1..B
Resource r = 1..R
```

Variables

```
place(i, b) in [0..Copies(i)]
surplus(b) in [0, +inf)
```
max_surplus in [0, +inf)

Constants

```
int Copies(i)
double Required(i, r)
double Available(b, r)
```

Constraints

for item i = 1..I:

$\sum_{b = 1..B}$ place(i, b) = Copies(i)

for resource r = 1..R:

for bin b = 1..B:

$\sum_{i = 1..I}$ Required(i, r) * place(i, b) ≤ Available(b, r)

for bin b = 1..B:

$\sum_{i = 1..I}$ Copies(i) / B - $\sum_{i = 1..I}$ place(i, b) ≤ surplus(b)

surplus(b) ≤ max_surplus

Objective

min **1e6 max_surplus** + $\sum_{b = 1..B}$ surplus(b)

Indices

```
Item i = 1..I
Bin b = 1..B
Resource r = 1..R
```

Variables

```
place(i, b) in [0..Copies(i)]
surplus(b) in [0, +inf)
max_surplus in [0, +inf)
```
diff(i, b) in [0, Copies(i)]

Constants

```
int Copies(i)
double Required(i, r)
double Available(b, r)
```
int Placed(i, b)

Constraints

for item i = 1..I:

$\sum_{b = 1..B}$ place(i, b) = Copies(i)

for resource r = 1..R:

for bin b = 1..B:

$\sum_{i = 1..I}$ Required(i, r) * place(i, b) ≤ Available(b, r)

for bin b = 1..B:

$\sum_{i = 1..I}$ Copies(i) / B - $\sum_{i = 1..I}$ place(i, b) ≤ surplus(b)

surplus(b) ≤ max_surplus

for item i = 1..I:

Placed(i, b) - place(i, b) ≤ diff(i, b)

Objective

min 1e6 max_surplus + $\sum_{b = 1..B}$ surplus(b) + **1e-3** $\sum_{b = 1..B}\sum_{i = 1..I}$ **diff(i, b)**

Easy to maintain and add new features

```
Indices                 Variables                             Constants
  Item i = 1..I            place(i, b) in [0..Copies(i)]          int Copies(i)
  Bin b                    surplus(b) in [0, +inf)                double Required(i, r)
  Reso                        surplus in [0, +inf)                double Available(b, r)
                                              ies(i)]             int Placed(i, b)
Constr
  for
    ∑

  for resource r = 1..R:
    for bin b = 1..B:
      ∑i = 1..I Required(i, r) * place(i, b) ≤ Available(b, r)

  for bin b = 1..B:
    ∑i = 1..I Copies(i) / B - ∑i = 1..I place(i, b) ≤ surplus(b)
    surplus(b) ≤ max_surplus
    for item i = 1..I:
      Placed(i, b) - place(i, b) ≤ diff(i, b)

Objective
  min 1e6 max_surplus + ∑b = 1..B surplus(b) + 1e-3 ∑b = 1..B ∑i = 1..I diff(i, b)
```

Indices

```
Item i = 1..I
Bin b ... 1..B
Resou...
```

Variables

```
place(i, b) in [0..Copies(i)]
surplus(b) in [0, +inf)
...surplus in [0, +inf)
...ies(i)]
```

Constants

```
int Copies(i)
double Required(i, r)
double Available(b, r)
int Placed(i, b)
```

Constr...

```
for ...
  ∑...

for resource r = 1..R:
  for ...
    ∑...                         ...) ≤ Available(b, r)

for ...
  ∑i...
  surplus(...)
  for item i = 1..I:
    Placed(i, b) - place(i, b) ≤ diff(i, b)
```

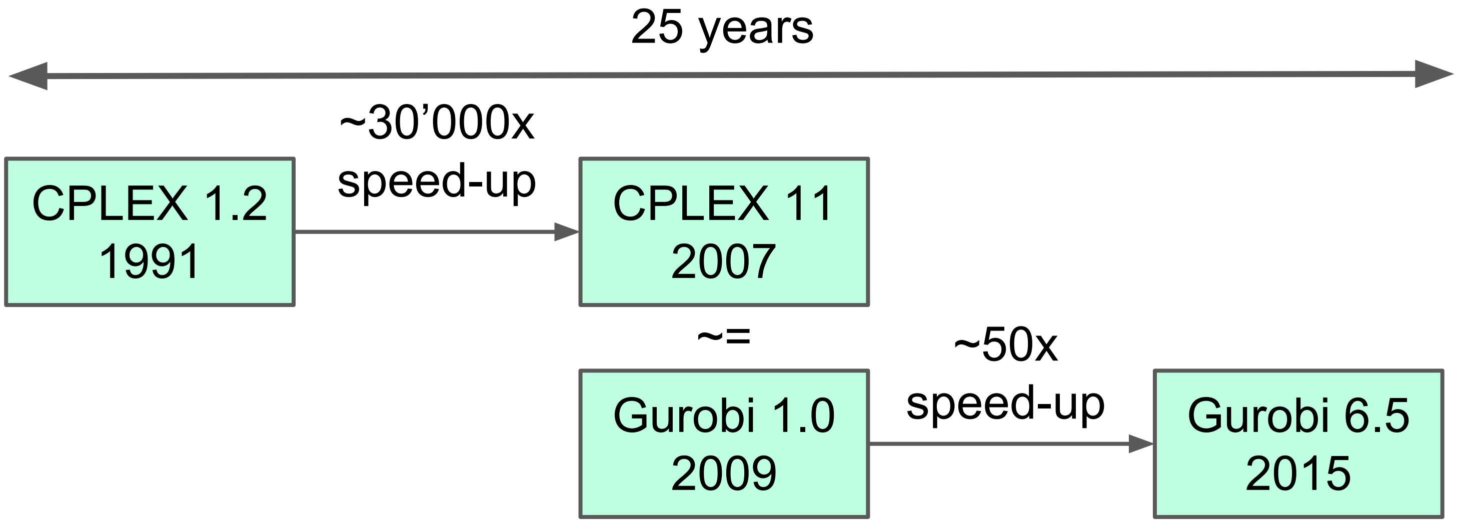
Objective

$$\min 1e6\ max\_surplus + \sum_{b = 1..B} surplus(b) + 1e\text{-}3 \sum_{b = 1..B} \sum_{i = 1..I} diff(i, b)$$

Easy to maintain and add new features

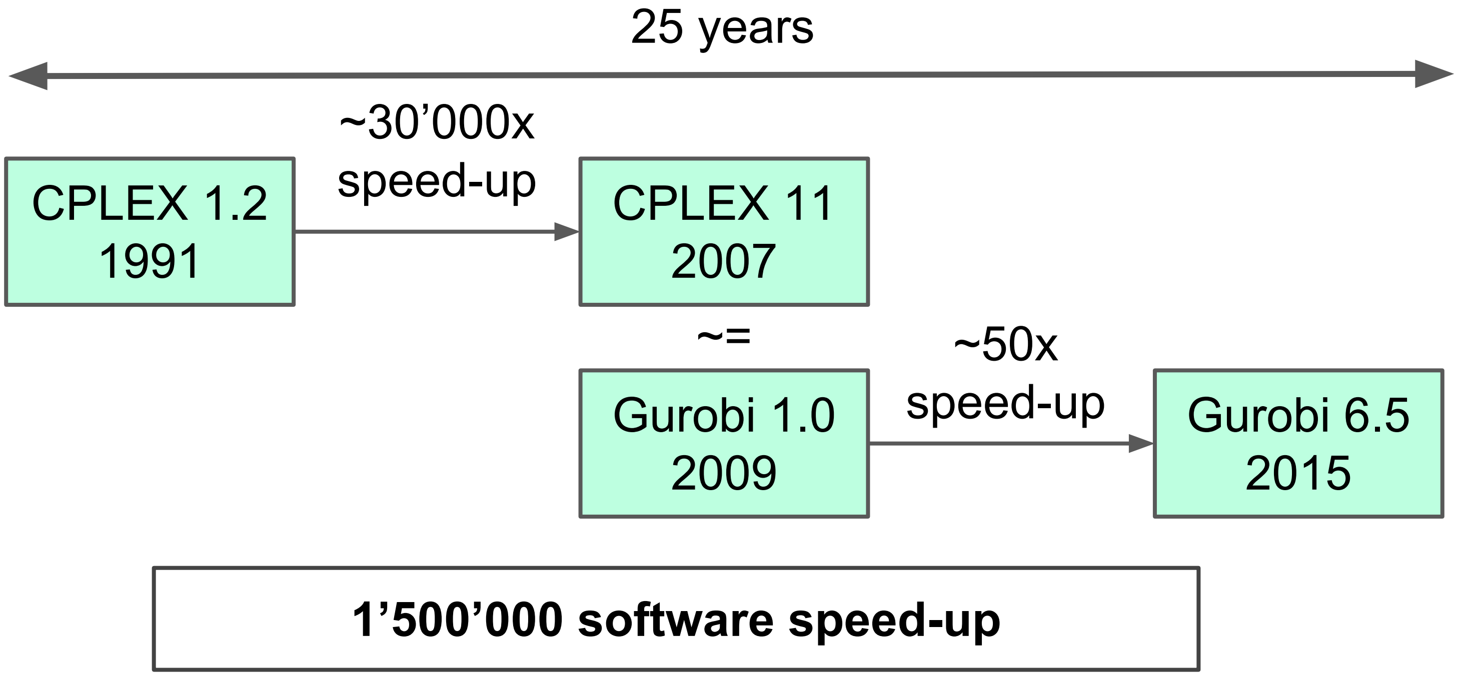
Minimalistic, yet very expressive

Efficiency



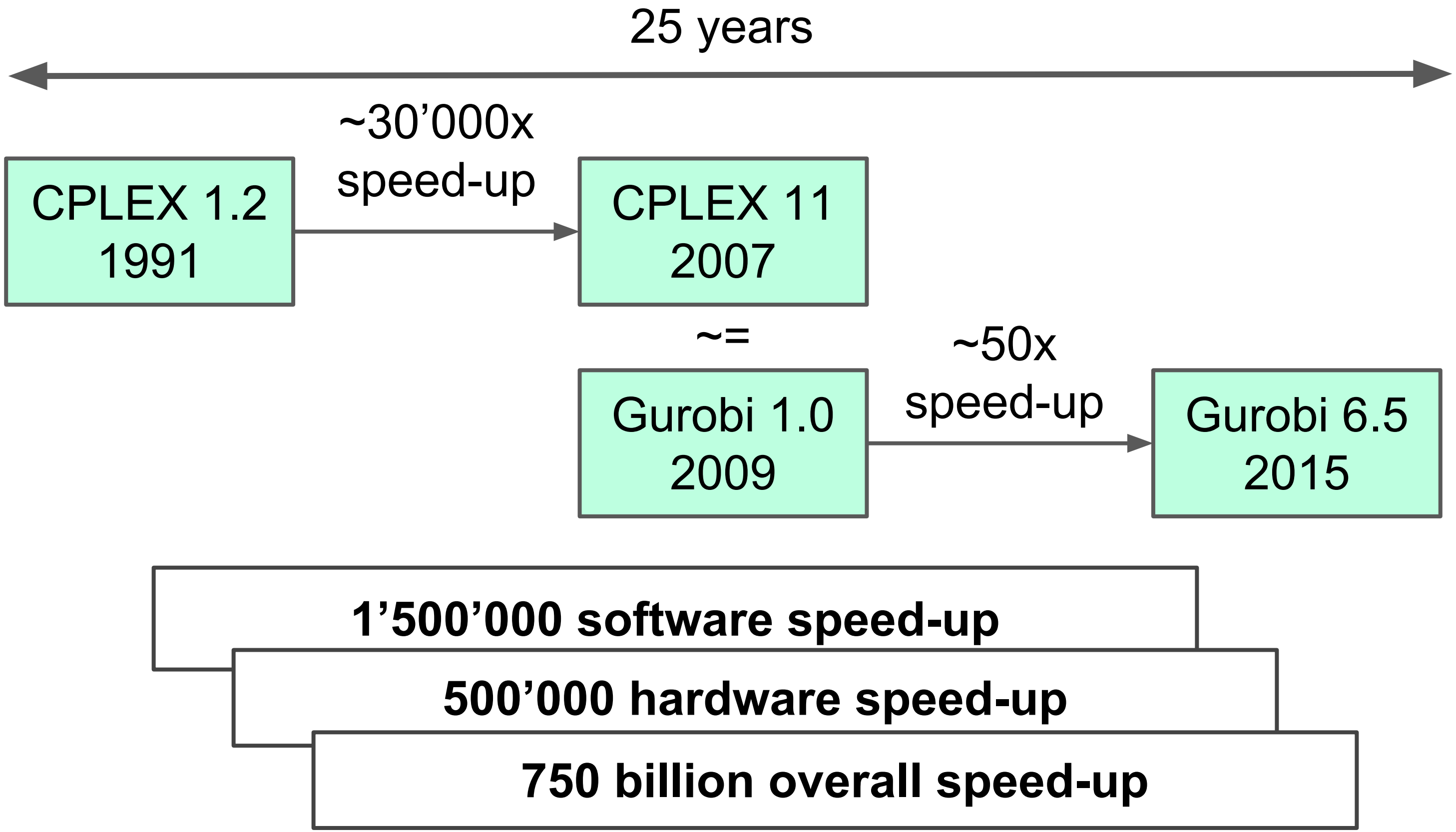
Bertsimas, D. (2016) “Machine Learning and Statistics via a modern optimization lens”, EURO’16, Poznan, Poland

Efficiency

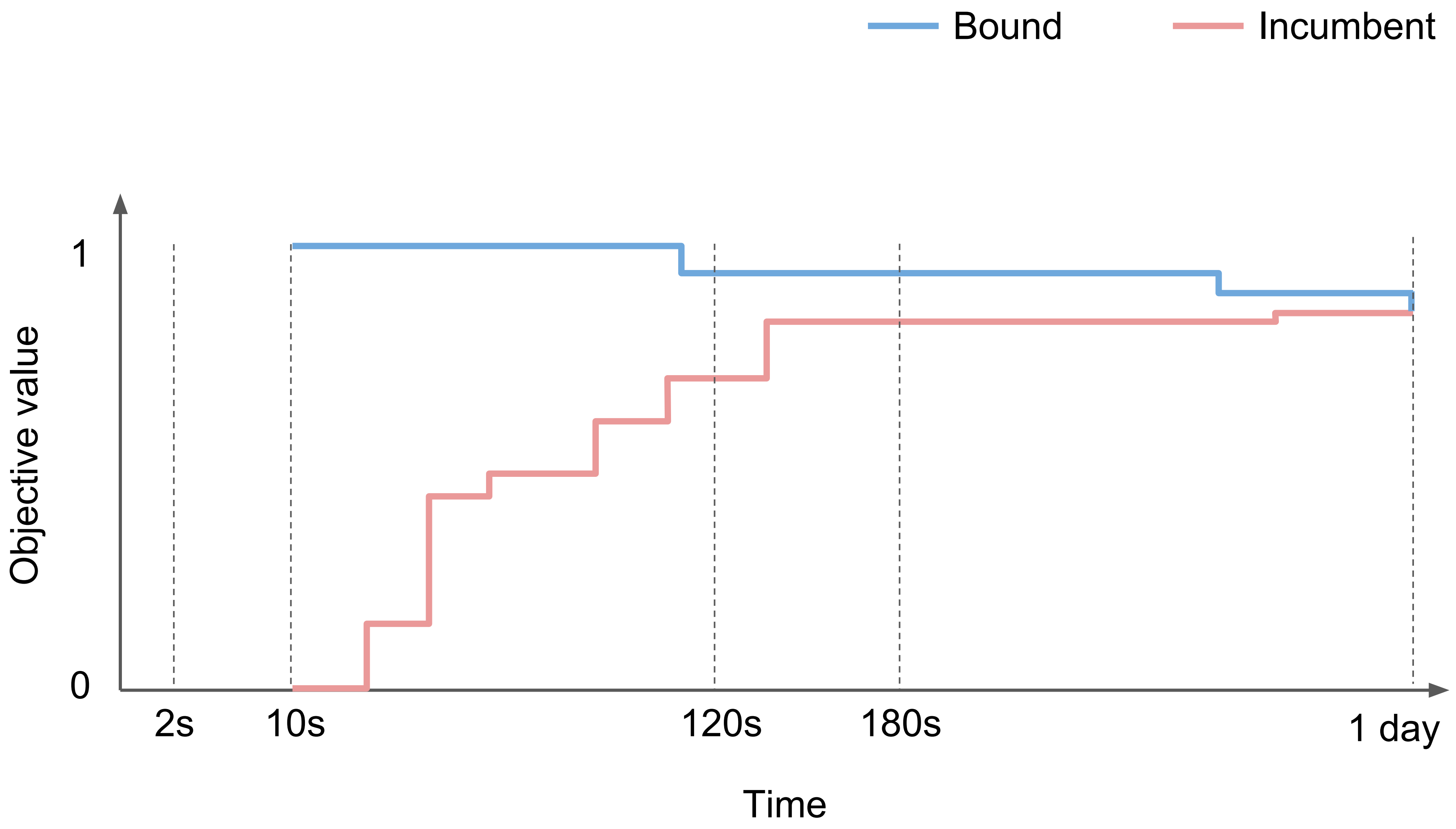


Bertsimas, D. (2016) "Machine Learning and Statistics via a modern optimization lens", EURO'16, Poznan, Poland

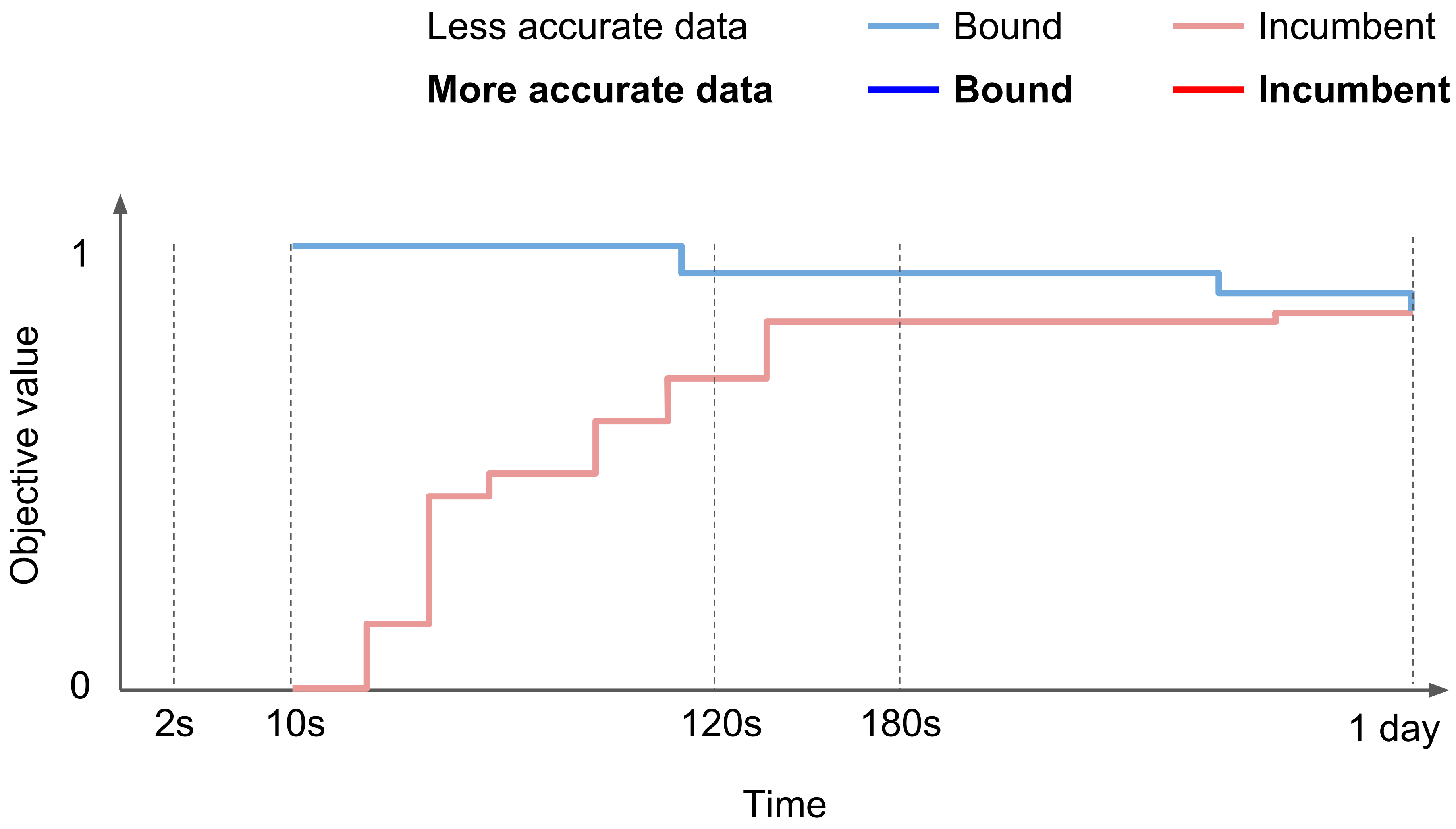
Efficiency

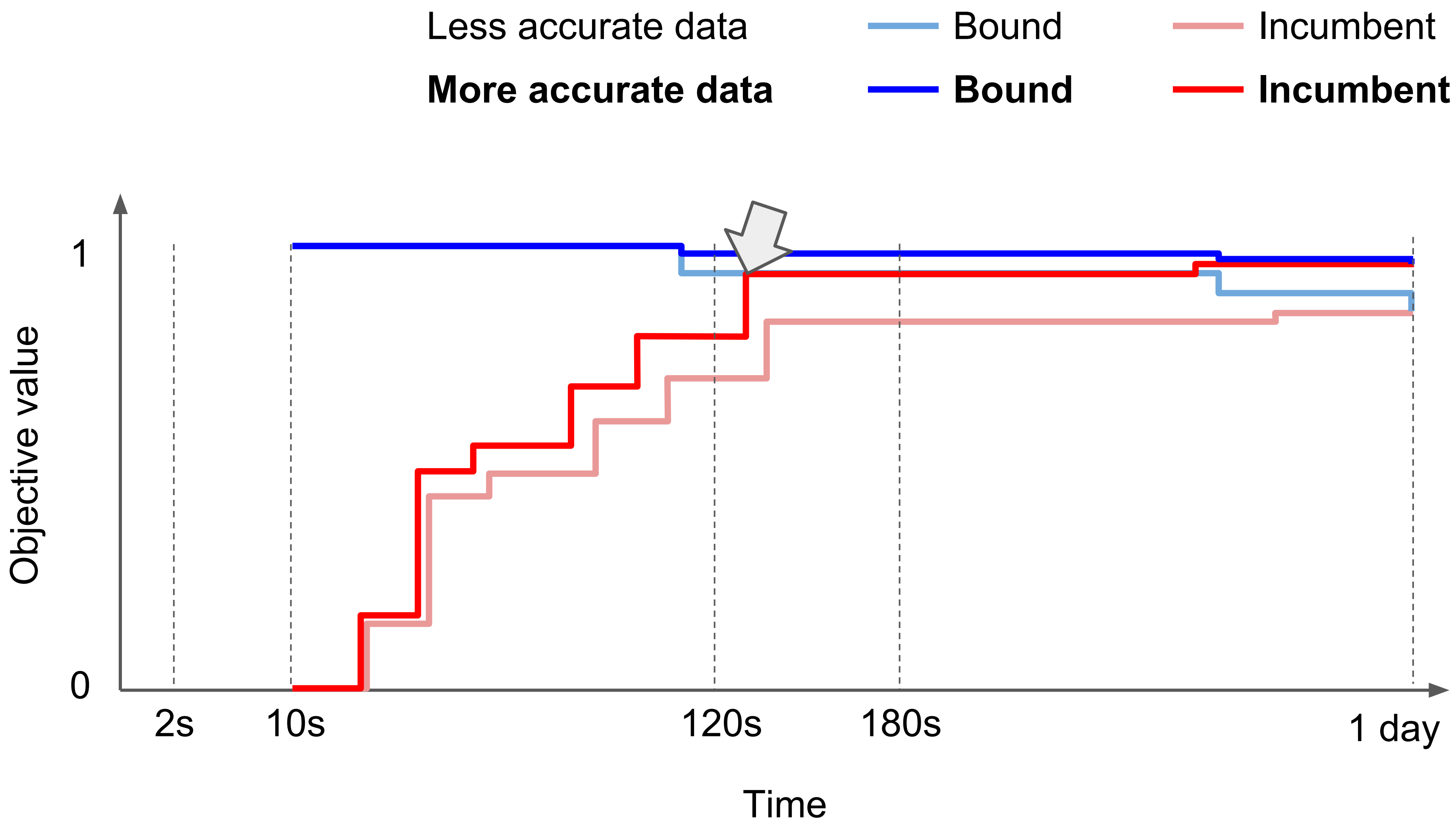


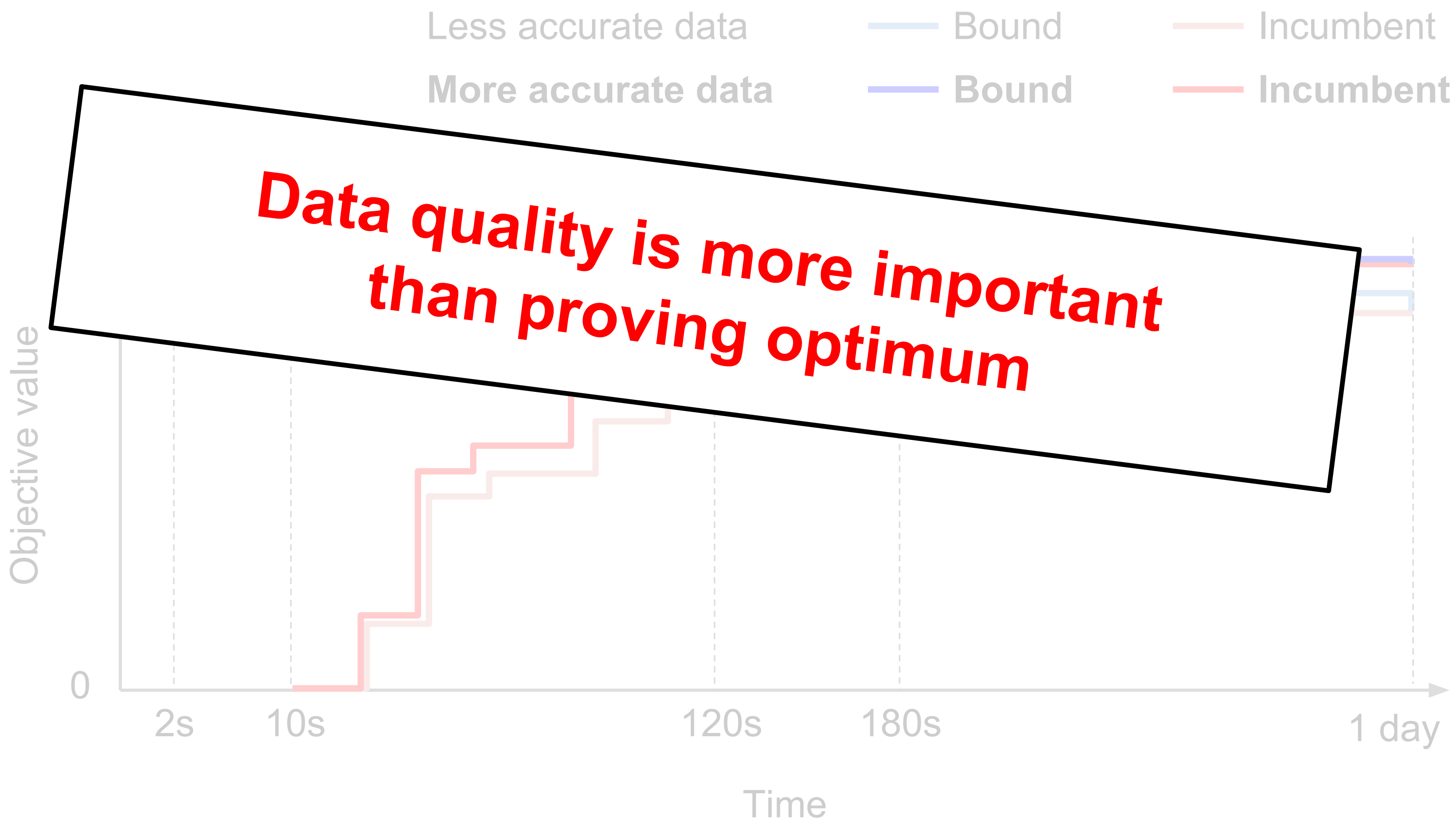
Bertsimas, D. (2016) “Machine Learning and Statistics via a modern optimization lens”, EURO’16, Poznan, Poland

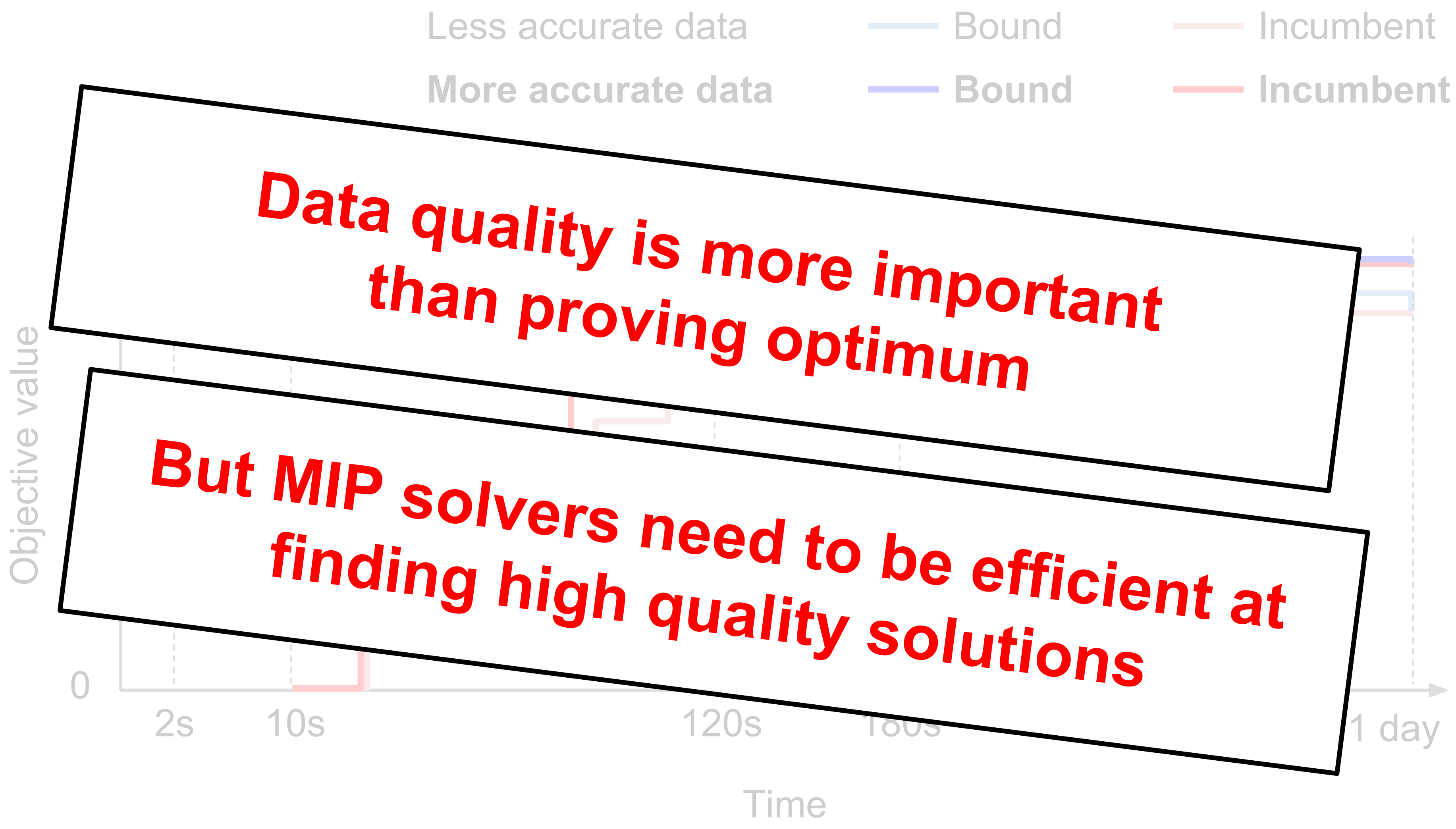


Bound
Incumbent
Objective value
1
0
2s
10s
120s
180s
1 day
Time


Less accurate data
Bound
Incumbent
More accurate data
Bound
Incumbent
1
0
Objective value
2s
10s
120s
180s
1 day
Time


Less accurate data
Bound
Incumbent
More accurate data
Bound
Incumbent
Objective value
1
0
2s
10s
120s
180s
1 day
Time


Less accurate data
Bound
Incumbent
More accurate data
Bound
Incumbent
Data quality is more important than proving optimum
Objective value
0
2s
10s
120s
180s
1 day
Time



Data quality is more important than proving optimum

But MIP solvers need to be efficient at finding high quality solutions

Outline

Why do we use MIP?

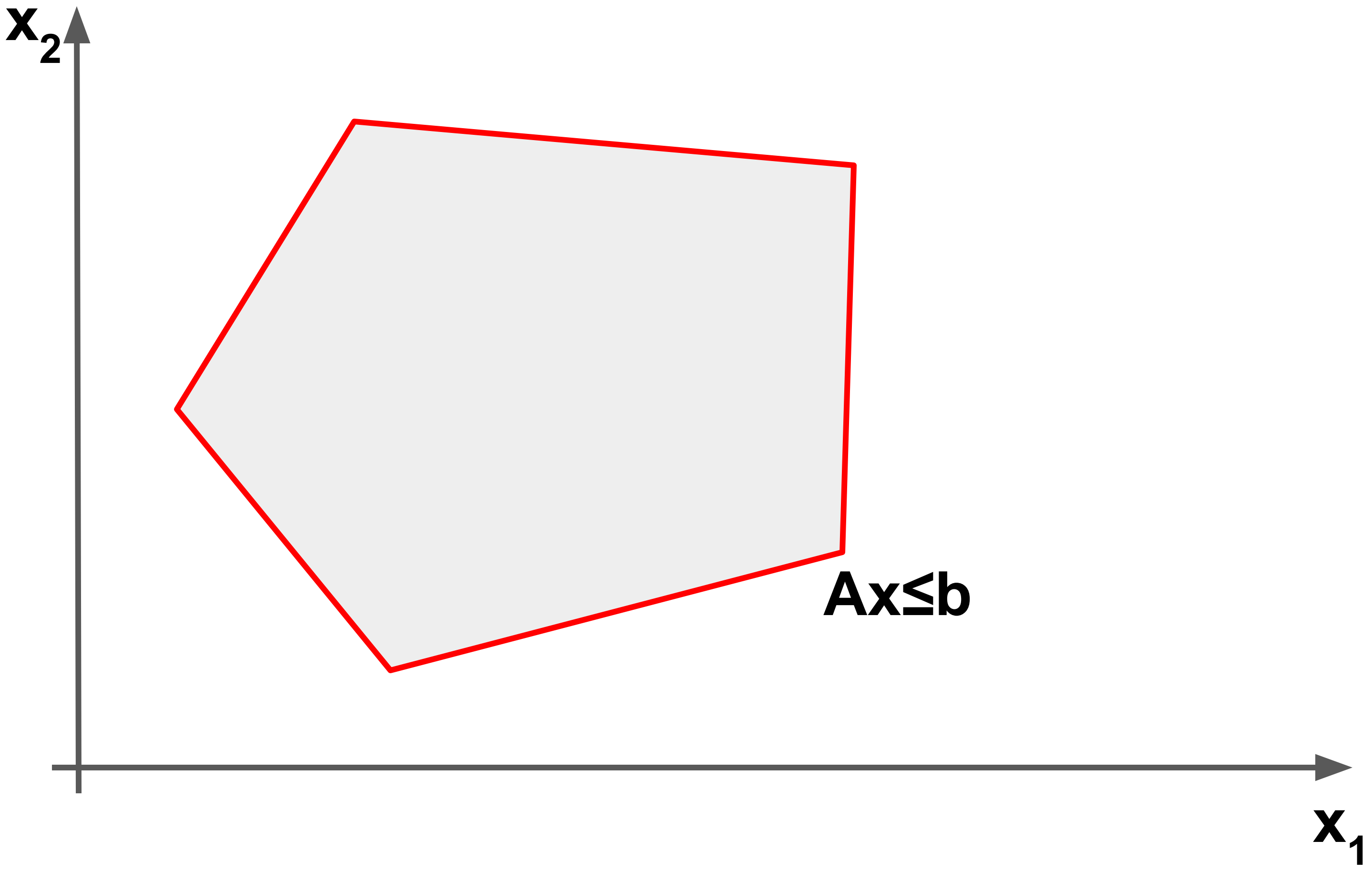
- Engineering
- Efficiency

Why are MIP solvers efficient?

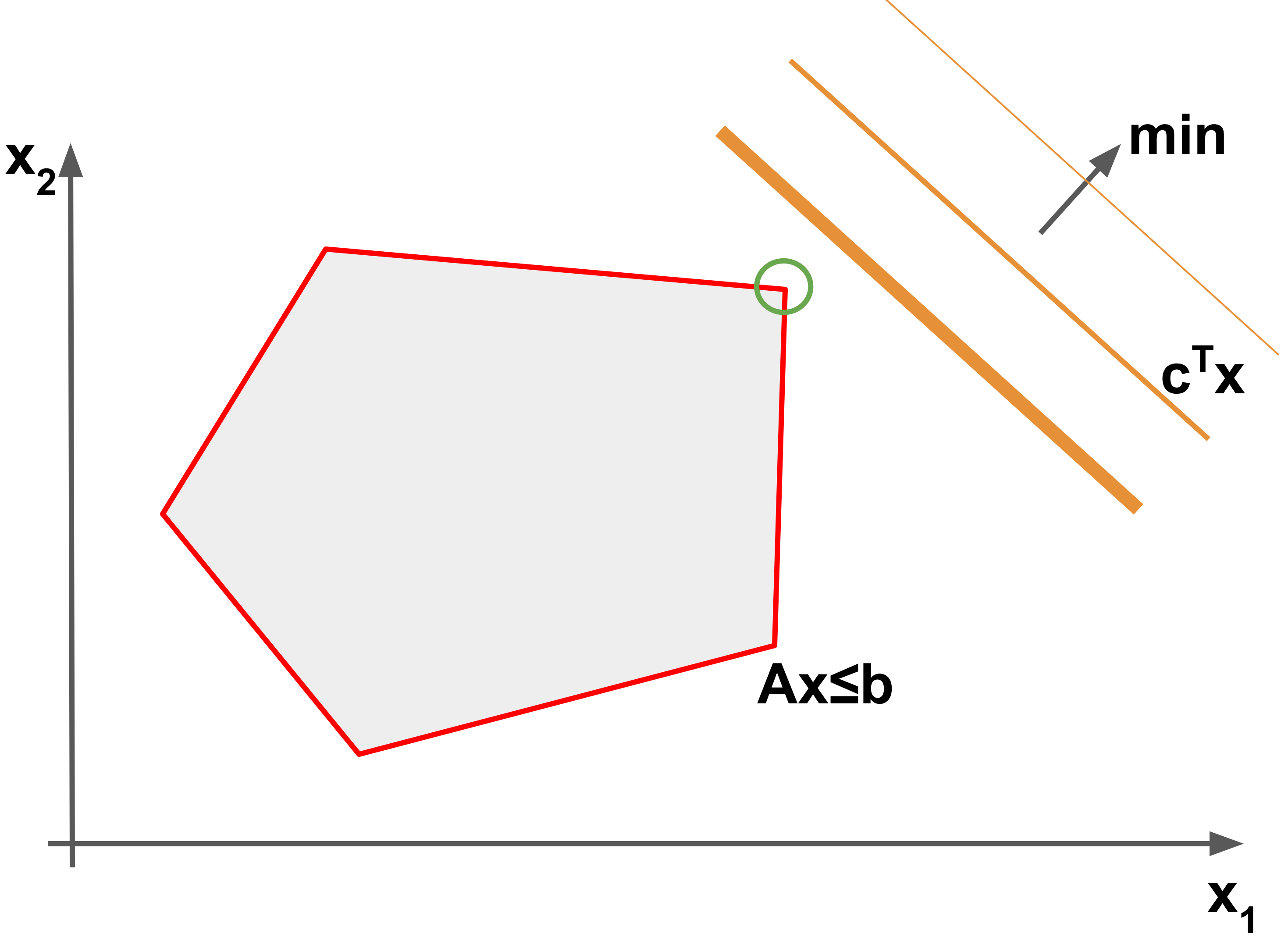
- **Solver**
- **Model**

Self-doubt

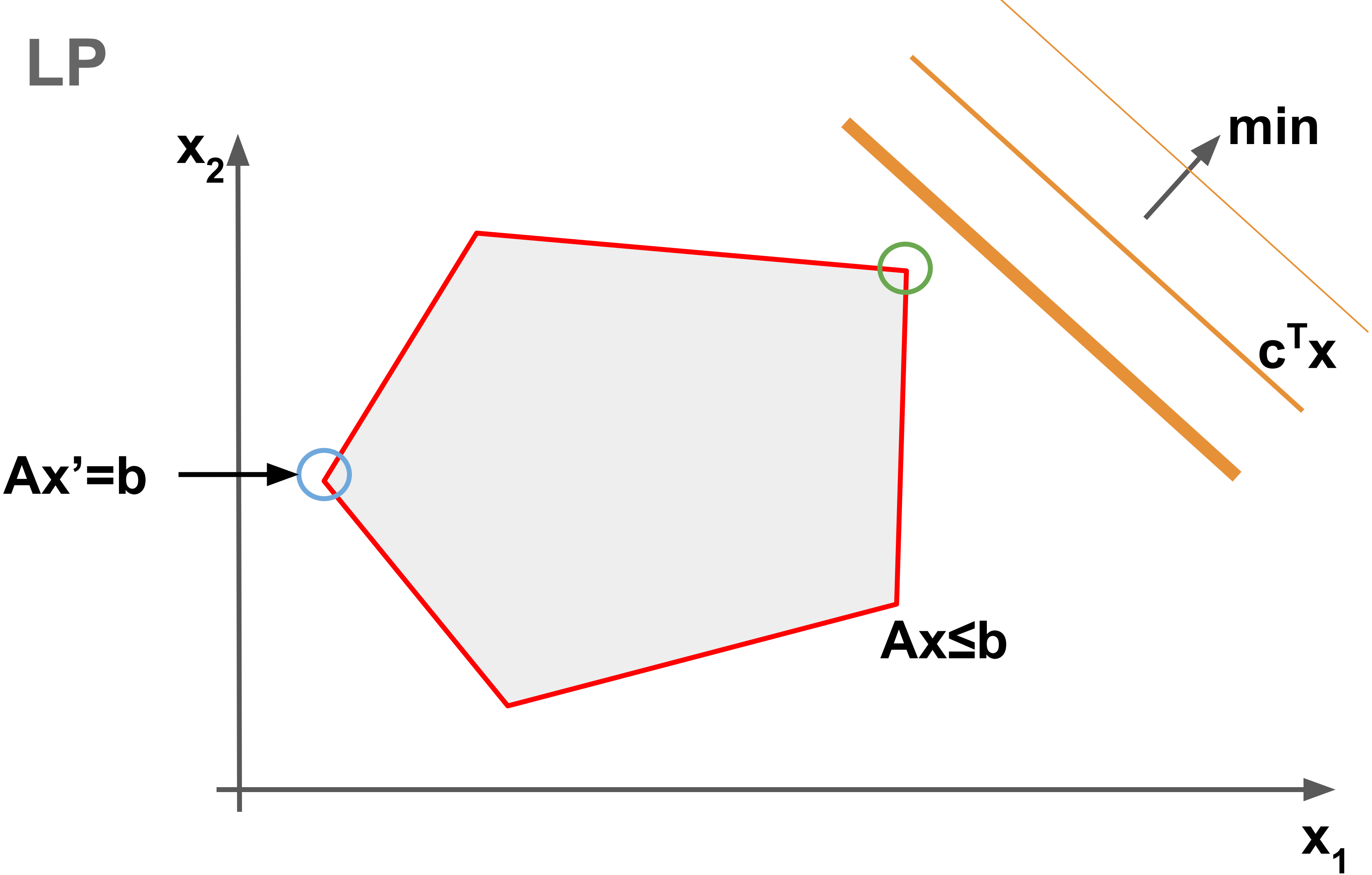
LP



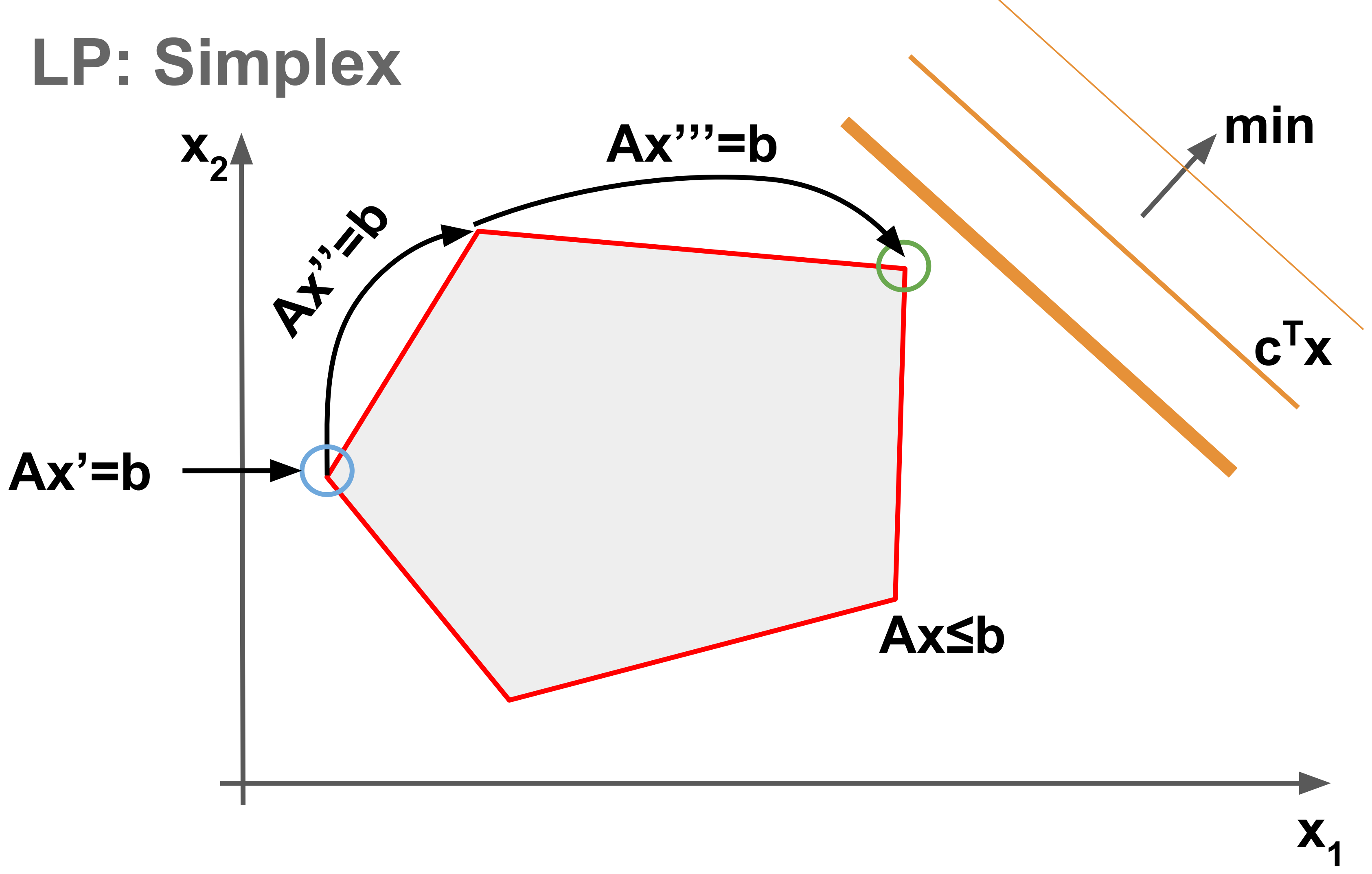
LP



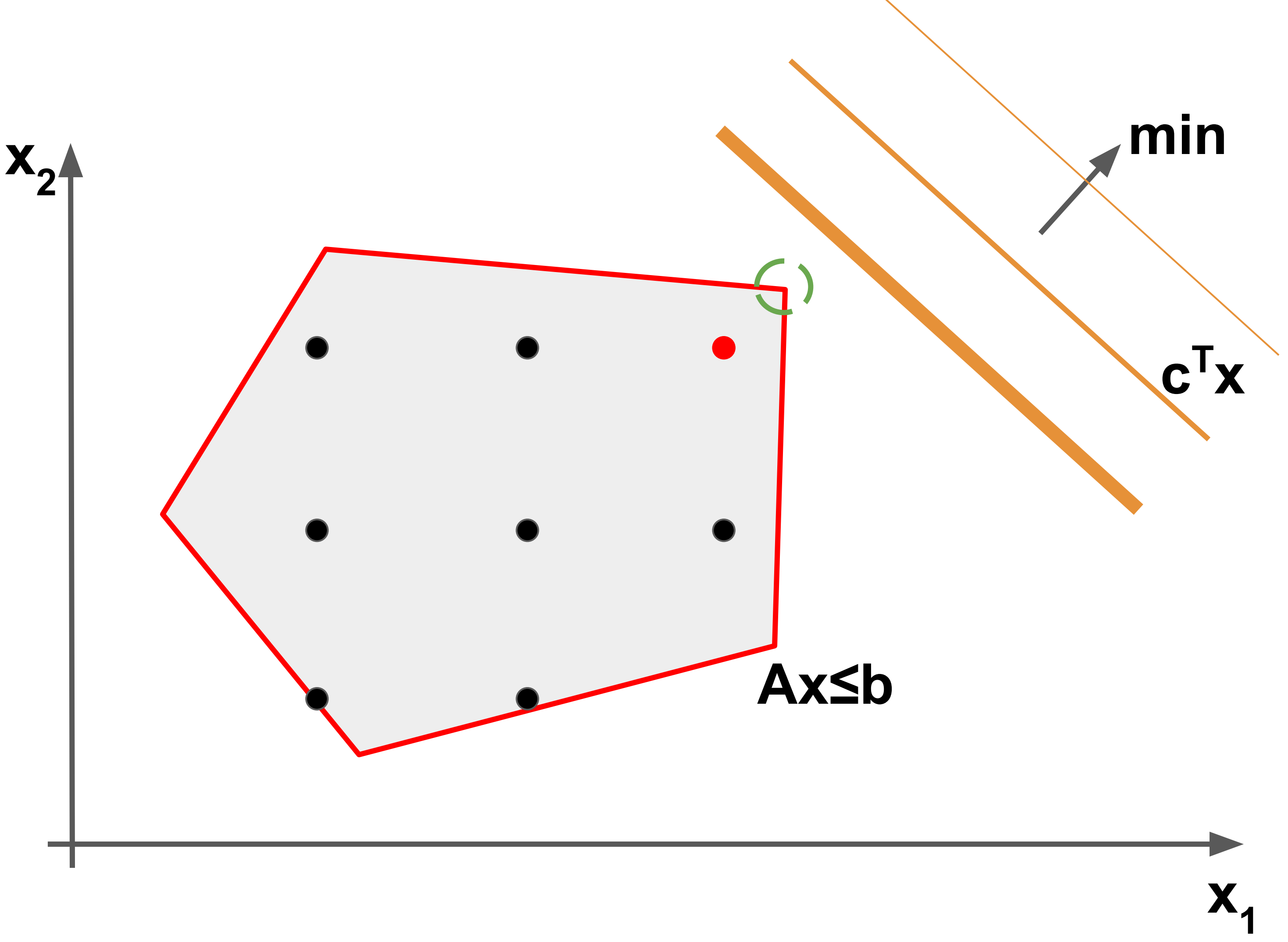
LP



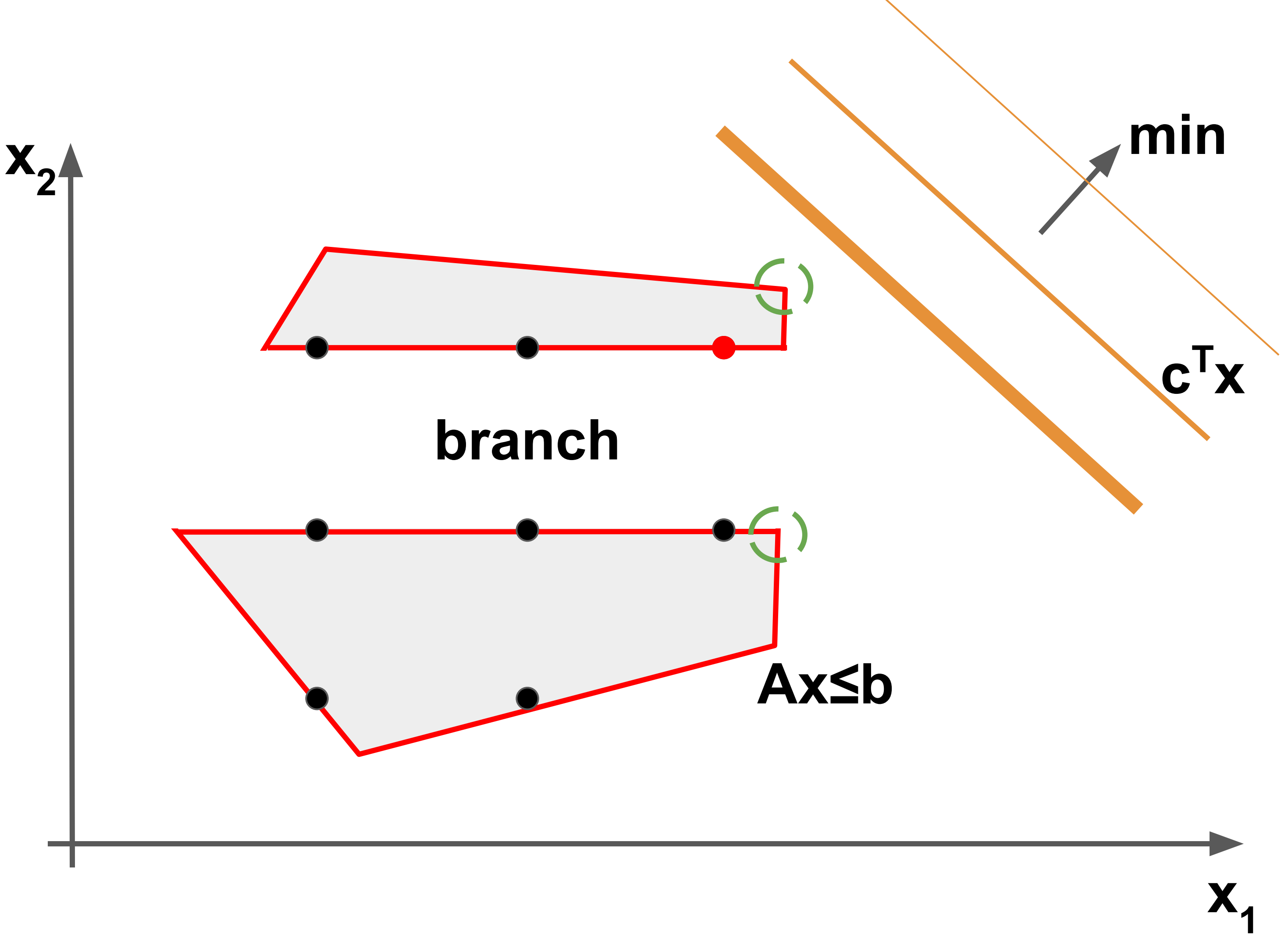
LP: Simplex



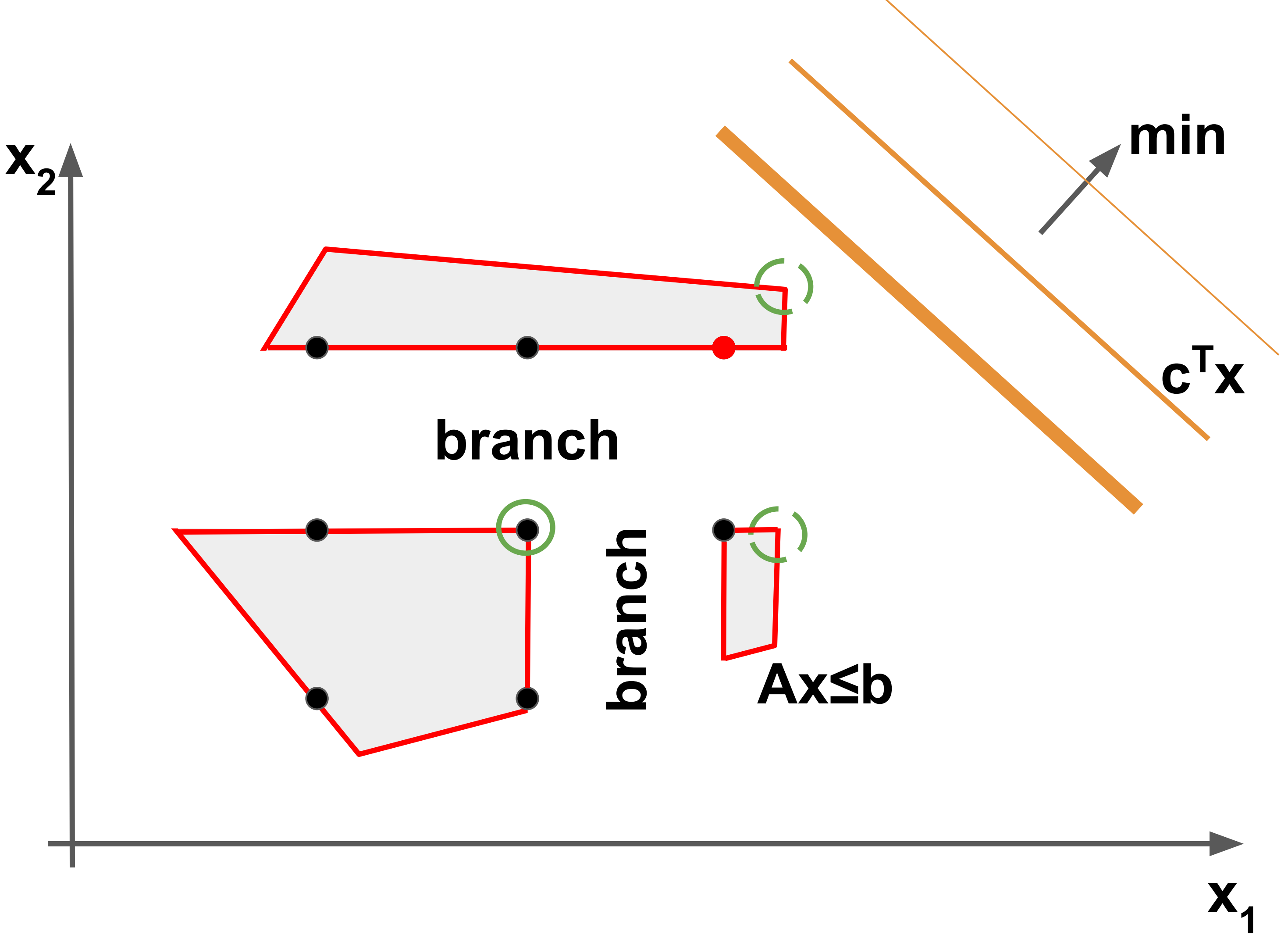
MIP



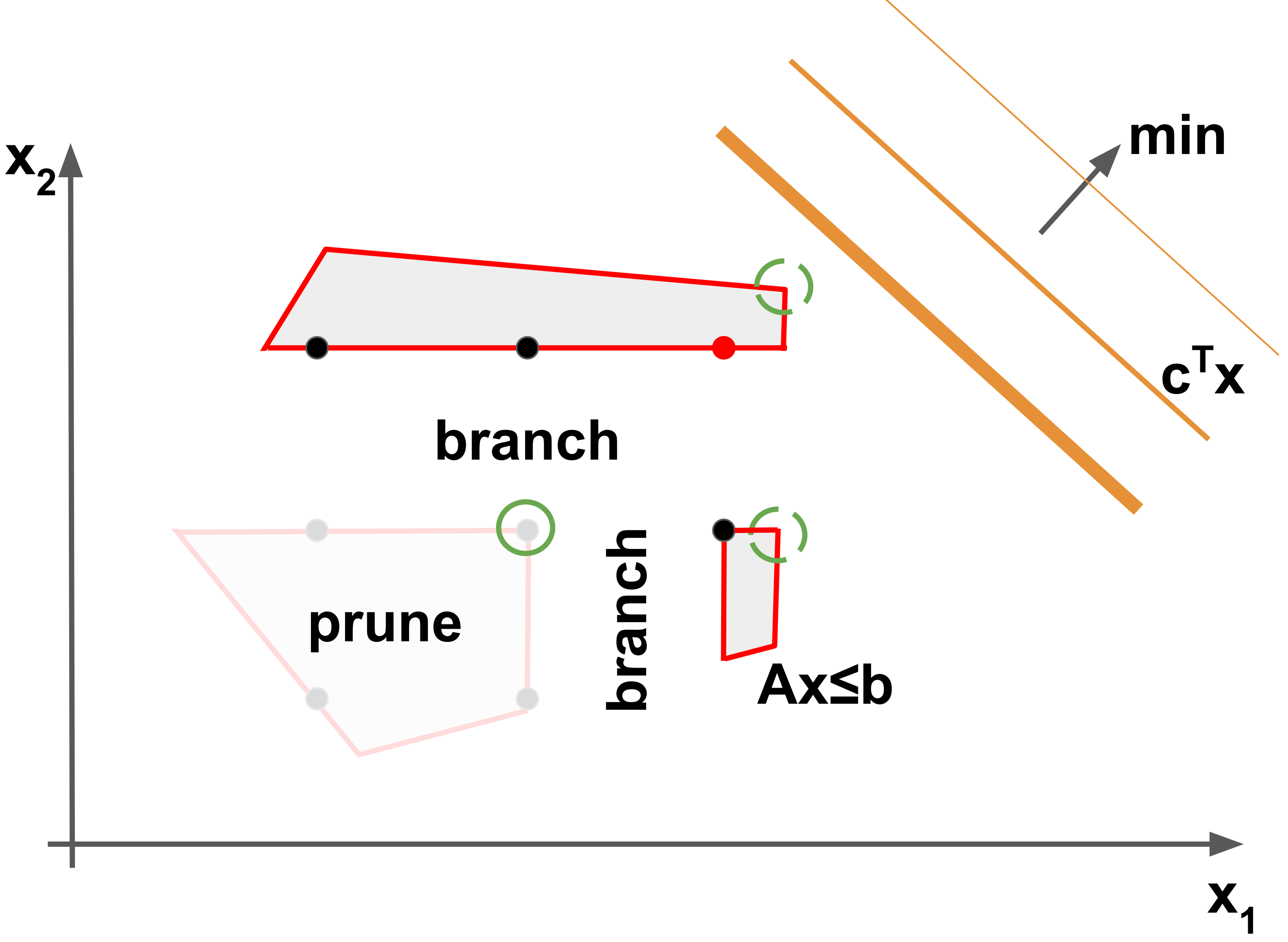
MIP



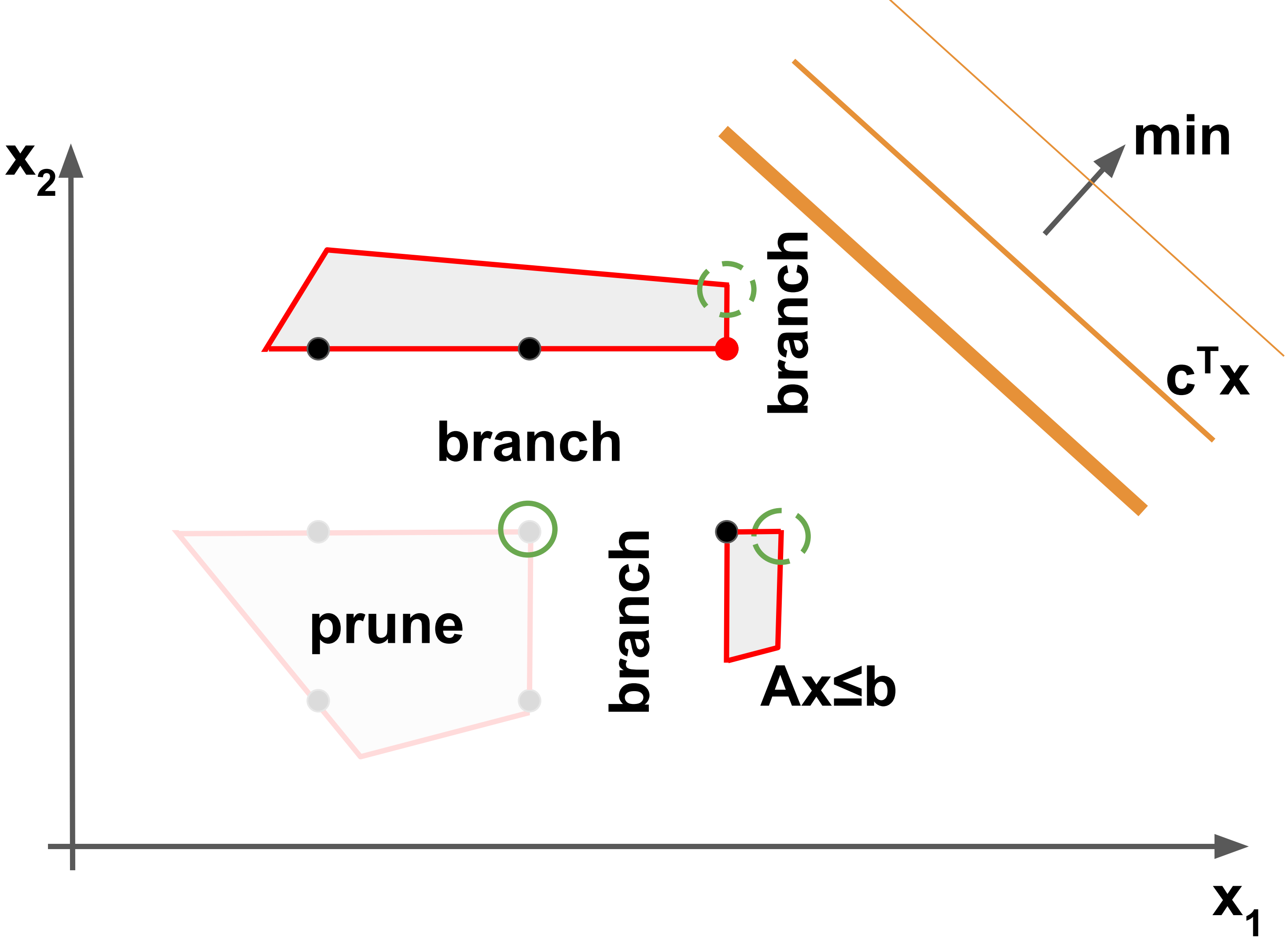
MIP



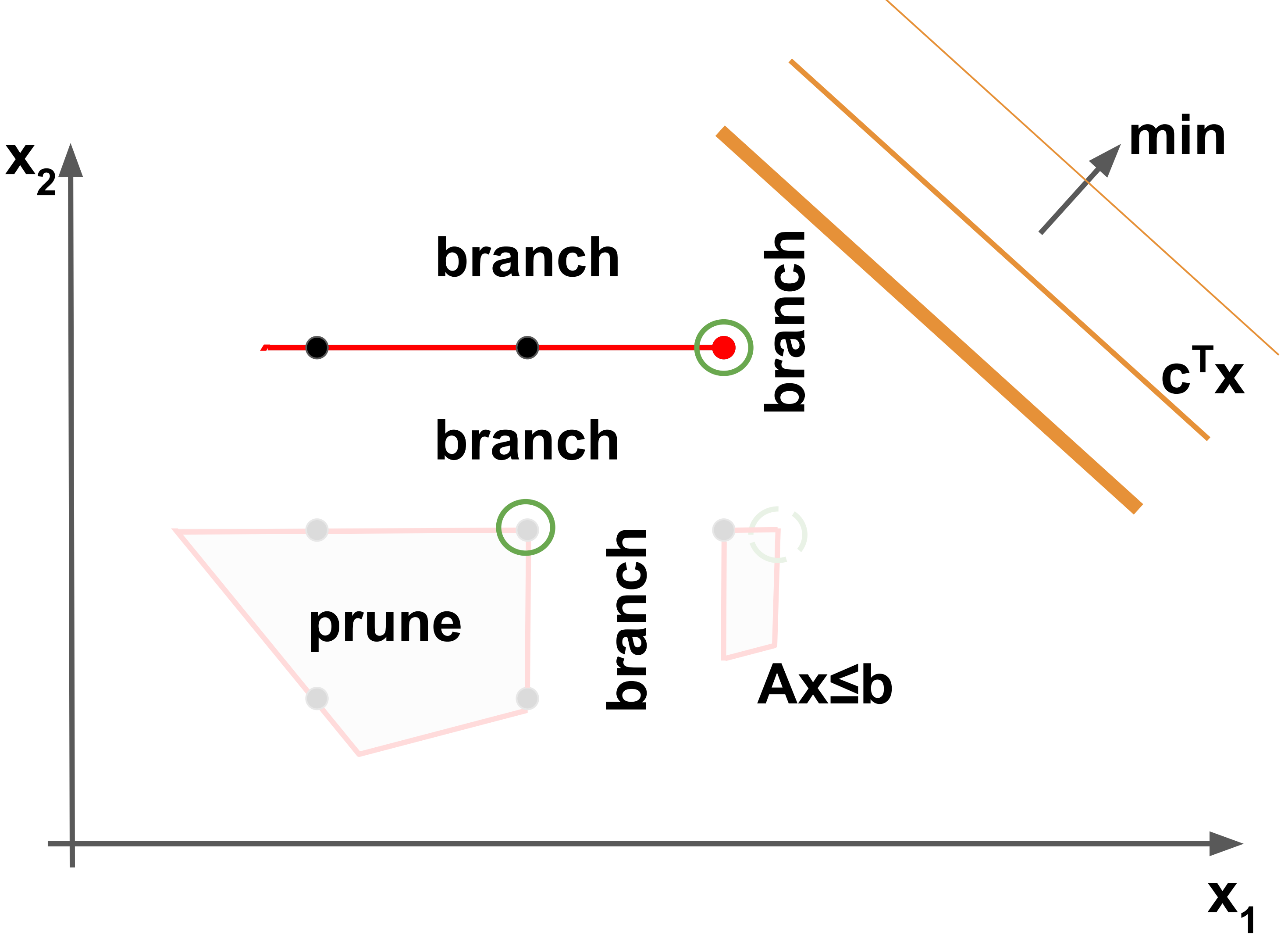
MIP



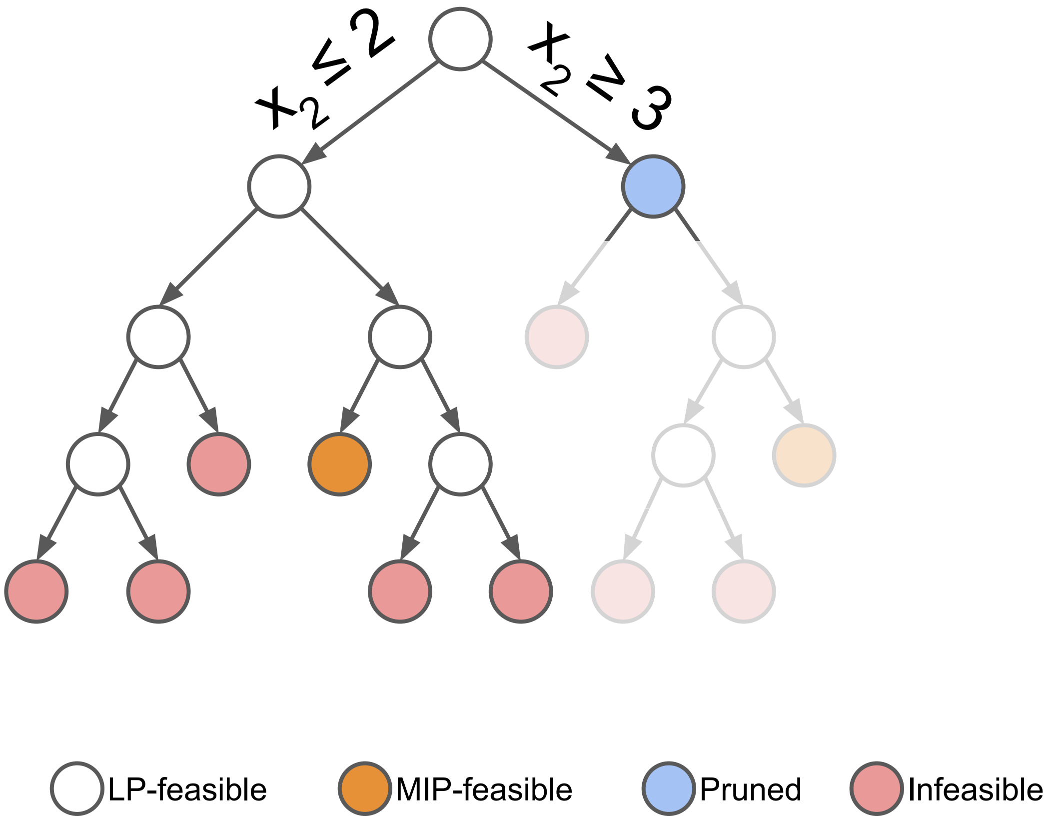
MIP



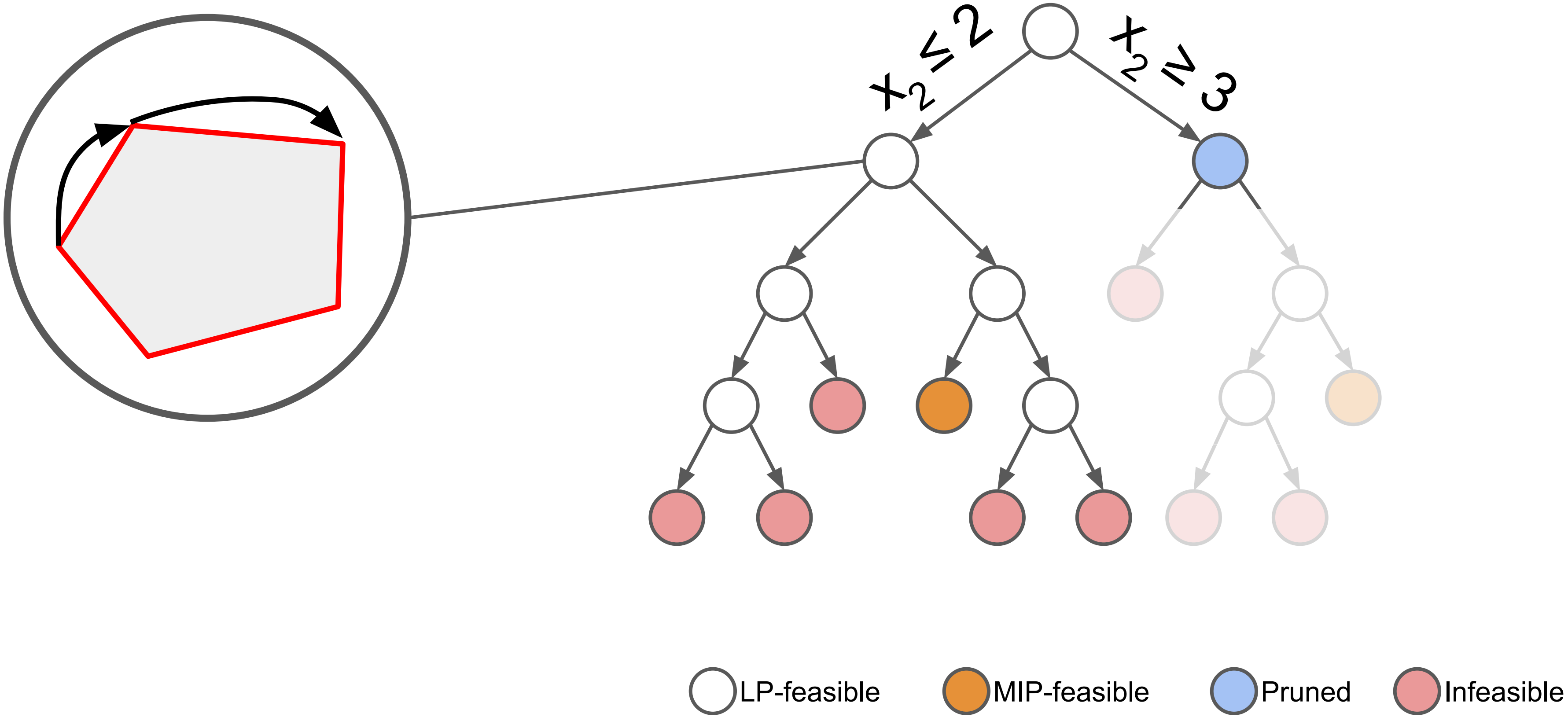
MIP



MIP: branch-and-bound



MIP: branch-and-bound + simplex

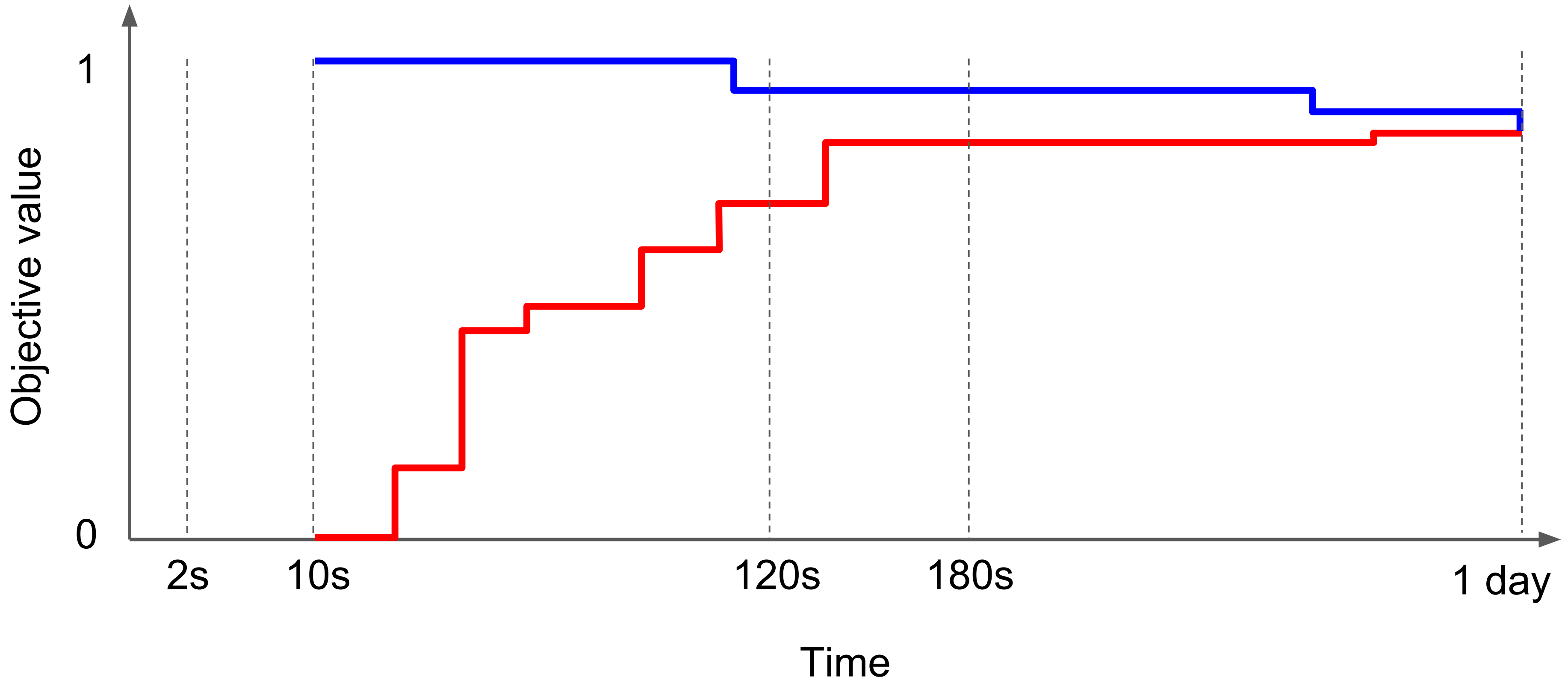


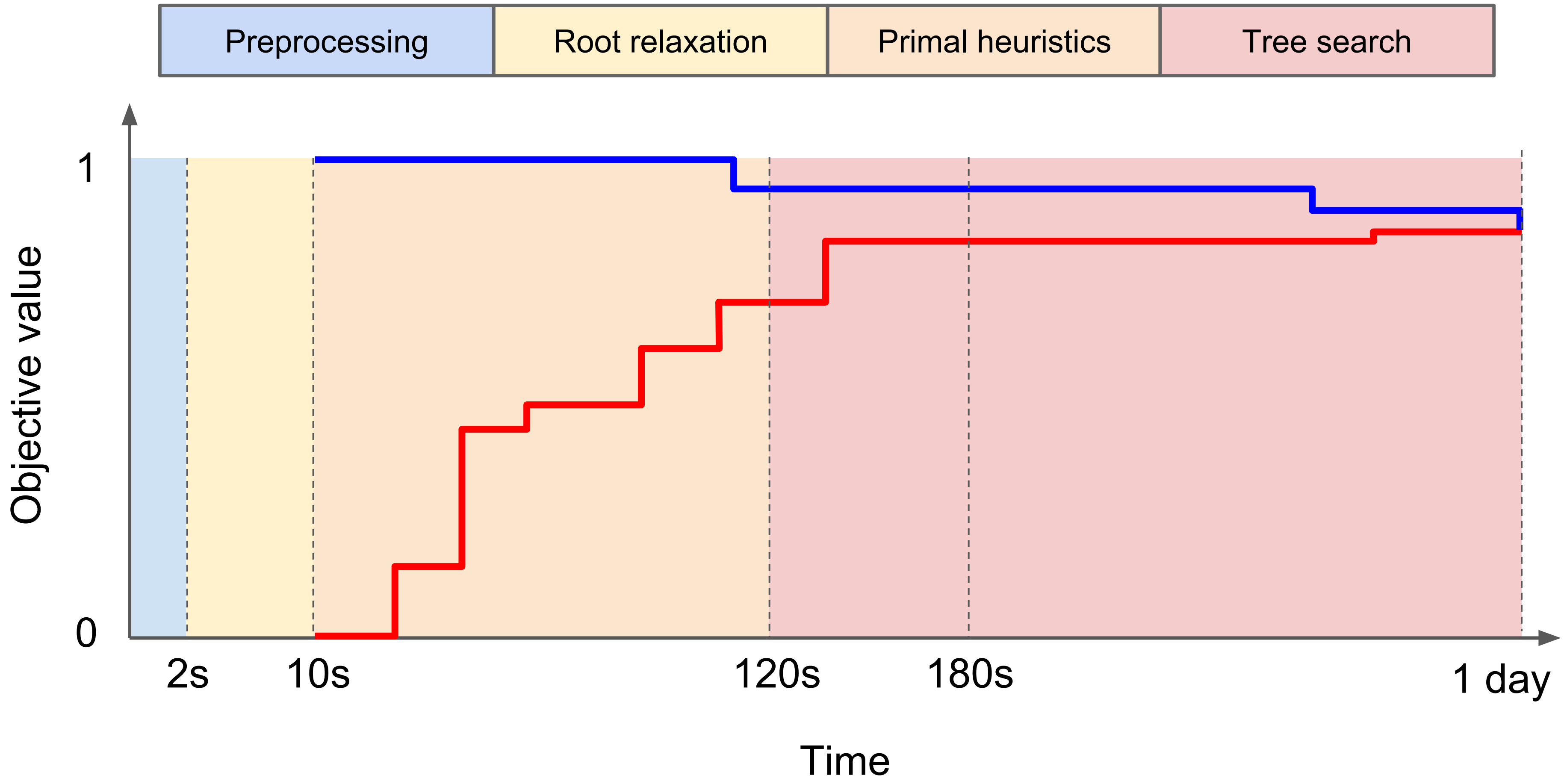
Outline

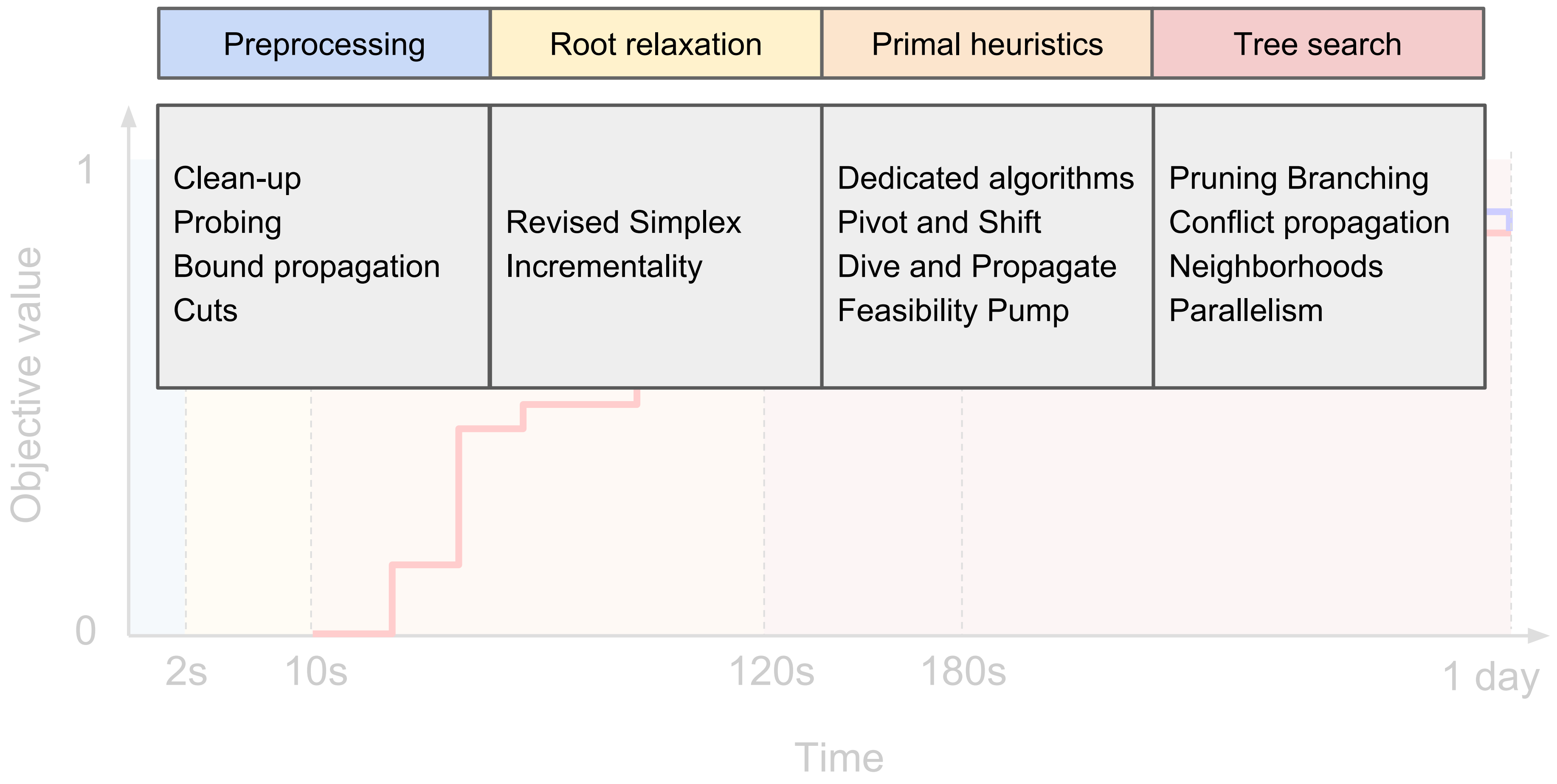
Why do we use MIP?
- Engineering
- Efficiency

Why are MIP solvers efficient?!?
- **Solver**
- **Model**

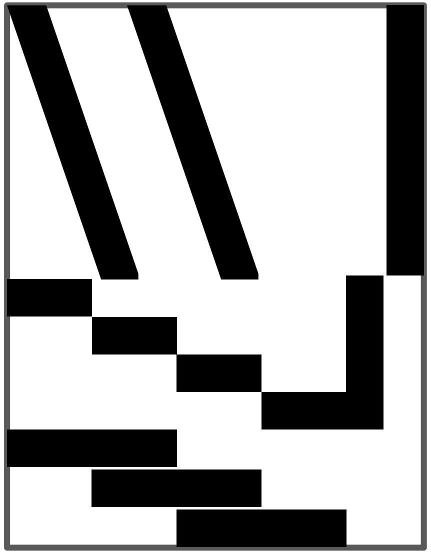
Self-doubt


1
0
Objective value
2s
10s
120s
180s
1 day
Time


Preprocessing
Root relaxation
Primal heuristics
Tree search
Objective value
1
0
2s
10s
120s
180s
1 day
Time

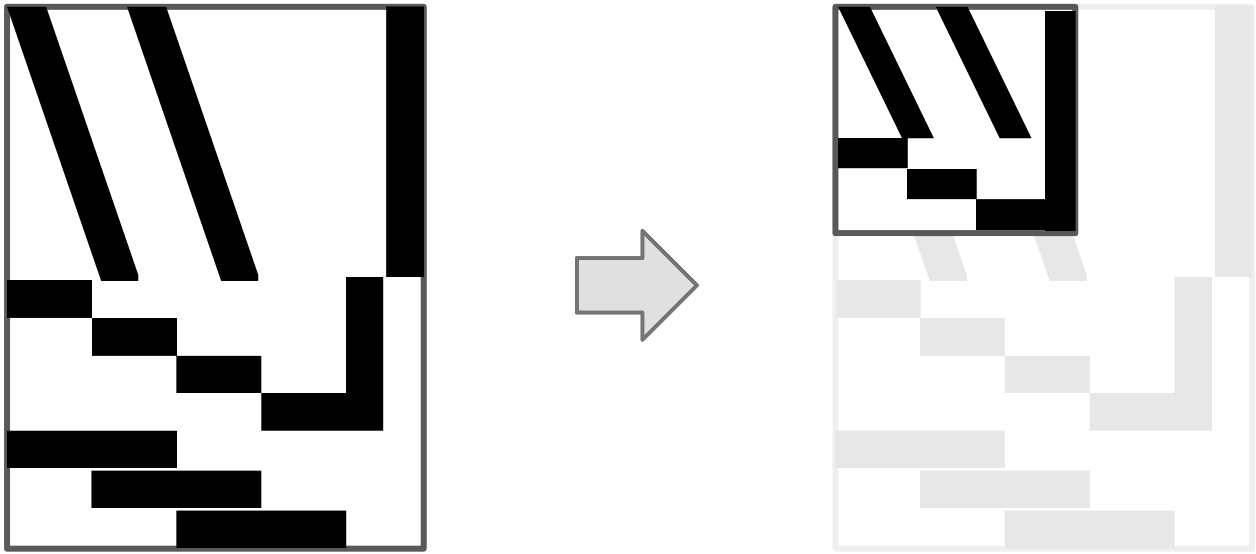

Preprocessing
Root relaxation
Primal heuristics
Tree search
Clean-up
Probing
Bound propagation
Cuts
Revised Simplex
Incrementality
Dedicated algorithms
Pivot and Shift
Dive and Propagate
Feasibility Pump
Pruning Branching
Conflict propagation
Neighborhoods
Parallelism
Objective value
1
0
2s
10s
120s
180s
1 day
Time

Preprocessing: Clean-up



Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

Preprocessing: Clean-up



Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

Preprocessing: Bound propagation

```
LB(c) ≤ ∑v = 1..V Coeff(c, v) * var(v) ≤ UB(c)

for constraint c = 1..C:
  for variable k = 1..V:
    if Coeff(c, k) > 0:
      UB’(k) = min(
          UB(k),
          (UB(c) - ∑v = 1..k, k+2..V Coeff(c, v) * LB(v)) / Coeff(c, k))

      LB’(k) = max(
          LB(k),
          (LB(c) - ∑v = 1..k, k+2..V Coeff(c, v) * UB(v)) / Coeff(c, k))
```

Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

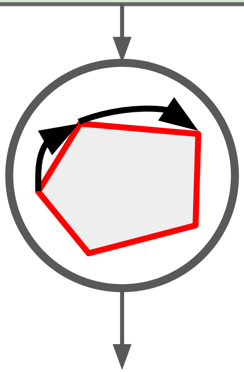
Preprocessing: Bound propagation

```
LB(c) ≤ ∑v = 1..V Coeff(c, v) * var(v) ≤ UB(c)

for constraint c = 1..C:
  for variable k = 1..V | var(k) ∈ Z:
    if Coeff(c, k) > 0:
      UB'(k) = min(
          UB(k),
          floor(UB(c) - ∑v = 1..k, k+2..V Coeff(c, v) * LB(v)) / Coeff(c, k))

      LB'(k) = max(
          LB(k),
          ceil(LB(c) - ∑v = 1..k, k+2..V Coeff(c, v) * UB(v)) / Coeff(c, k))
```

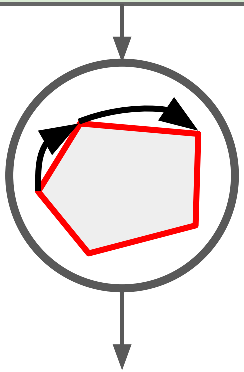
Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

Preprocessing: Probing

$\min c^T x$	$\min c^T x$
$Ax \leq b$	$Ax \leq b$
$x_k = 0$	$x_k = 1$
$x_j \in \{0, 1\}$	$x_j \in \{0, 1\}$



LP-feasible

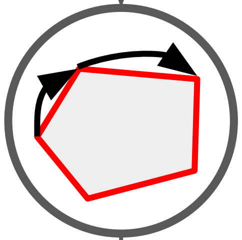


Infeasible

Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

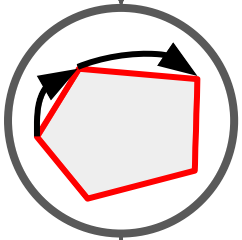
Preprocessing: Probing

$\min c^T x$
$Ax \leq b$
$x_k = 0$
$x_j \in \{0, 1\}$



LP-feasible

$\min c^T x$
$Ax \leq b$
$x_k = 1$
$x_j \in \{0, 1\}$



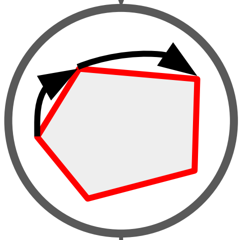
Infeasible

$\min c^T x$
$Ax \leq b$
$x_k = 0$
$x_j \in \{0, 1\}$

Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

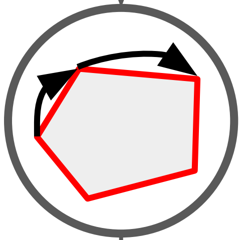
Preprocessing: Probing

$$\begin{aligned} &\min\ c^T x \\ &Ax \leq b \\ &x_k = 0 \\ &x_j \in \{0, 1\} \end{aligned}$$



LP-feasible

$$\begin{aligned} &\min\ c^T x \\ &Ax \leq b \\ &x_k = 1 \\ &x_j \in \{0, 1\} \end{aligned}$$



Infeasible

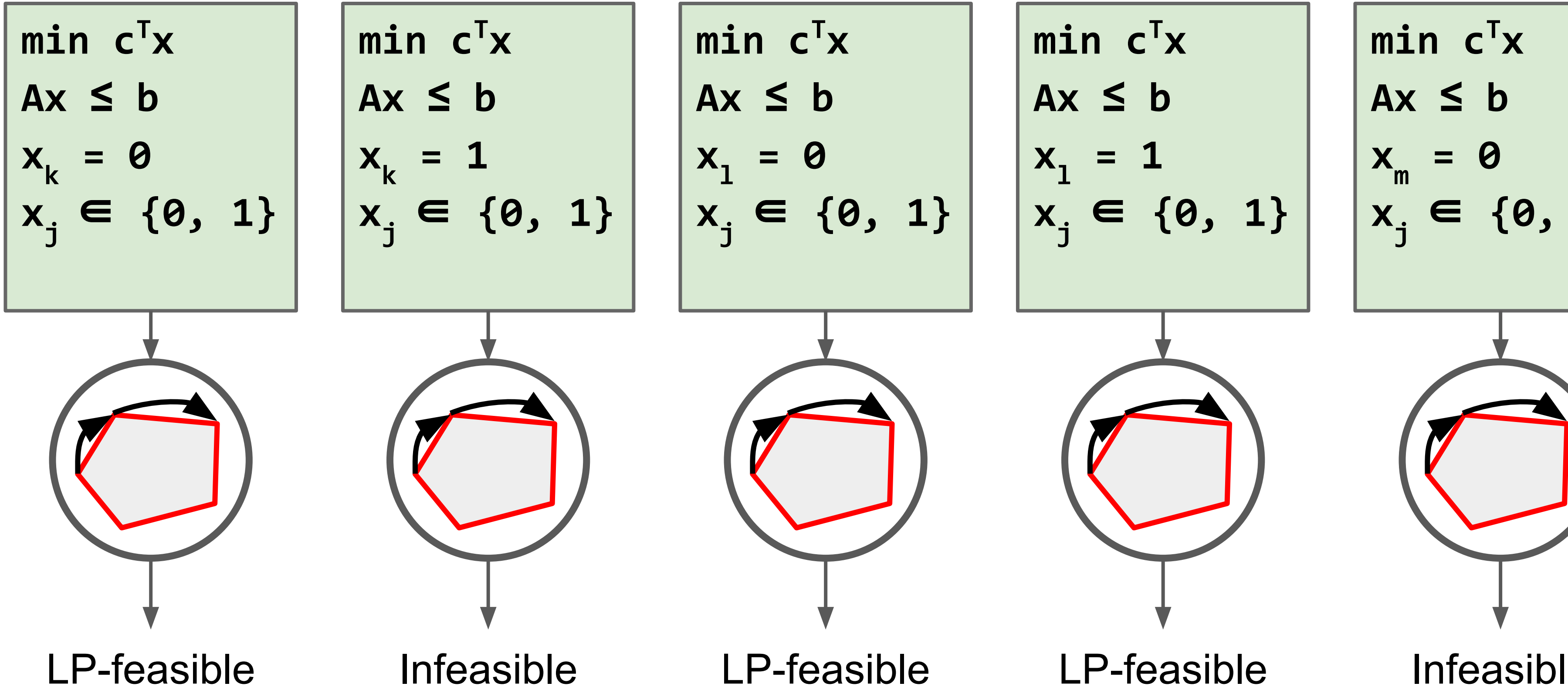
$$\begin{aligned} &\min\ c^T x \\ &Ax \leq b \\ &x_k = 0 \\ &x_j \in \{0, 1\} \end{aligned}$$

$$\begin{aligned} &\min\ c'^T x' \\ &A'x' \leq b \\ &x_j \in \{0, 1\} \end{aligned}$$

Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44

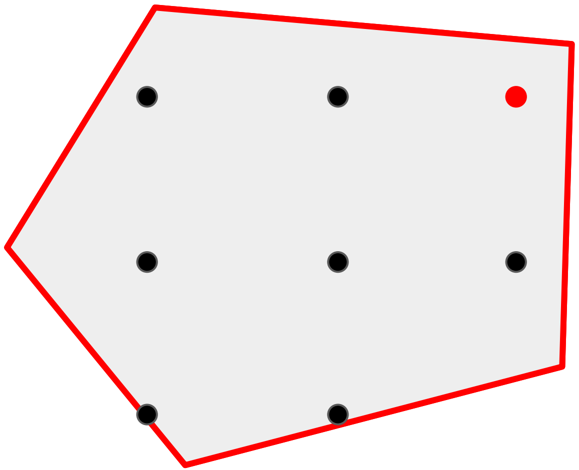
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

Preprocessing: Probing



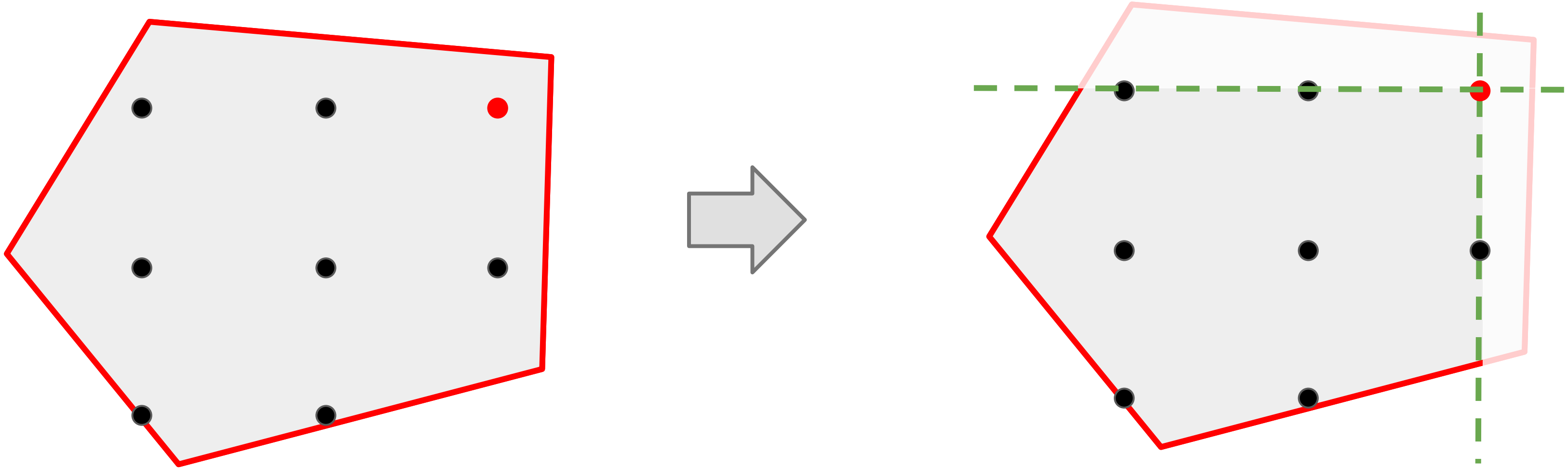
Achterberg, Bixby, Gu, Rothberg, Weninger (2016) "Presolve Reductions in Mixed Integer Programming" ZIB Report 16-44
Savelsbergh (1994) "Preprocessing and probing techniques for MIP problems" ORSA Journal on Computing 6.4, 445-454

Preprocessing: Cuts



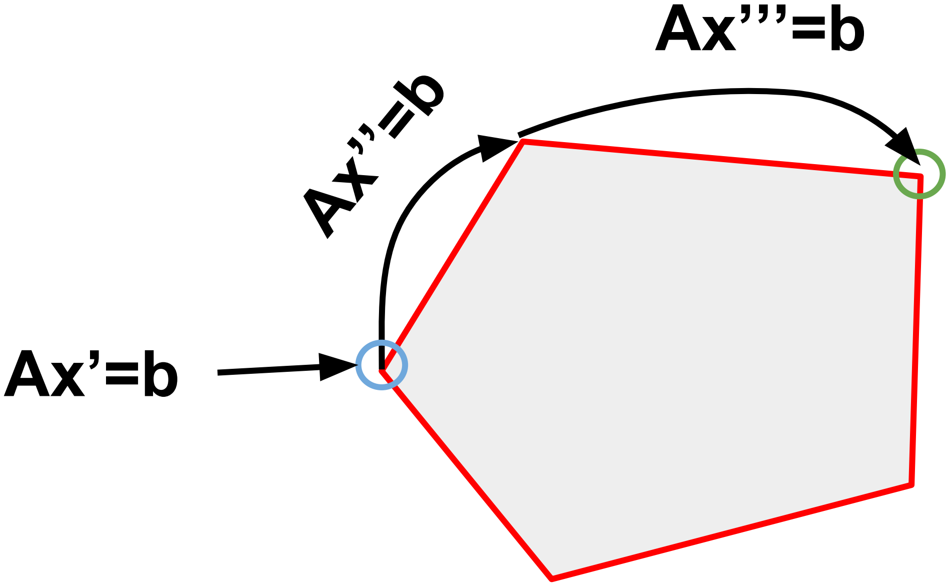
Russell (2006) "Cutting Planes for Mixed Integer Programming" (unpublished)

Preprocessing: Cuts

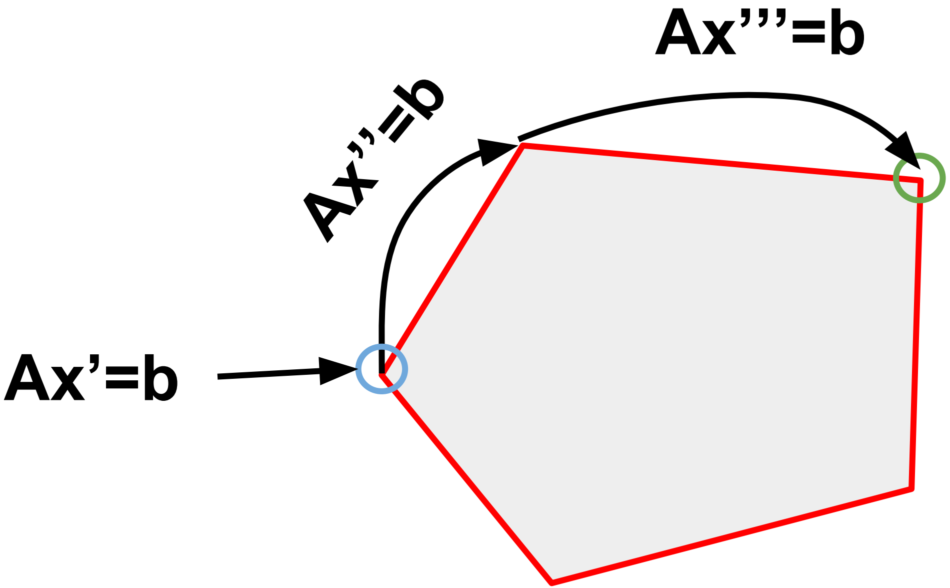


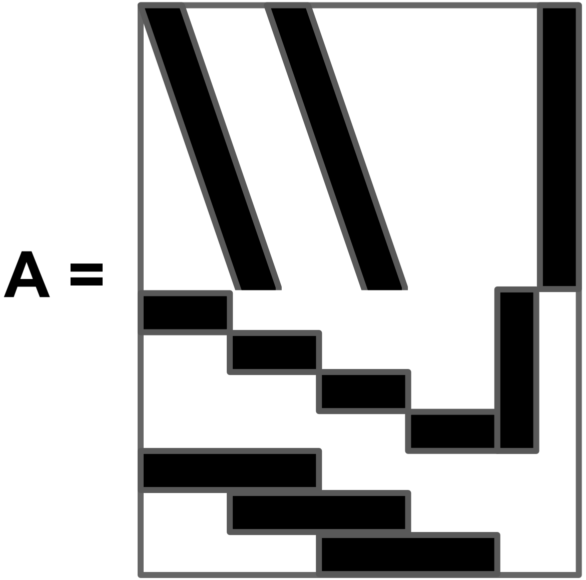
Russell (2006) "Cutting Planes for Mixed Integer Programming" (unpublished)

Relaxation: Revised Simplex

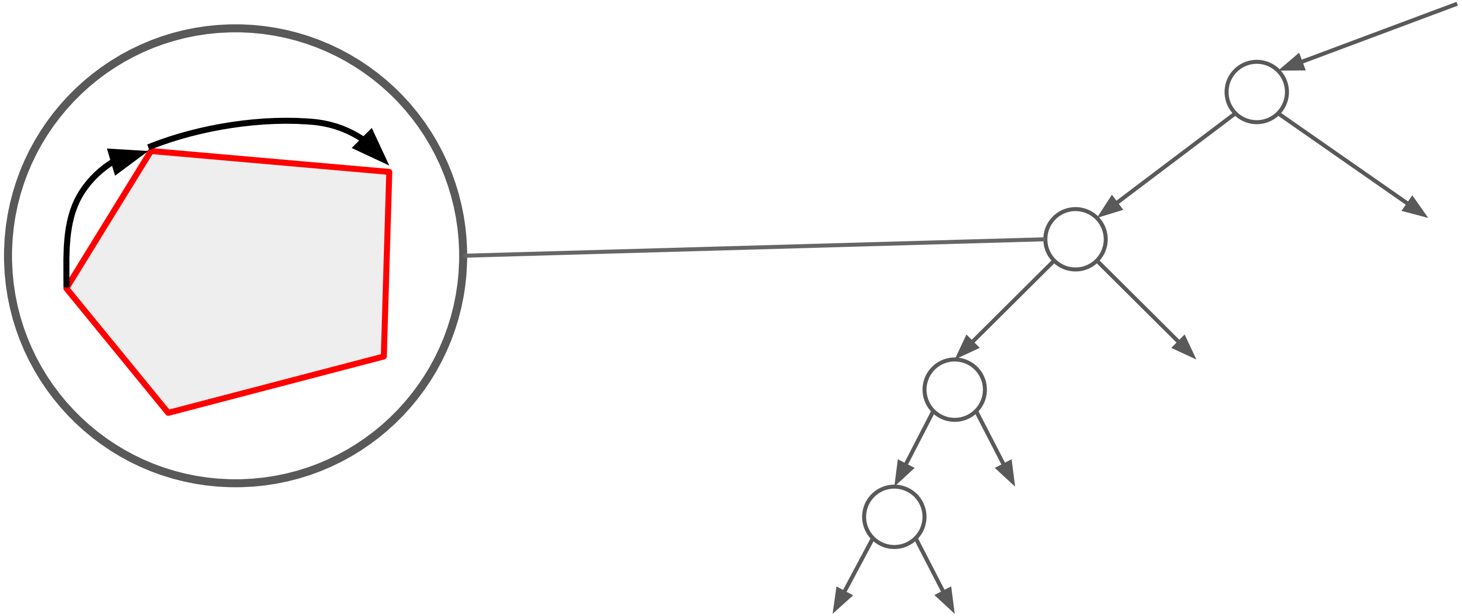


Relaxation: Revised Simplex

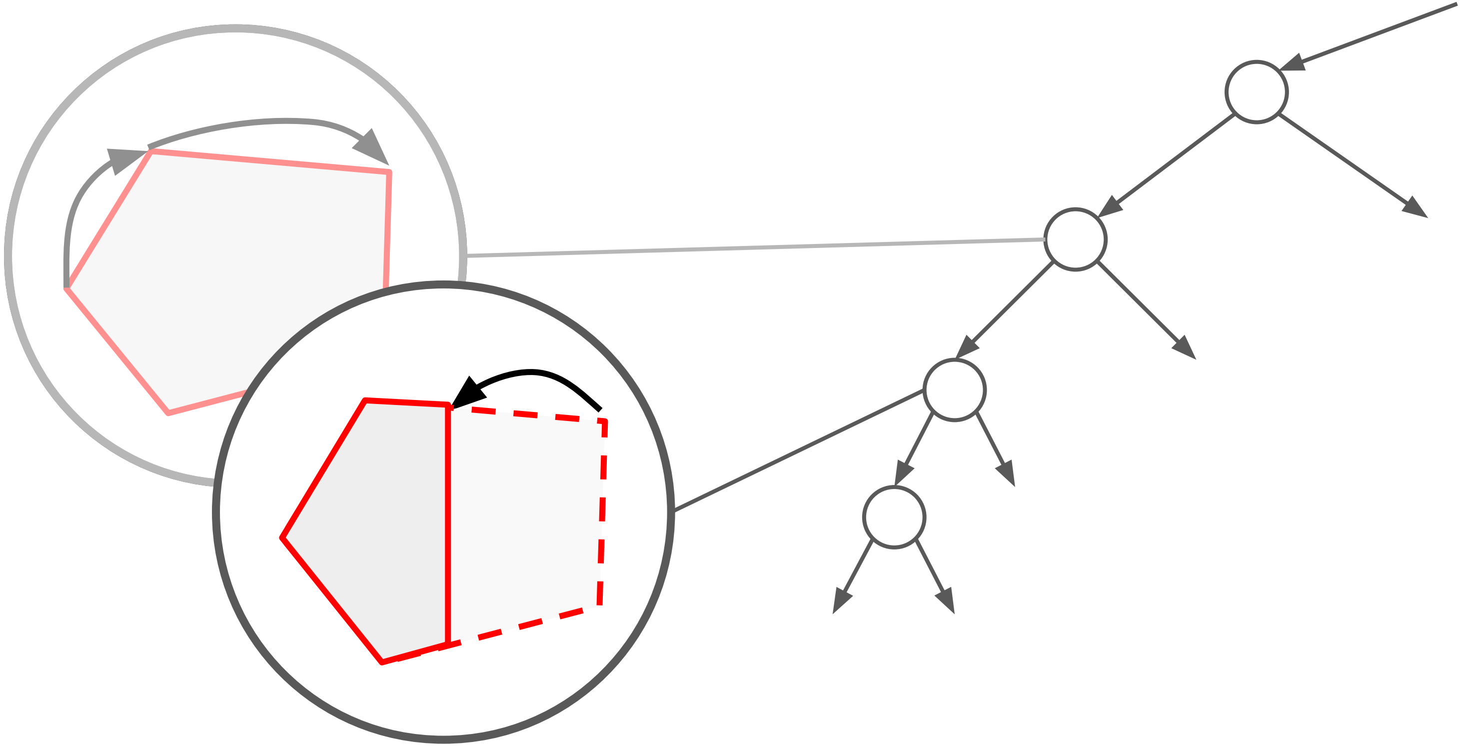




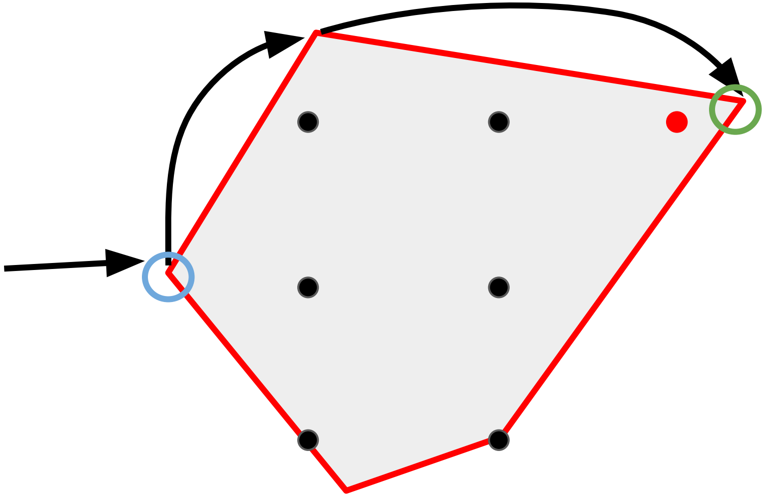
Relaxation: Incrementality



Relaxation: Incrementality

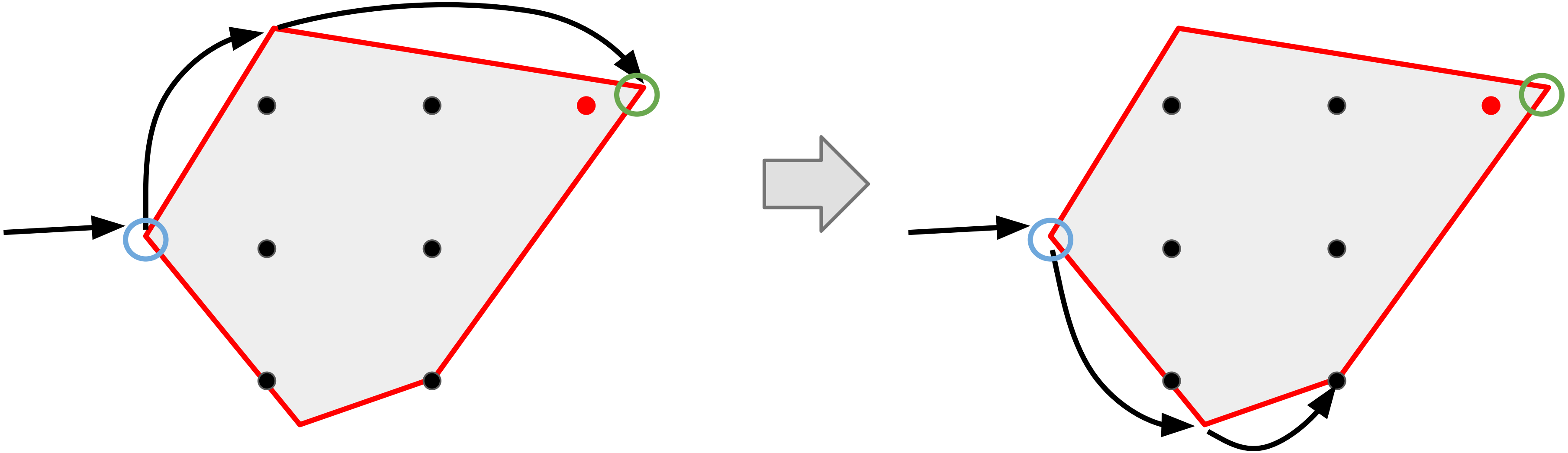


Primal heuristics: Pivot and Shift



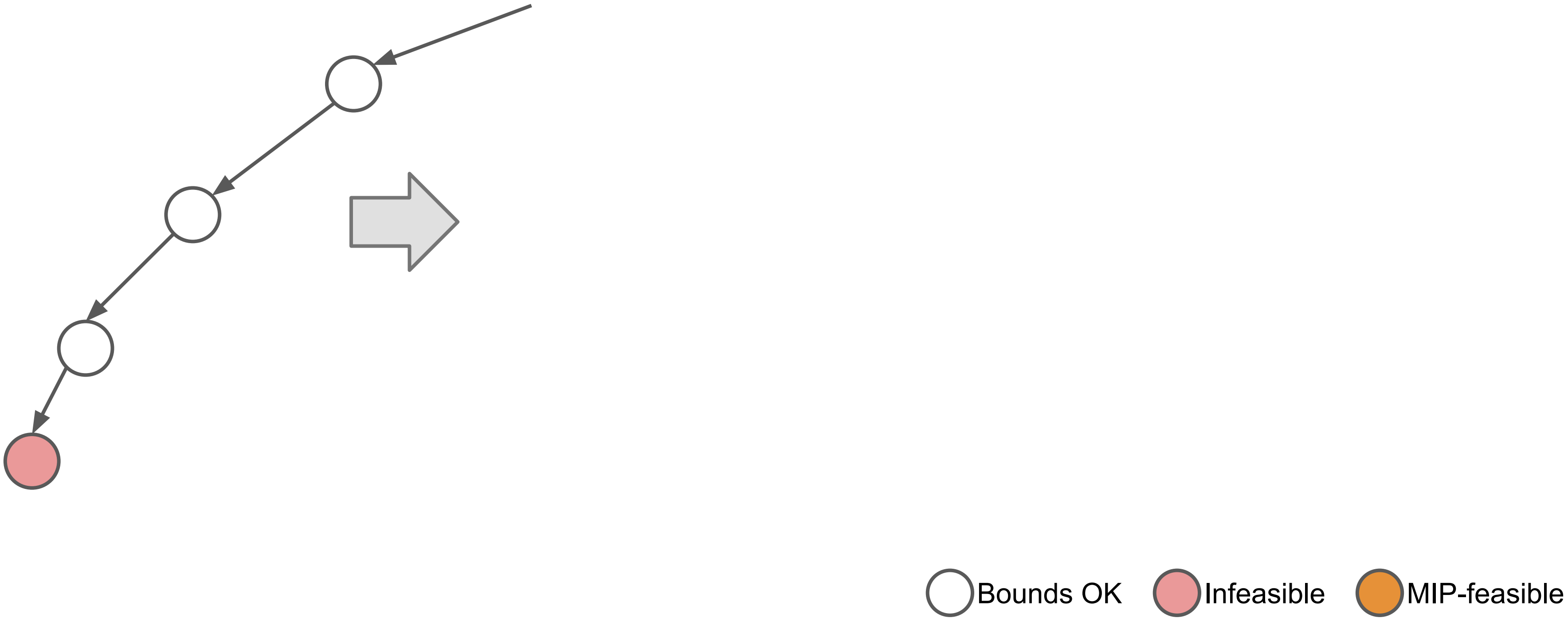
Balas, Schmieta, Wallace (2004) "Pivot and shift - a mixed integer programming heuristic." Discrete Optimization 1.1, 3-12

Primal heuristics: Pivot and Shift



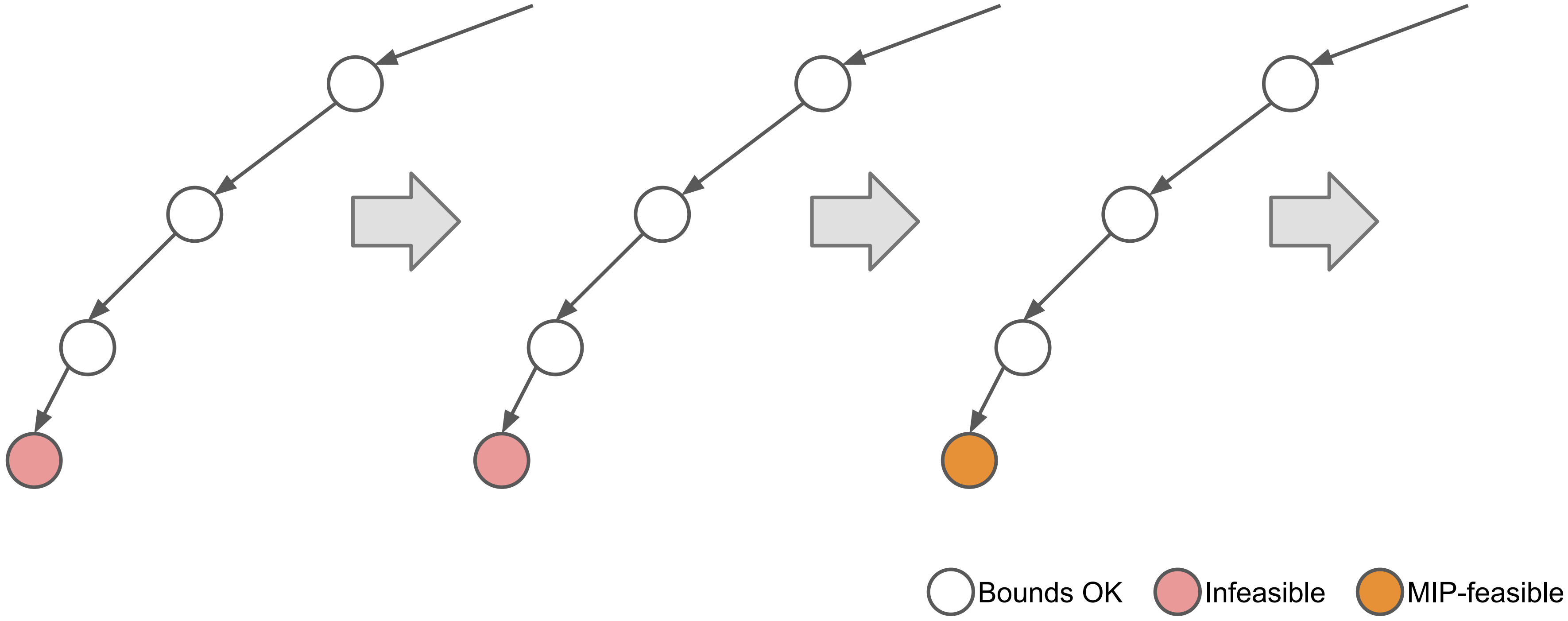
Balas, Schmieta, Wallace (2004) "Pivot and shift - a mixed integer programming heuristic." Discrete Optimization 1.1, 3-12

Primal heuristics: Shift and Propagate



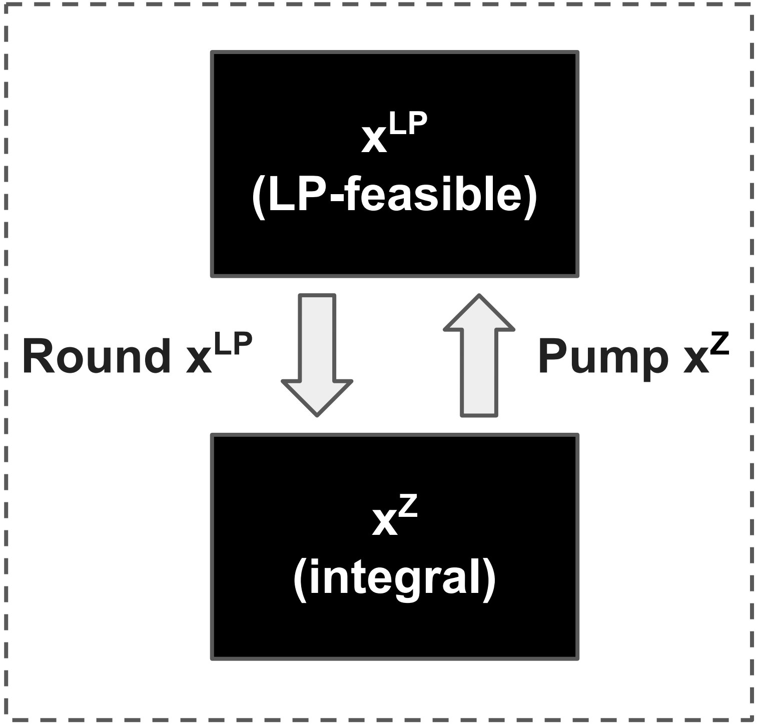
Berthold, Hendel (2015) "Shift and propagate" Journal of Heuristics, 21:73-106

Primal heuristics: Shift and Propagate



Berthold, Hendel (2015) "Shift and propagate" Journal of Heuristics, 21:73-106

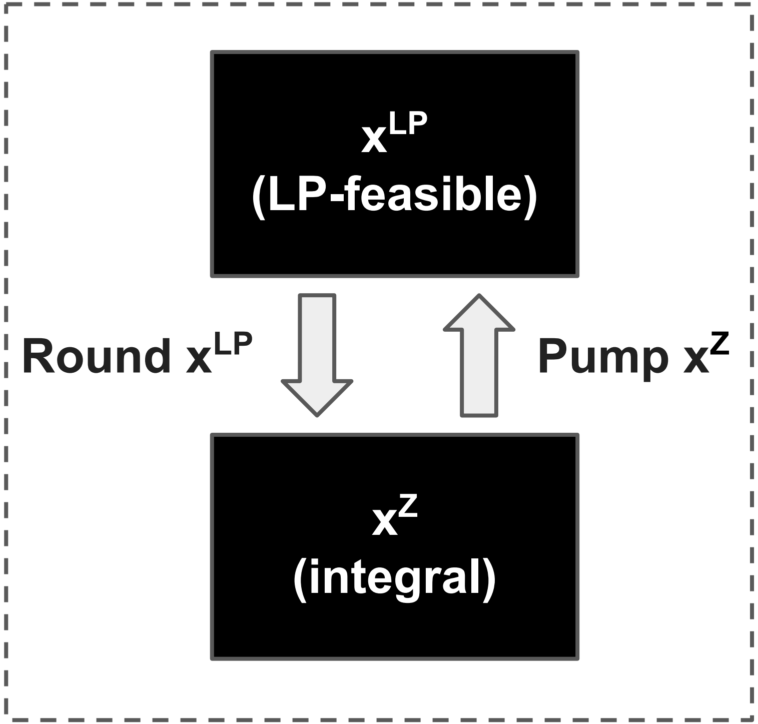
Primal heuristics: Feasibility Pump



Repeat until x^{LP+Z}

Fischetti, Glover, Lodi (2005) “The feasibility pump”, Math. Program., Ser. A 104, 91–104

Primal heuristics: Feasibility Pump



Repeat until x^{LP+Z}

Round:

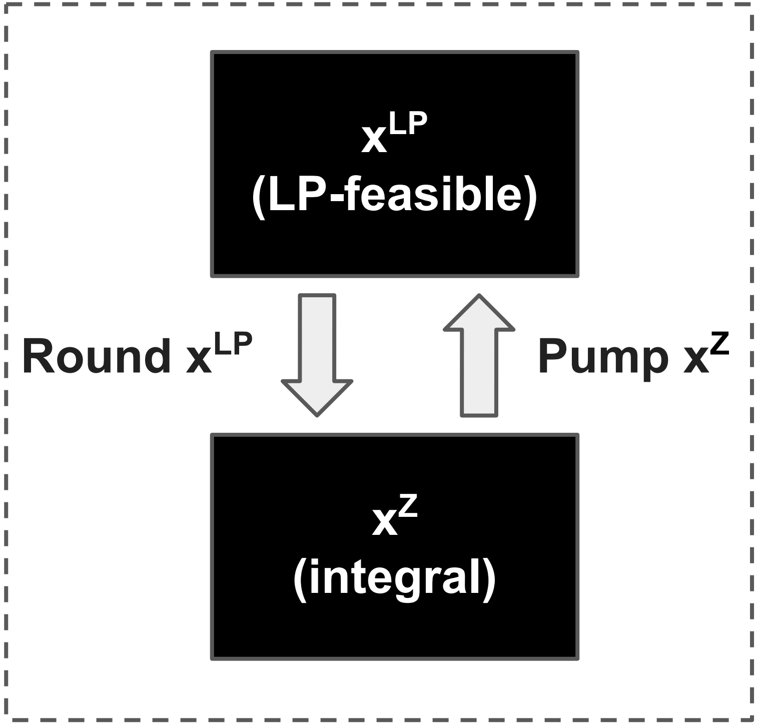
$$x^Z = \mathrm{round}(x^{LP} + \mathrm{Rand}())$$

Pump:

$$\min |x - x^Z|_1$$
$$Ax \leq b$$
$$x_j \in Z$$

Fischetti, Glover, Lodi (2005) “The feasibility pump”, Math. Program., Ser. A 104, 91-104

Primal heuristics: Objective Feasibility Pump



Repeat until x^{LP+Z}

Round:

```
x^Z = round(x^LP + Rand())
```

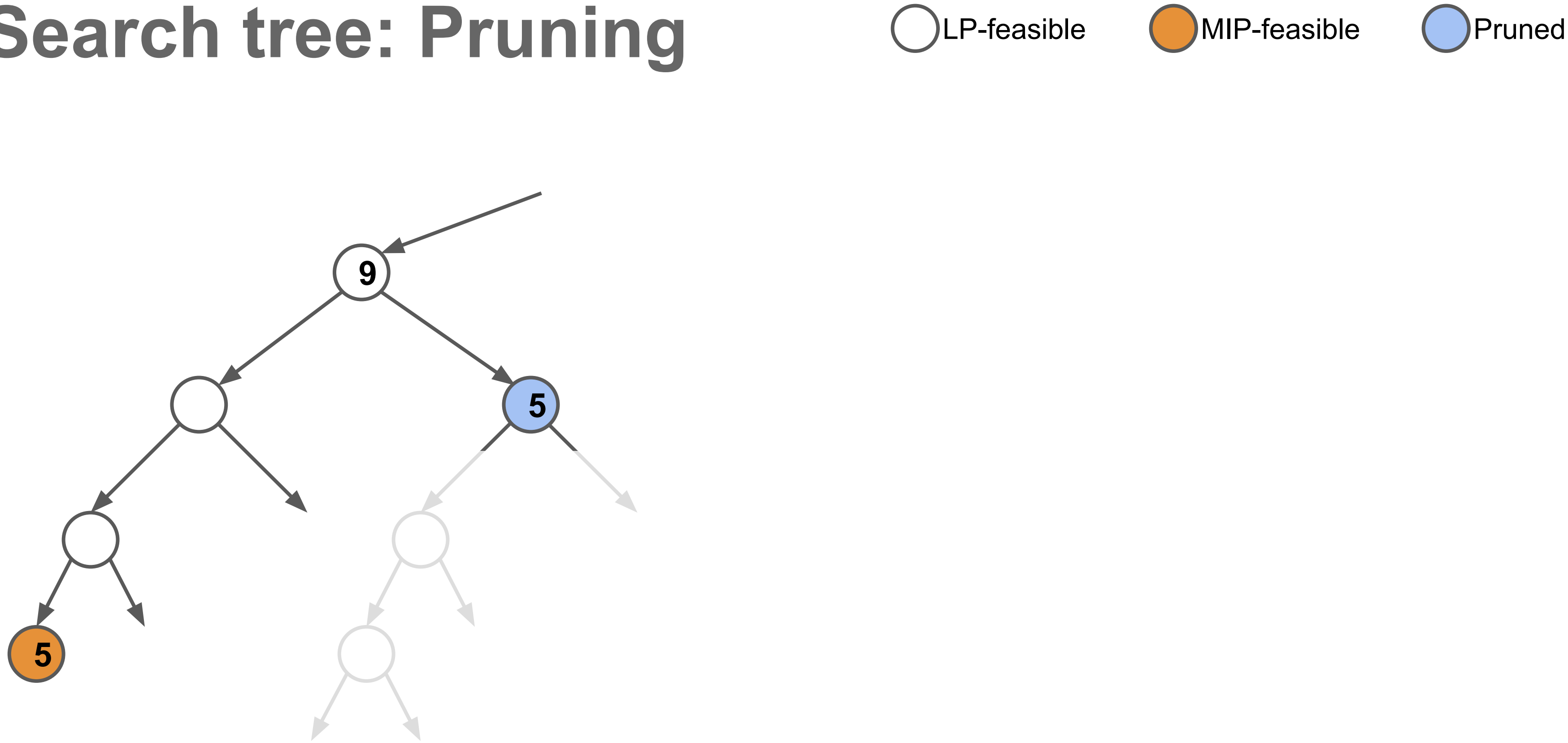
Pump:

```
min w * c^T x + (1 - w) * |x - x^Z|_1
Ax ≤ b
x_j ∈ Z
```

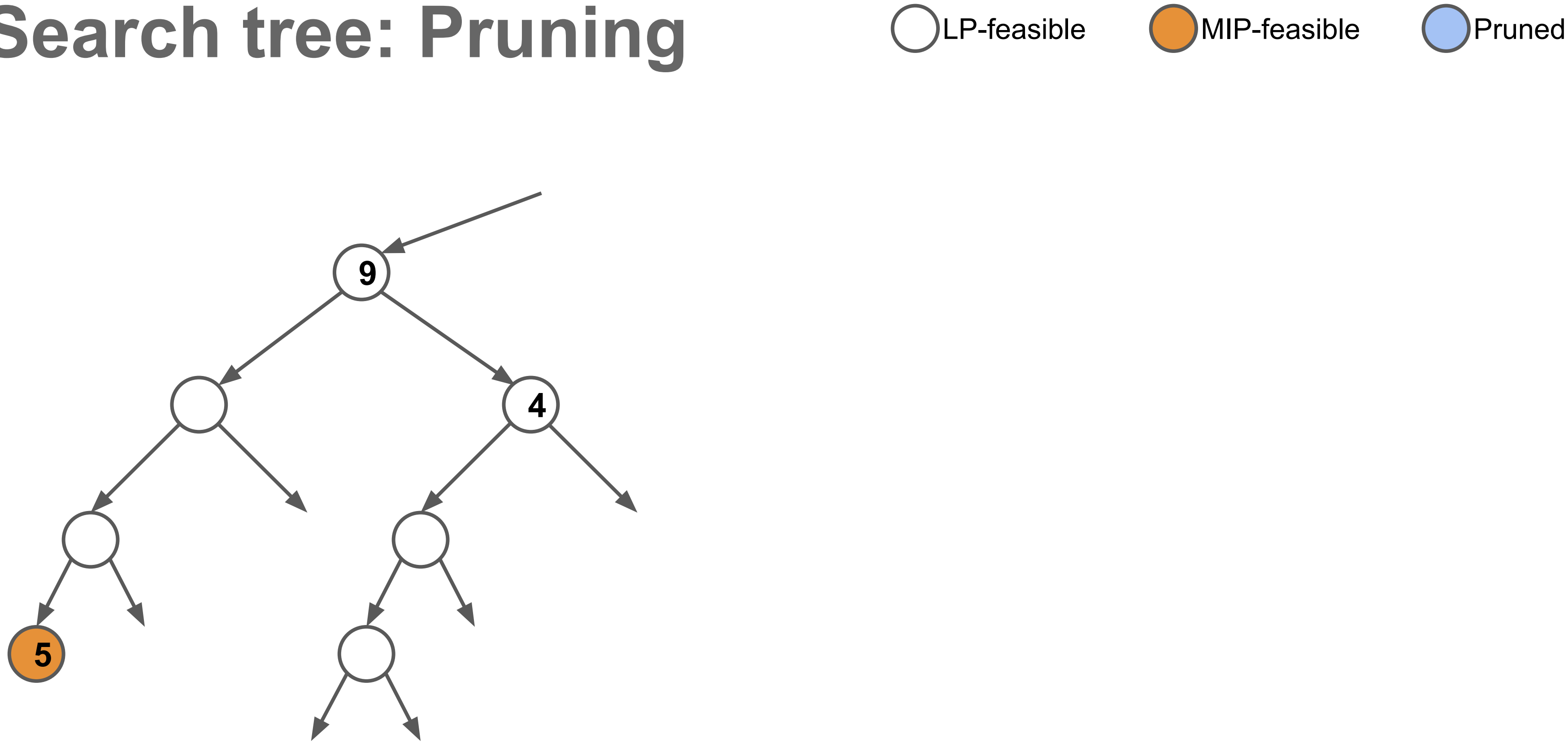

Achterberg, Berthold (2009) “Improving the feasibility pump”, Discrete Optimization 4, 77-86

Fischetti, Glover, Lodi (2005) “The feasibility pump”, Math. Program., Ser. A 104, 91-104

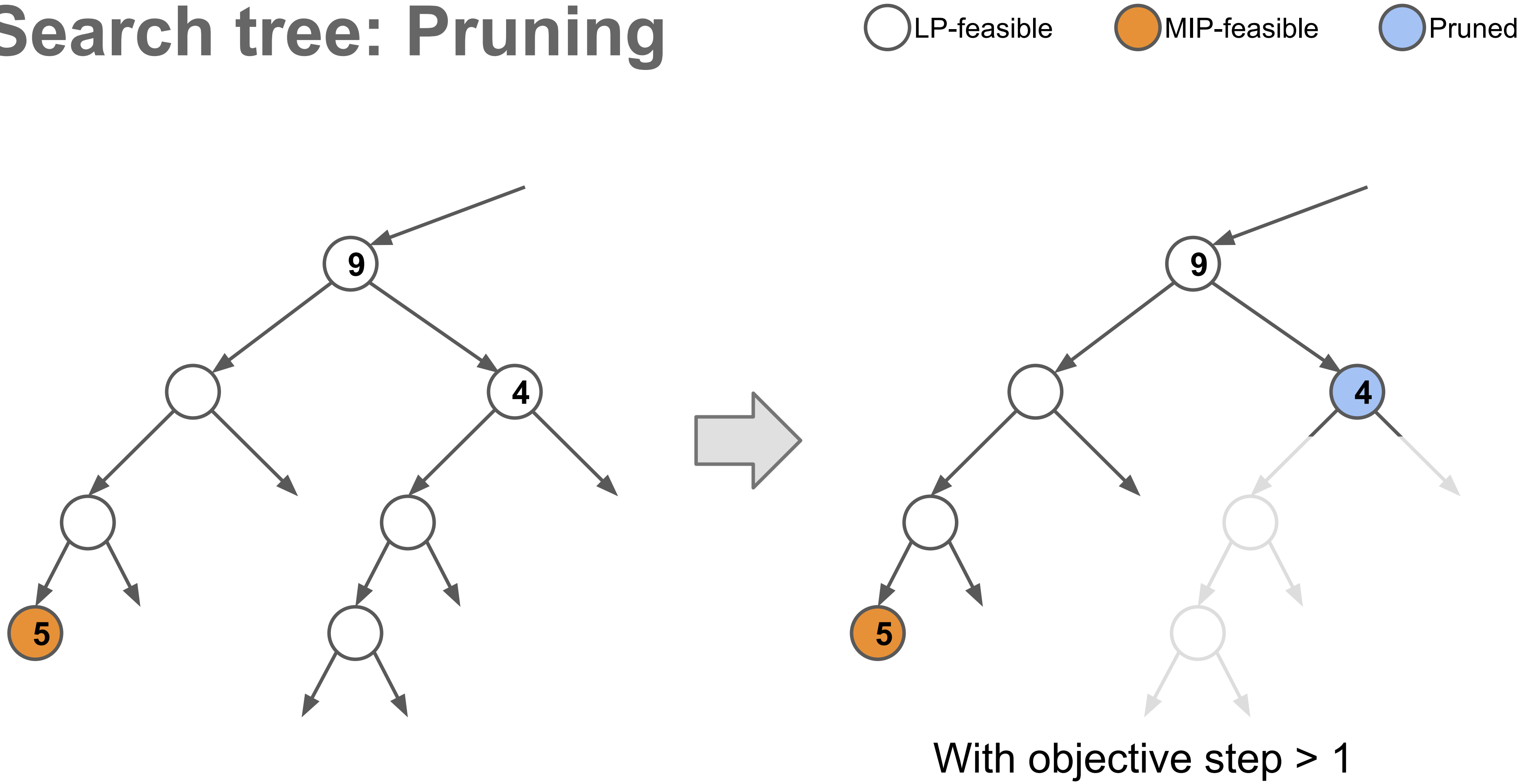
Search tree: Pruning



Search tree: Pruning



Search tree: Pruning

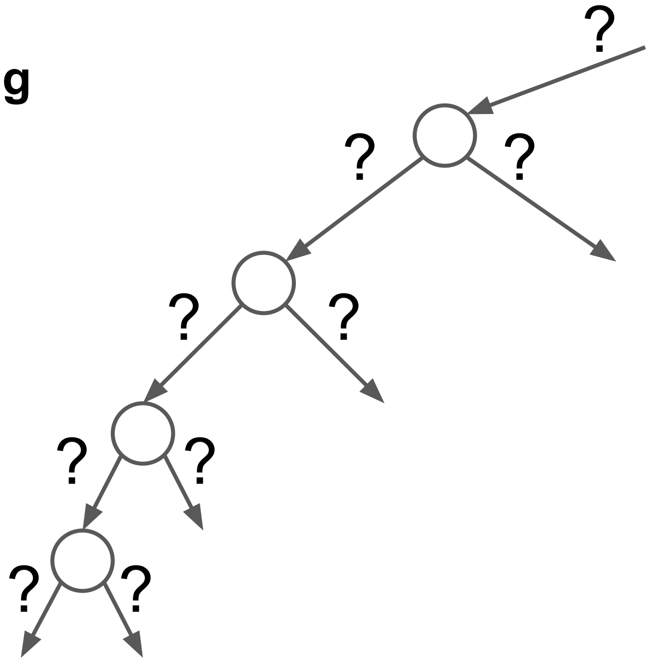


Search tree: Branching

Pseudo-cost branching

Track objective improvement incurred by branching

Pick the variable with highest predicted impact



Achterberg, Koch, Martin (2005) "Branching rules revisited" Operations Research Letters 33.1, 42-54.

Search tree: Branching

Pseudo-cost branching

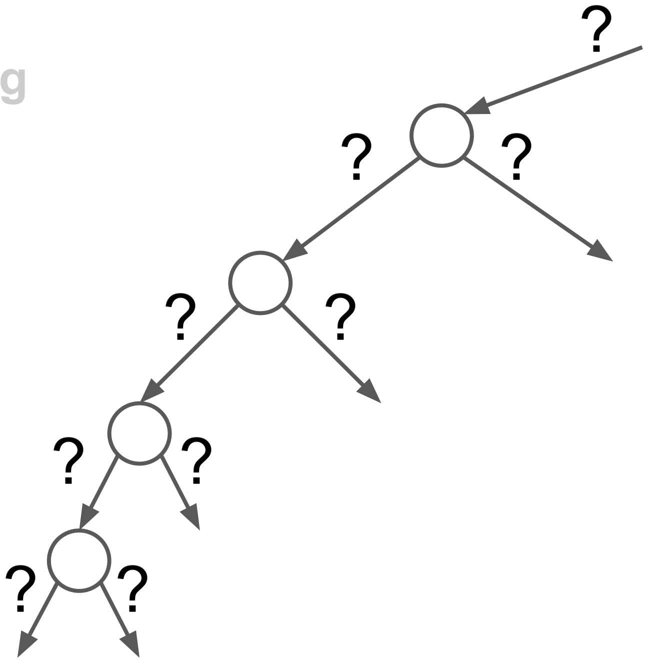
Track objective improvement incurred by branching

Pick the variable with highest predicted impact

Strong branching

Try to branch on all variables

Pick the variable with highest actual impact



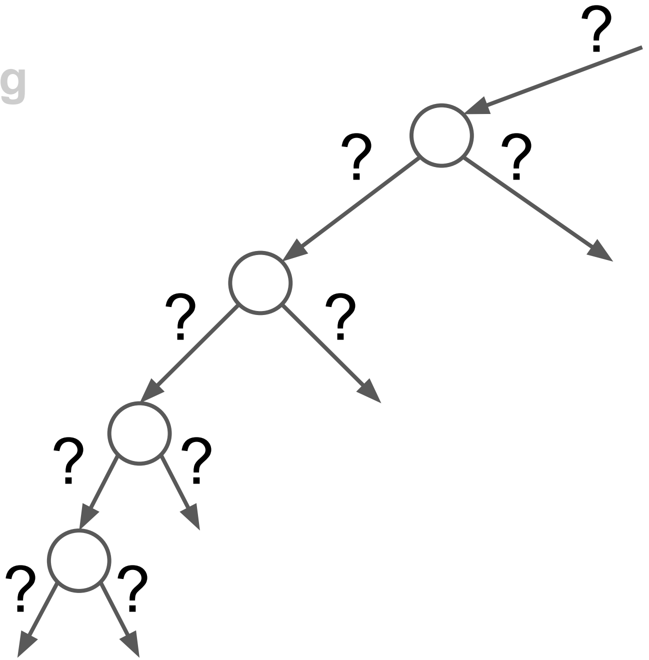
Achterberg, Koch, Martin (2005) "Branching rules revisited" Operations Research Letters 33.1, 42-54.

Search tree: Branching

Pseudo-cost branching
Track objective improvement incurred by branching
Pick the variable with highest predicted impact

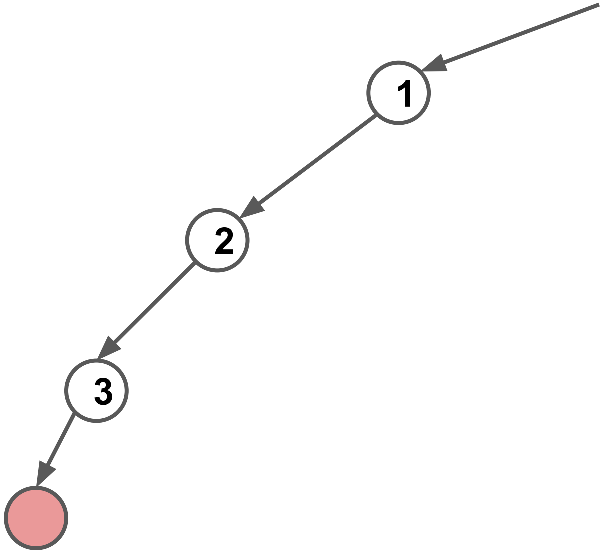
Strong branching
Try to branch on all variables
Pick the variable with highest actual impact

Active constraint branching
Compute constraints' slacks
Pick the variable from "active" constraints



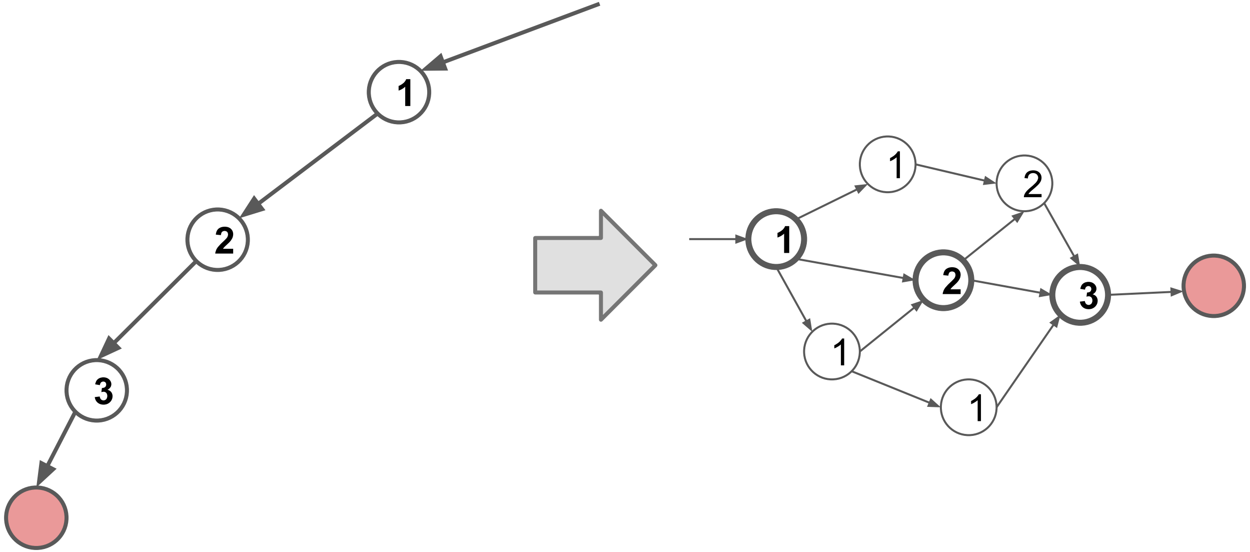
Pryor, Chinneck (2011) "Faster integer-feasibility in mixed-integer linear programs by branching to force change" Computers & Operations Research 38.8, 1143-1152.

Search tree: Conflict propagation



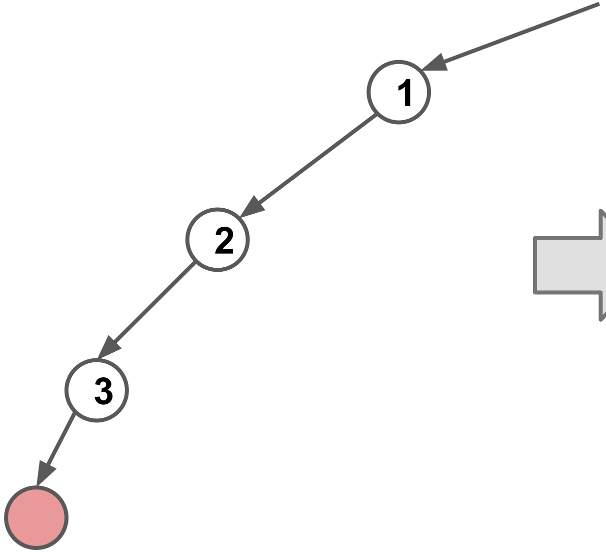
Sandholm, Tuomas, and Robert Shields (2006) "Nogood learning for mixed integer programming"
Achterberg (2007) “Conflict analysis in mixed integer programming”, Discrete Optimization 4, 4-20
Nieuwenhuis (2014) "The intsat method for integer linear programming"

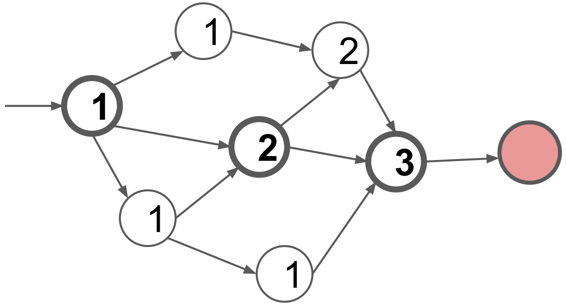
Search tree: Conflict propagation



Sandholm, Tuomas, and Robert Shields (2006) "Nogood learning for mixed integer programming"
Achterberg (2007) “Conflict analysis in mixed integer programming”, Discrete Optimization 4, 4-20
Nieuwenhuis (2014) "The intsat method for integer linear programming"

Search tree: Conflict propagation

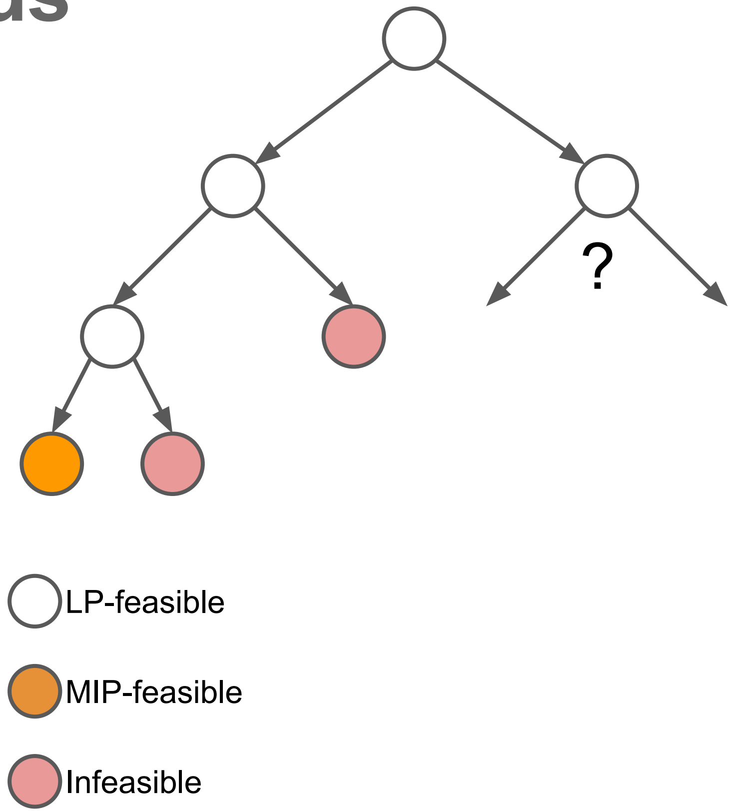




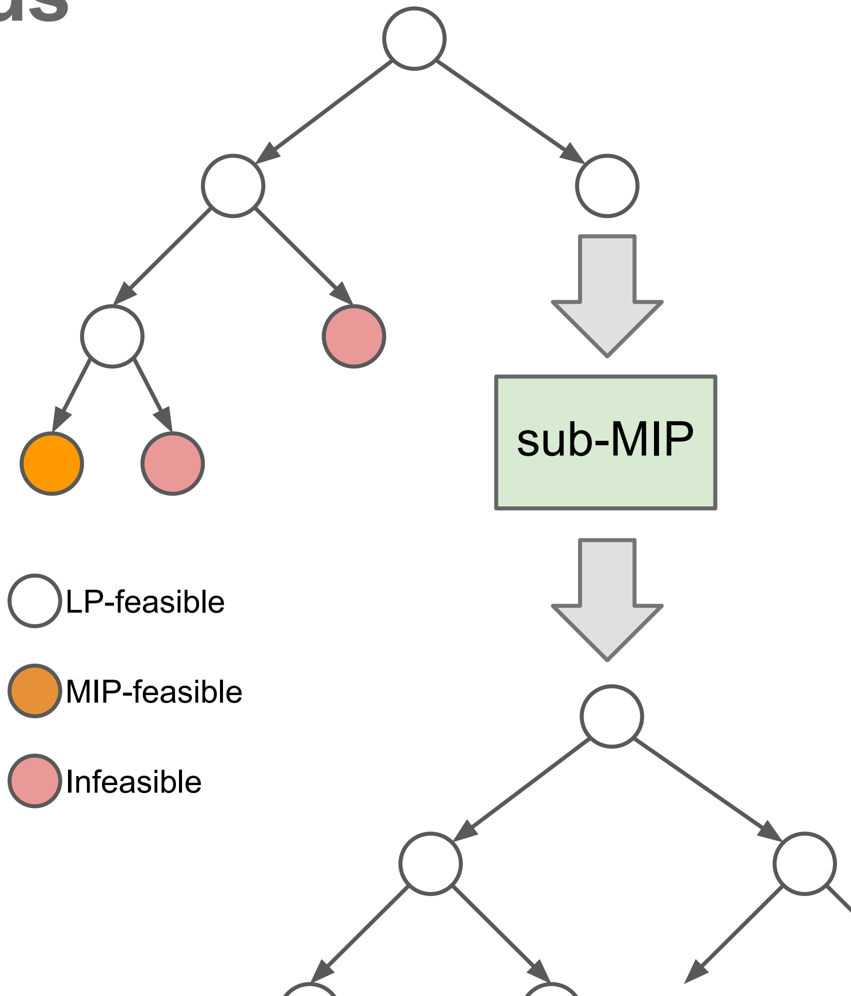
$$\min c^T x$$
$$Ax \leq b$$
$$x_j \in Z$$
$$\mathbf{x_1 + x_2 = 1}$$

Sandholm, Tuomas, and Robert Shields (2006) "Nogood learning for mixed integer programming"
Achterberg (2007) “Conflict analysis in mixed integer programming”, Discrete Optimization 4, 4-20
Nieuwenhuis (2014) "The intsat method for integer linear programming"

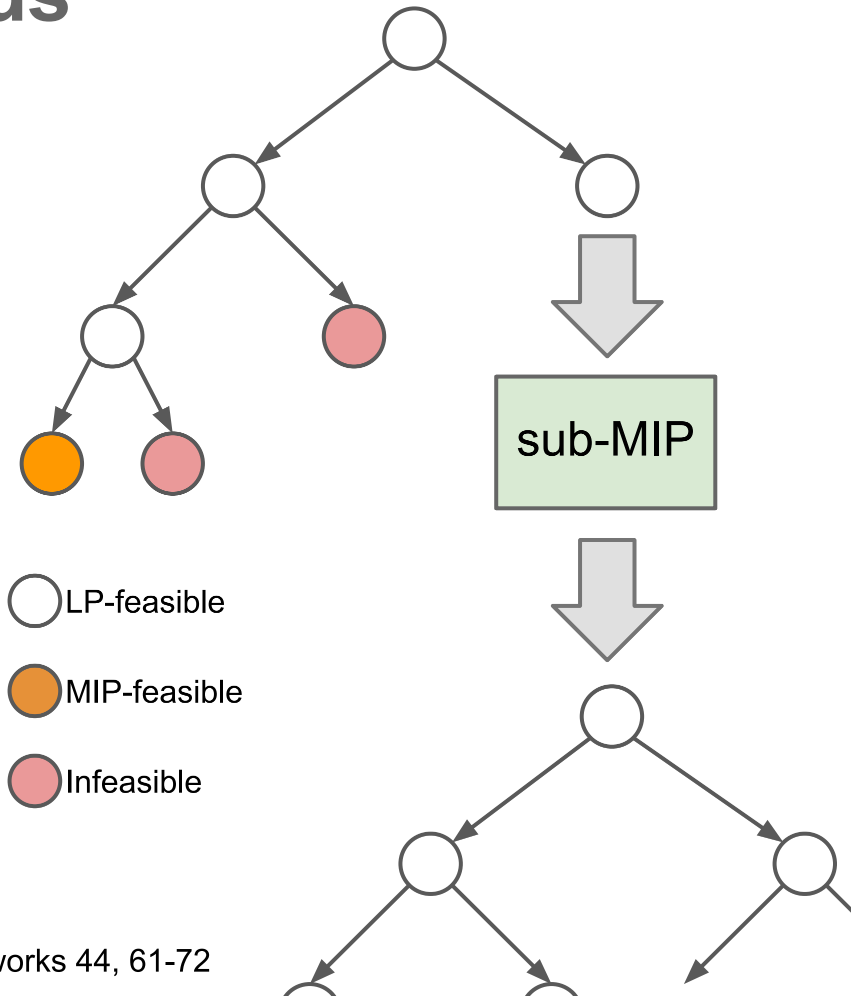
Search tree: Neighborhoods



Search tree: Neighborhoods



Search tree: Neighborhoods



Local branching

Upper bound Manhattan distance from incumbent

Solve the sub-MIP

Fischetti, Polo, Scantamburlo (2004) “A Local Branching Heuristic...”, Networks 44, 61-72

Search tree: Neighborhoods

Local branching

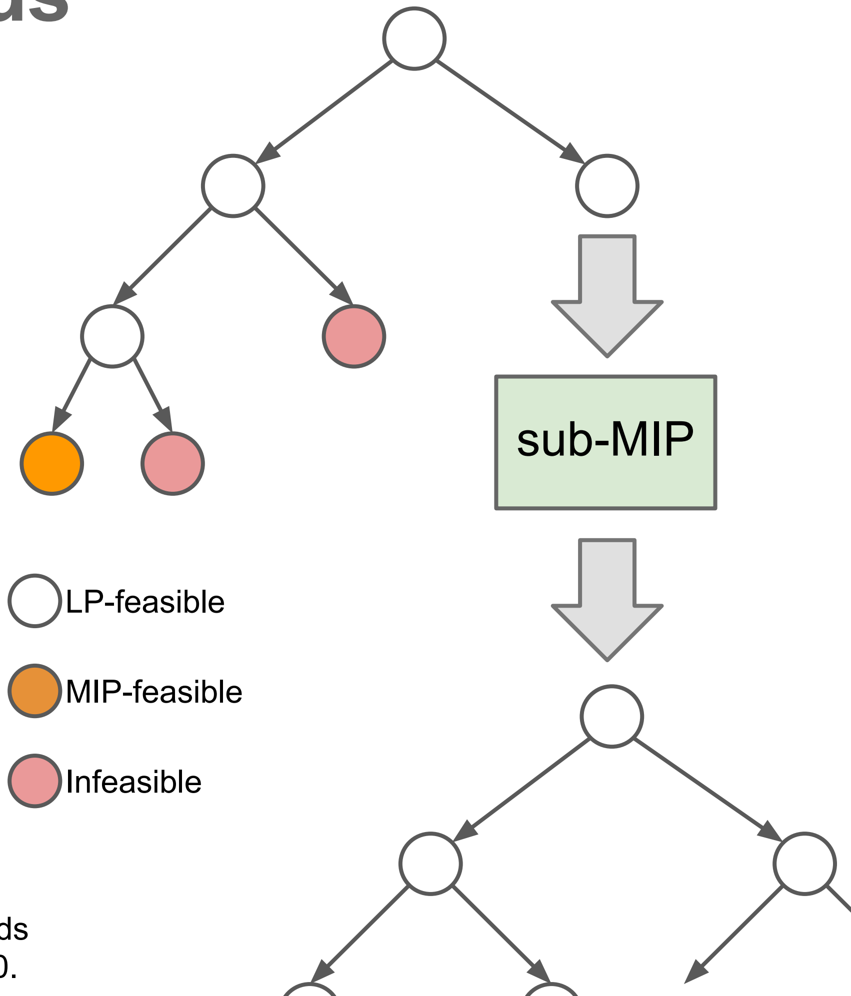
Upper bound Manhattan distance from incumbent

Solve the sub-MIP

Relaxation Induced Neighborhood Search

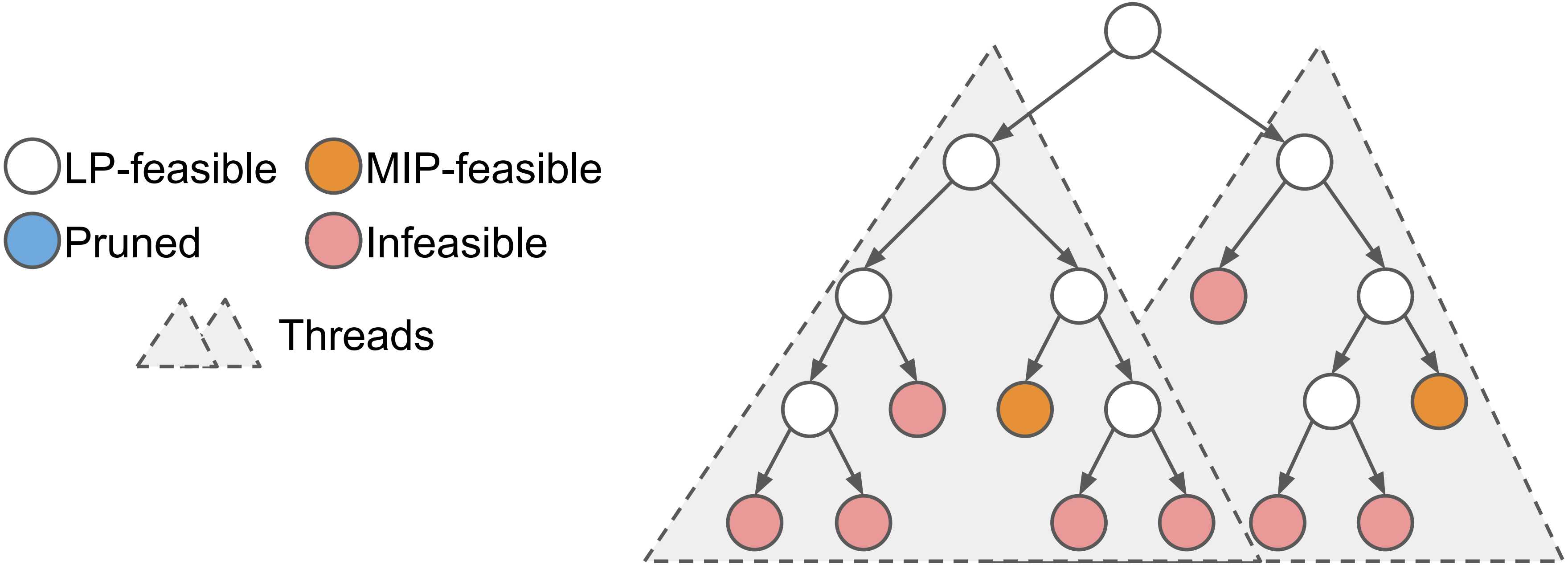
Fix integer variables with same value as incumbent(s)

Solve the sub-MIP

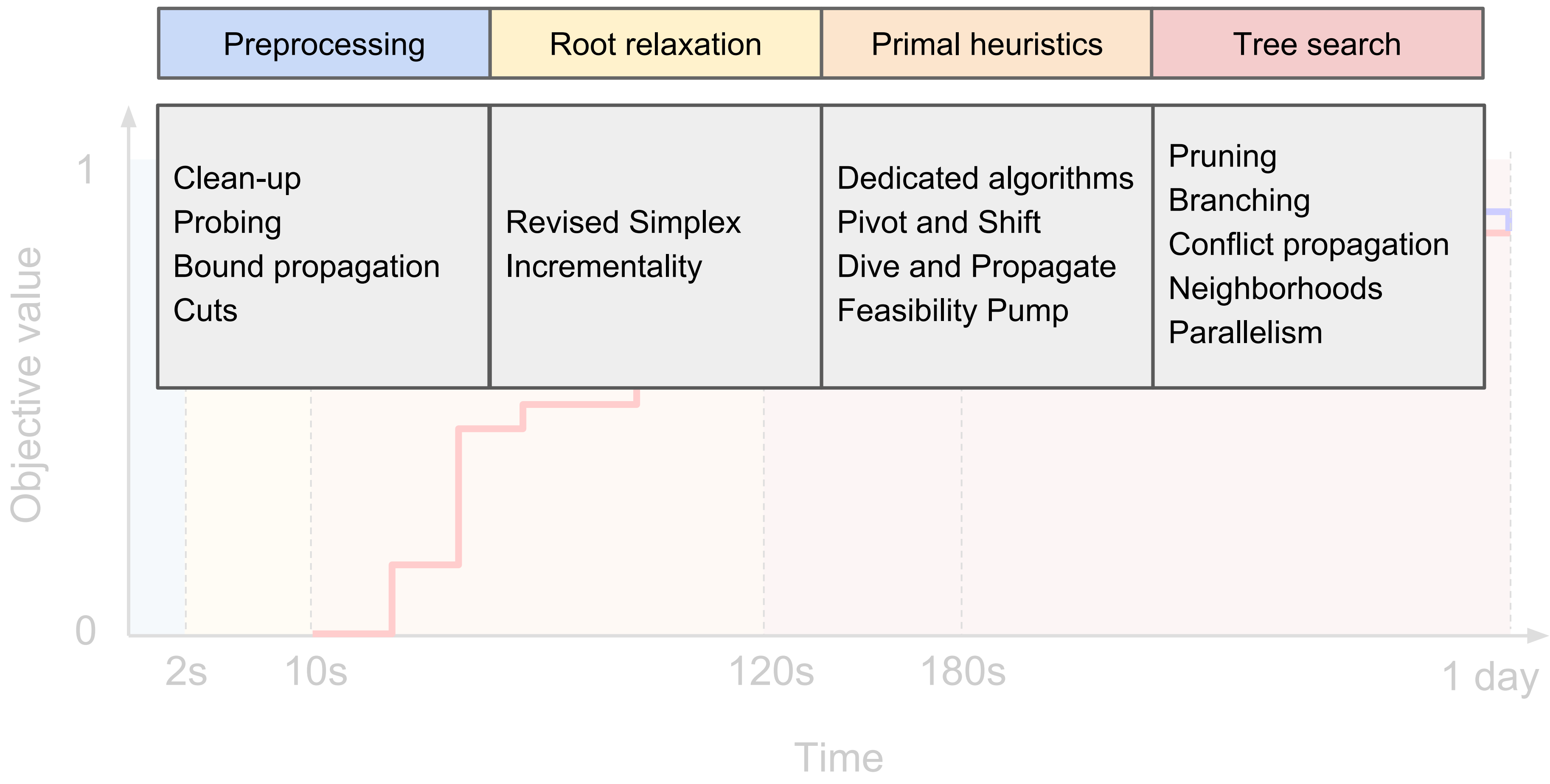


Danna, Rothberg, Pape (2005) "Exploring relaxation induced neighborhoods to improve MIP solutions", Mathematical Programming 102.1 (2005): 71-90.

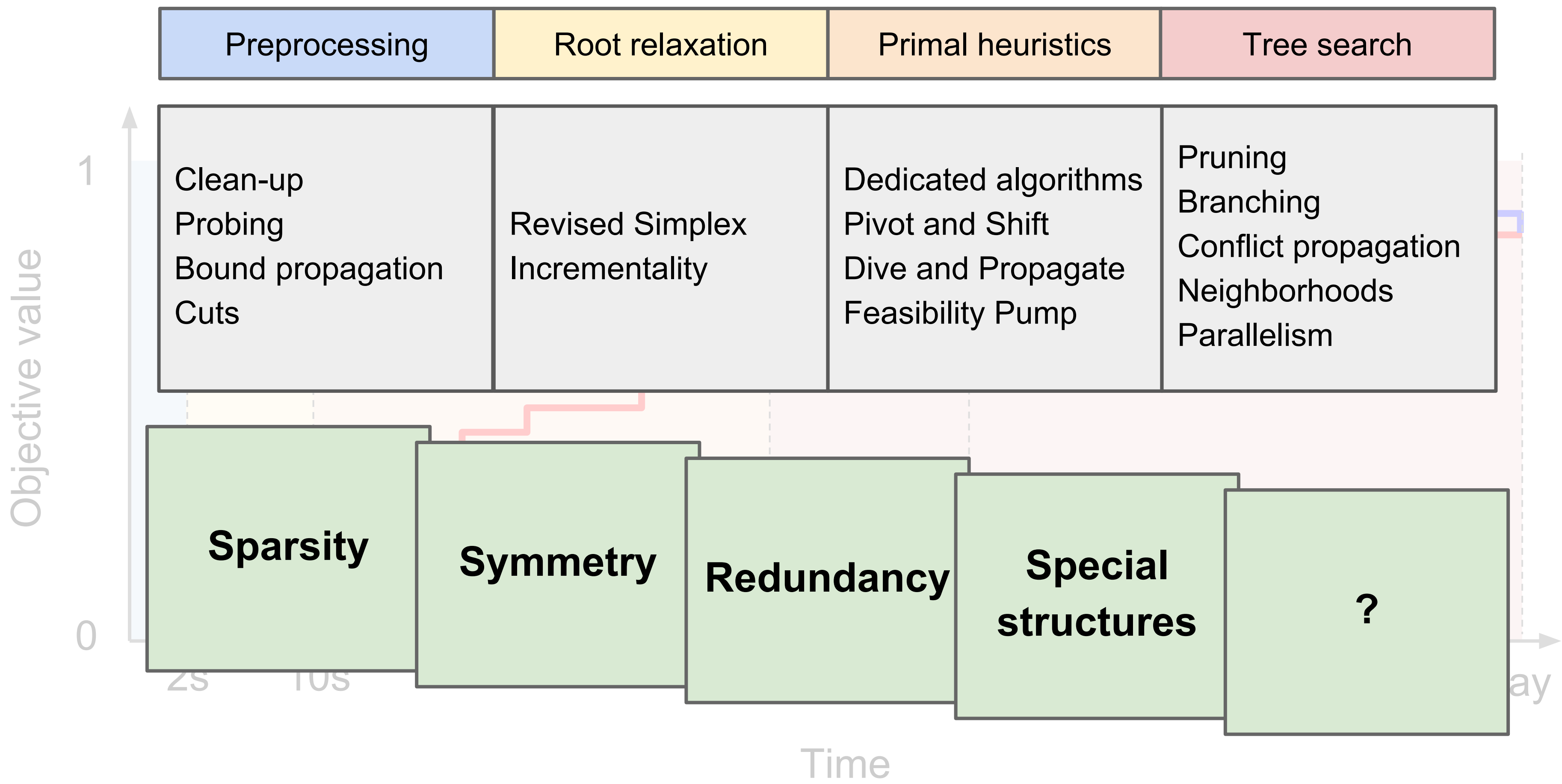
Search tree: Parallelism



Berthold, Farmer, Heinz, Perregaard (2016) "Parallelization of the FICO Xpress-Optimizer", LNCS 9725, 251-258



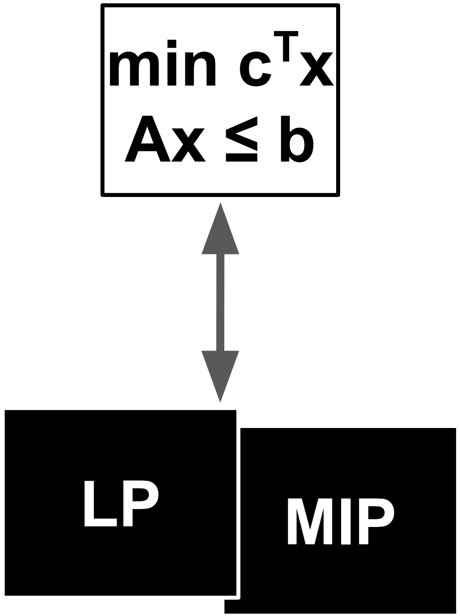
Preprocessing
Root relaxation
Primal heuristics
Tree search
Clean-up
Probing
Bound propagation
Cuts
Revised Simplex
Incrementality
Dedicated algorithms
Pivot and Shift
Dive and Propagate
Feasibility Pump
Pruning
Branching
Conflict propagation
Neighborhoods
Parallelism
Objective value
1
0
2s
10s
120s
180s
1 day
Time


Preprocessing
Root relaxation
Primal heuristics
Tree search
Clean-up
Probing
Bound propagation
Cuts
Revised Simplex
Incrementality
Dedicated algorithms
Pivot and Shift
Dive and Propagate
Feasibility Pump
Pruning
Branching
Conflict propagation
Neighborhoods
Parallelism
Sparsity
Symmetry
Redundancy
Special structures
?
Objective value
1
0
Time

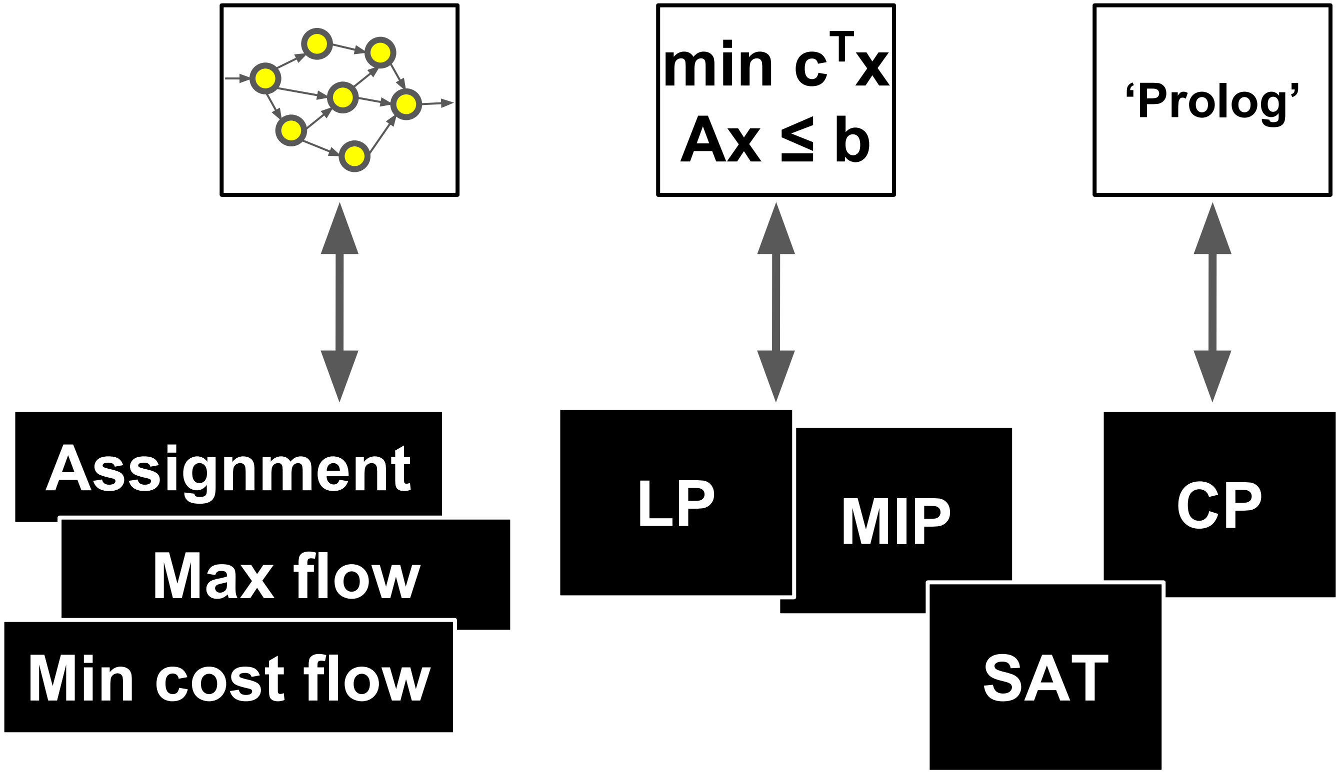
Outline

- Why do we use MIP?
  - Engineering
  - Efficiency
- Why are MIP solvers efficient?
  - Solver
  - Model
- **Self-doubt**

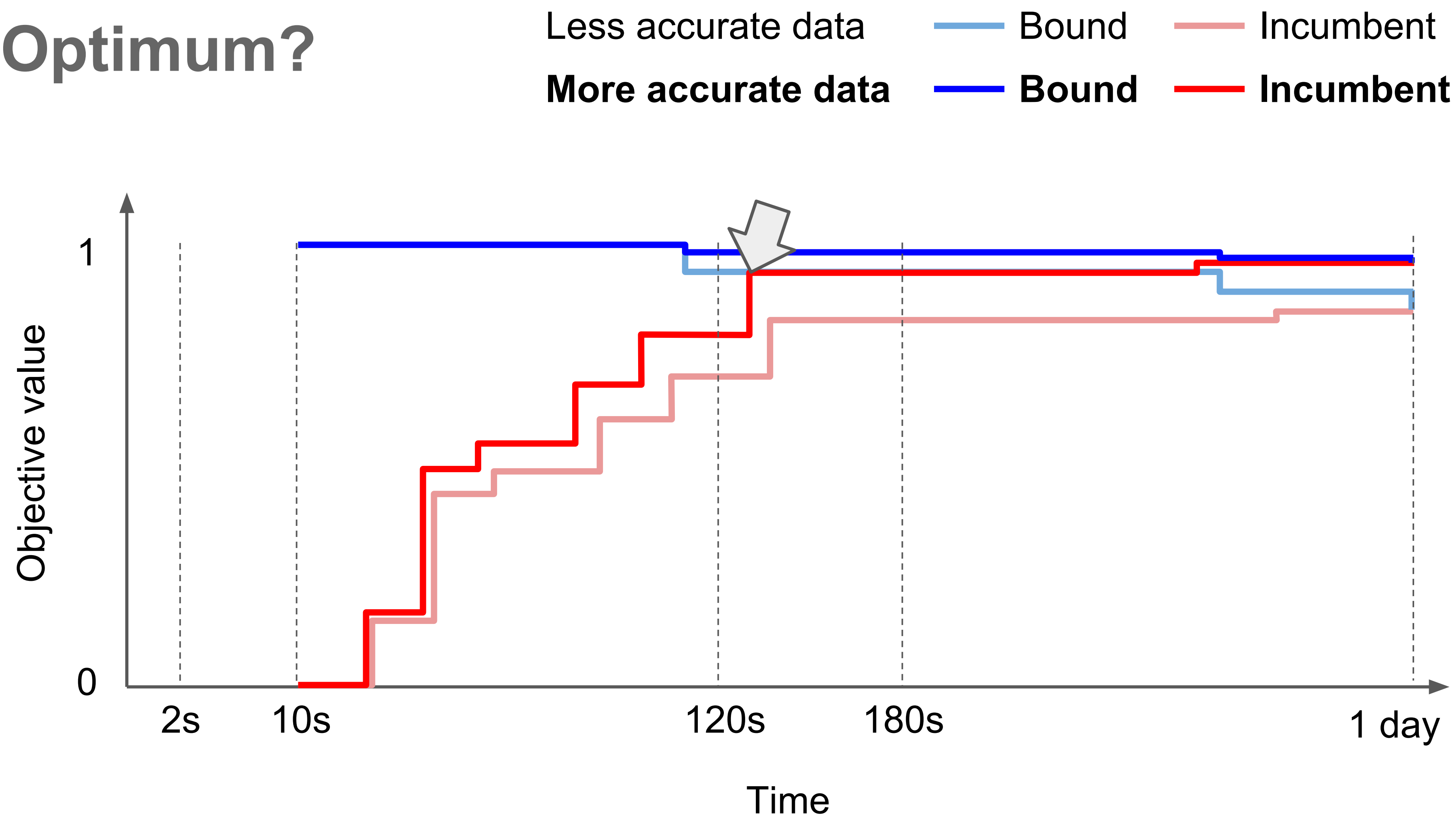
Other models?



Other models?



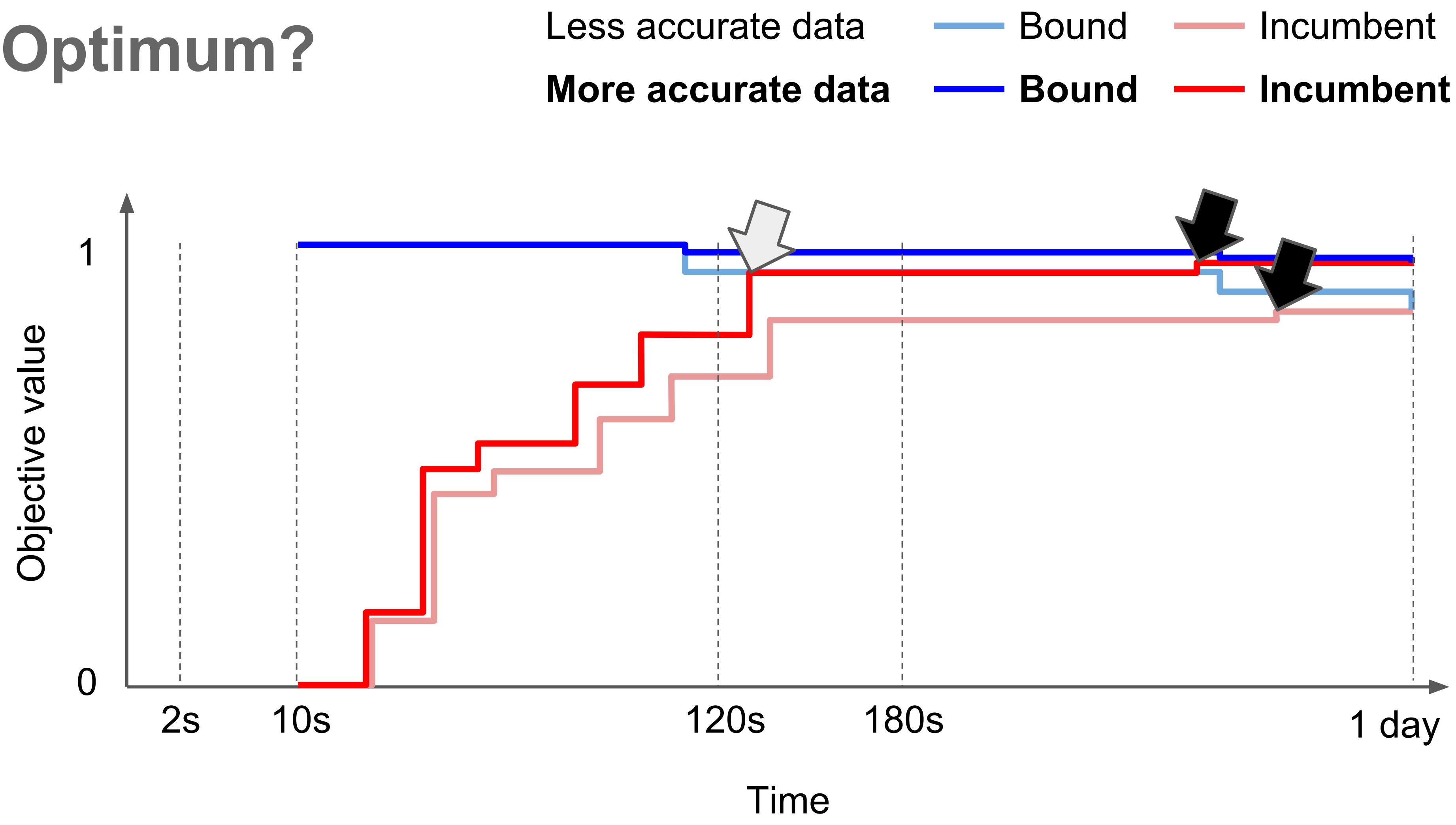
Optimum?



Less accurate data
Bound
Incumbent
More accurate data
Bound
Incumbent
1
0
Objective value
2s
10s
120s
180s
1 day
Time

```
Indices                  Variables                                Constants
  Item i = 1..I            place(i, b) in [0..Copies(i)]            int Copies(i)
  Bin b = 1..B             surplus(b) in [0, +inf)                  double Required(i, r)
  Resource r = 1..R        max_surplus in [0, +inf)                 double Available(b, r)
                           diff(i, b) in [0, Copies(i)]             int Placed(i, b)
Constraints
  for item i = 1..I:
    ∑b = 1..B place(i, b) = Copies(i)

  for resource r = 1..R:
    for bin b = 1..B:
      ∑i = 1..I Required(i, r) * place(i, b) ≤ Available(b, r)

  for bin b = 1..B:
    ∑i = 1..I Copies(i) / B - ∑i = 1..I place(i, b) ≤ surplus(b)
    surplus(b) ≤ max_surplus
    for item i = 1..I:
      Placed(i, b) - place(i, b) ≤ diff(i, b)

Objective
  min 1e6 max_surplus + ∑b = 1..B surplus(b) + 1e-3 ∑b = 1..B ∑i = 1..I diff(i, b)
```

Optimum?


Less accurate data
Bound
Incumbent
More accurate data
Bound
Incumbent
Objective value
1
0
2s
10s
120s
180s
1 day
Time

Summary

Why do we use MIP?

- **Engineering**
- **Efficiency**

Why are MIP solvers efficient?

- **Solver**
- **Model**

Self-doubt