A Search-Infer-and-Relax Framework for Integrating Solution Methods

John Hooker
Carnegie Mellon University

CPAIOR, May 2005

Why integrate solution methods?

- One-stop shopping.
  - One solver does it all.

Why integrate solution methods?

- One-stop shopping.
  - One solver does it all.
- More modeling options.
  - Natural models, less debugging & development time.

Why integrate solution methods?

- One-stop shopping.
  - One solver does it all.
- More modeling options.
  - Natural models, less debugging & development time.
- Computational speedup.
  - A selection of results…

Computational Advantage of Integrating CP and MILP

Using CP + relaxation from MILP

	Problem	*Speedup*
Focacci, Lodi, Milano (1999)	Lesson timetabling	2 to 50 times faster than CP
Refalo (1999)	Piecewise linear costs	2 to 200 times faster than MILP
Hooker & Osorio (1999)	Flow shop scheduling, etc.	4 to 150 times faster than MILP.
Thorsteinsson & Ottosson (2001)*	Product configuration	30 to 40 times faster than CP, MILP

*Will discuss

Computational Advantage of Integrating CP and MILP

Using CP + relaxation from MILP

	Problem	*Speedup*
Sellmann & Fahle (2001)	Automatic recording	1 to 10 times faster than CP, MILP
Van Hoeve (2001)	Stable set problem	Better than CP in less time
Bollapragada, Ghattas & Hooker (2001)	Structural design (nonlinear)	Up to 600 times faster than MILP
Beck & Refalo (2003)	Scheduling with earliness & tardiness costs	Solved 67 of 90, CP solved only 12

Computational Advantage of Integrating CP and MILP

Using CP-based Branch and Price

	Problem	*Speedup*
Yunes, Moura & de Souza (1999)	Urban transit crew scheduling	Optimal schedule for 210 trips, vs. 120 for traditional branch and price
Easton, Nemhauser & Trick (2002)	Traveling tournament scheduling	First to solve 8-team instance

Computational Advantage of Integrating CP and MILP

Using CP/MILP Benders methods

	Problem	*Speedup*
Jain & Grossmann (2001)*	Min-cost planning & scheduing	20 to 1000 times faster than CP, MILP
Thorsteinsson (2001)	Min-cost planning & scheduling	10 times faster than Jain & Grossmann
Timpe (2002)	Polypropylene batch scheduling at BASF	Solved previously insoluble problem in 10 min

*Will discuss

Computational Advantage of Integrating CP and MILP

Using CP/MILP Benders methods

	Problem	*Speedup*
Benoist, Gaudin, Rottembourg (2002)	Call center scheduling	Solved twice as many instances as traditional Benders
Hooker (2004)	Min-cost, min-makespan planning & cumulative scheduling	100-1000 times faster than CP, MILP
Hooker (2005)	Min tardiness planning & cumulative scheduling	10-1000 times faster than CP, MILP

Proposal:

- View solution methods as **special cases** of the same basic algorithm.

Proposal:

- View solution methods as **special cases** of the same basic algorithm.
- “Hybrid” methods can then be viewed as **other** special cases of the same basic algorithm.

The basic algorithm:

- **Search:** Enumerate problem restrictions
  - *e.g., search tree nodes, Benders subproblems, neighborhoods*

The basic algorithm:

- **Search:** Enumerate problem restrictions
  - *e.g., search tree nodes, Benders subproblems, neighborhoods*
- **Infer:** Deduce constraints from current restriction
  - *e.g., Reduced domains, cutting planes, nogoods, Benders cut*

The basic algorithm:

- **Search:** Enumerate problem restrictions
  - *e.g., search tree nodes, Benders subproblems, neighborhoods*
- **Infer:** Deduce constraints from current restriction
  - *e.g., Reduced domains, cutting planes, nogoods, Benders cuts*
- **Relax**: Solve relaxation of current restriction
  - *e.g., domain store, linear programming relaxation, Lagrangean relaxation, Benders master problem.*

The basic algorithm:

- **Search:** Enumerate problem restrictions
  - *e.g., search tree nodes, Benders subproblems, neighborhoods*
- **Infer:** Deduce constraints from current restriction
  - *e.g., Reduced domains, cutting planes, nogoods, Benders cuts*
- **Relax**: Solve relaxation of current restriction
  - *e.g., domain store, linear programming relaxation, Lagrangean relaxation, Benders master problem*
  - **Selection function** determines which solution of relaxation to use.

The basic algorithm:

- **Search:** Enumerate problem restrictions
  - *e.g., search tree nodes, Benders subproblems, neighborhoods*
- **Infer:** Deduce constraints from current restriction
  - *e.g., Reduced domains, cutting planes, nogoods, Benders cuts*
- **Relax**: Solve relaxation of current restriction
  - *e.g., domain store, linear programming relaxation, Lagrangean relaxation, Benders master problem*
  - **Selection function** determines which solution of relaxation to use.
  - Use **post-relaxation inference** if desired.
  - Solution of relaxation guides choice of next restriction.

The basic algorithm:

- **Search:** Enumerate problem restrictions $P_1, \ldots, P_k$

The basic algorithm:

- **Search:** Enumerate problem restrictions $P_1, \ldots, P_k$
- **Infer:** Deduce constraints from each P_i and add them to P_i.

The basic algorithm:

- **Search:** Enumerate problem restrictions $P_1, \ldots, P_k$
- **Infer:** Deduce constraints from each P_i and add them to P_i
- **Relax**: Formulate a relaxation R_i of each P_i.

The basic algorithm:

- **Search:** Enumerate problem restrictions $P_1, \ldots, P_k$
- **Infer:** Deduce constraints from each P_i and add them to P_i
- **Relax**: Formulate a relaxation R_i of each P_i.
  - Select a solution $s(R_i)$ of R_i.

The basic algorithm:

- **Search:** Enumerate problem restrictions $P_1, \ldots, P_k$
- **Infer:** Deduce constraints from each P_i and add them to P_i
- **Relax**: Formulate a relaxation R_i of each P_i.
  - Select a solution $s(R_i)$ of R_i.
  - Let $s(R_i)$ guide deduction of further constraints from P_i.

The basic algorithm:

- **Search:** Enumerate problem restrictions $P_1, \ldots, P_k$
- **Infer:** Deduce constraints from each P_i and add them to P_i
- **Relax**: Formulate a relaxation R_i of each P_i.
  - Select a solution $s(R_i)$ of R_i.
  - Let $s(R_i)$ guide deduction of further constraints from P_i.
  - Let $s(R_i)$ guide choice of P_{i+1}.

Underlying idea:

"Primal-dual" algorithm
exploits duality
of problem **restriction** and **relaxation**.

- The algorithm is interesting only when you see how it works out in examples.

- The algorithm is interesting only when you see how it works out in examples.
- Don’t read the paper.
  - It contains no examples and isn’t interesting.

- The algorithm is interesting only when you see how it works out in examples.
- Don't read the paper.
  - It contains no examples and isn't interesting.
  - If you must read it, read the revised version on my website.

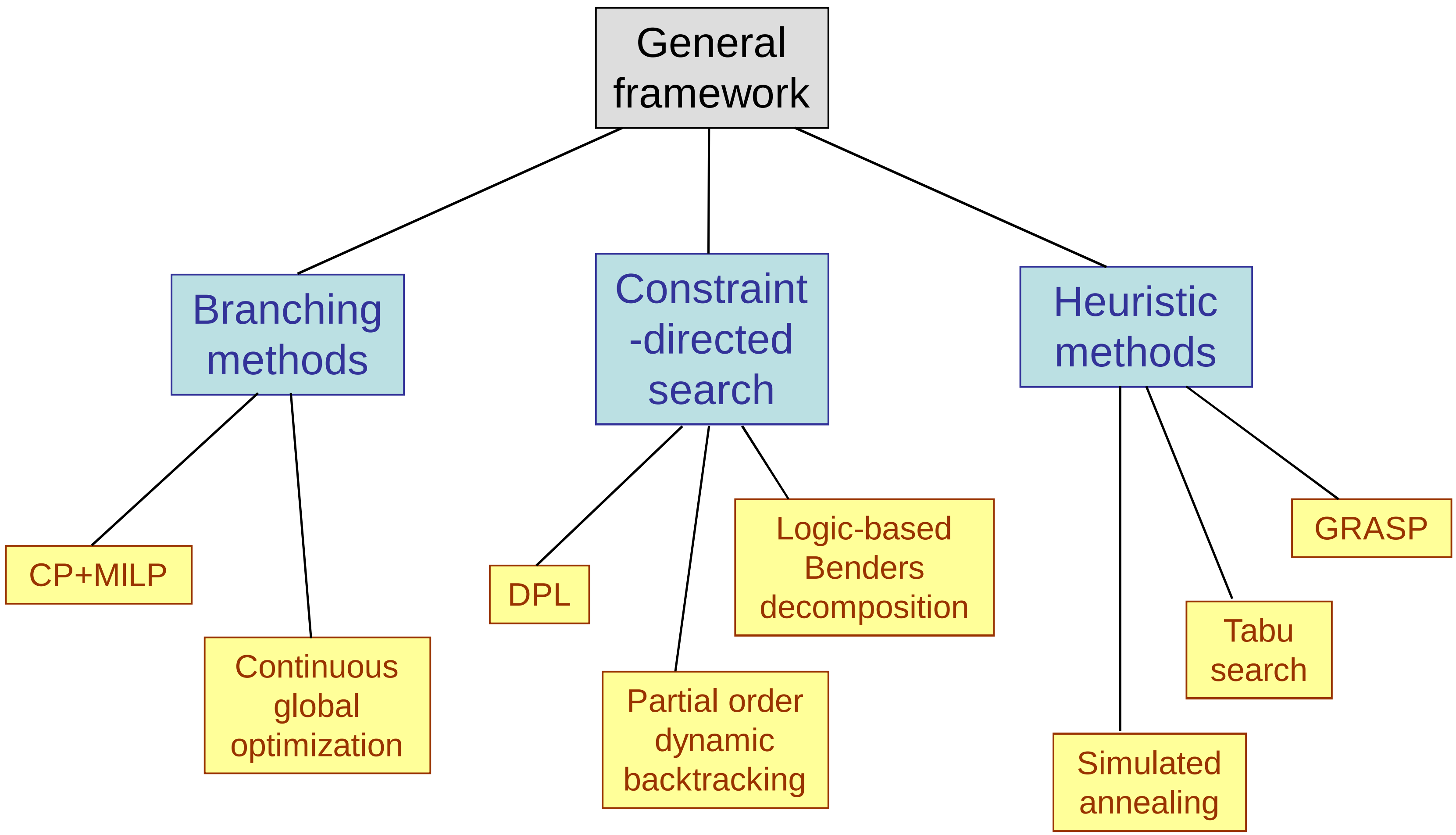
- The algorithm is interesting only when you see how it works out in examples.
- Don’t read the paper.
  - It contains no examples and isn’t interesting.
  - If you must read it, read the revised version on my website.
- After looking at the examples, you can judge whether this framework is helpful or artificial.

General Framework

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Branching CP, MILP, global optimization	Node of search tree	LP, NLP, domains	Any optimal (feasible) solution of R_k	Domain filtering, cutting planes
Constraint directed DPL, dynamic backtracking, Benders	Add nogoods generated so far	Processed nogoods generated so far	Solution that results in easy R_k	Nogood generation
Heuristics Local search, GRASP	Neighborhood of current solution	Research topic	Random, best, etc.	?

Structure of Talk

Depth-first traversal of tree:



General Framework

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Branching **CP, MILP, global optimization**	**Node of search tree**	**LP, NLP, domains**	**Any optimal (feasible) solution of R_k**	**Domain filtering, cutting planes**
Constraint directed DPL, dynamic backtracking, Benders	Add nogoods generated so far	Processed nogoods generated so far	Solution that results in easy R_k	Nogood generation
Heuristics Local search, GRASP	Neighborhood of current solution	Research topic	Random, best, etc.	?

Branching Methods

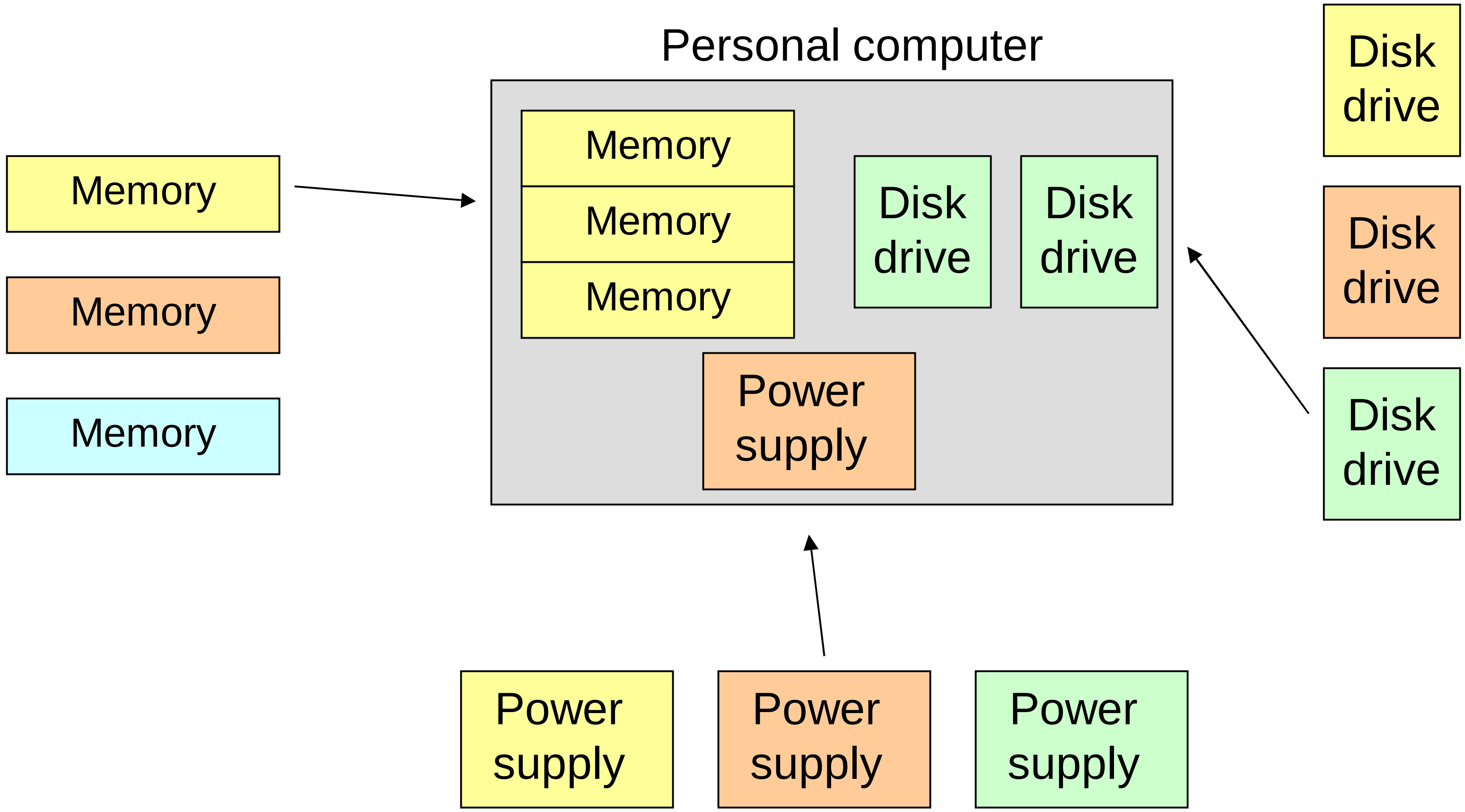
Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
CP	Created by splitting domain, etc.	Current domains	Any feasible solution of R_k	Domain filtering, propagation
MILP Branch and cut	Created by branching on fractional variables	LP relaxation + cutting planes	Optimal solution of R_k	Cutting planes, preprocessing
Continuous global optimization	Created by splitting intervals	LP or convex NLP relaxation	Optimal solution of R_k .	Interval propagation, Lagrangean bounding

Branching Methods

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
CP	**Created by splitting domain, etc.**	**Current domains**	**Any feasible solution of R_k**	**Domain filtering, propagation**
MILP Branch and cut	**Created by branching on fractional variables**	**LP relaxation + cutting planes**	**Optimal solution of R_k**	**Cutting planes, preprocessing**
Continuous global optimization	Created by splitting intervals	LP or convex NLP relaxation	Optimal solution of R_k .	Interval propagation, Lagrangean bounding

Branching Search:
Product Configuration by CP/MILP

Choose what type of each component,
and how many



This example illustrates
how a “hybrid” method
is a special case
of the general algorithm.

Model of the problem

Amount of attribute *j* produced
(< 0 if consumed):
memory, heat, power, weight, etc.

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \quad \text{all } j$$

$$L_j \le v_j \le U_j, \quad \text{all } j$$

Quantity of component *i* installed

Amount of attribute *j* produced by type t_i of component *i*

Amount of attribute *j* produced (< 0 if consumed): memory, heat, power, weight, etc.

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \quad \text{all } j$$

$$L_j \le v_j \le U_j, \quad \text{all } j$$

Quantity of component *i* installed

Amount of attribute *j* produced by type t_i of component *i*

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \ \text{all } j$$

$$L_j \le v_j \le U_j, \ \text{all } j$$

Amount of attribute *j* produced
(< 0 if consumed):
memory, heat, power, weight, etc.

t_i is a **variable index**

Quantity of component *i* installed

Unit cost of producing attribute *j*

Amount of attribute *j* produced by type t_i of component *i*

Amount of attribute *j* produced (< 0 if consumed): memory, heat, power, weight, etc.

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \quad \text{all } j$$

$$L_j \le v_j \le U_j, \quad \text{all } j$$

t_i is a **variable index**

Quantity of component *i* installed

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.
- **Infer:** propagate *indexed linear* constraint and bounds on v_j.

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.
- **Infer:** propagate *indexed linear* constraint and bounds on v_j.
  - **Variable index** is converted to specially structured *element* constraint, which is filtered.

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.
- **Infer:** propagate *indexed linear* constraint and bounds on v_j.
  - **Variable index** is converted to specially structured *element* constraint, which is filtered.
  - Valid **knapsack** cuts are derived and propagated.

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.
- **Infer:** propagate *indexed linear* constraint and bounds on v_j.
  - **Variable index** is converted to specially structured *element* constraint, which is filtered.
  - Valid **knapsack** cuts are derived and propagated.
- **Relax:** use linear continuous relaxations.

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.
- **Infer:** propagate *indexed linear* constraint and bounds on v_j.
  - **Variable index** is converted to specially structured *element* constraint, which is filtered.
  - Valid **knapsack** cuts are derived and propagated.
- **Relax:** use linear continuous relaxations.
  - Use special purpose relaxation for *element*

To solve it:

- **Search**: branch on domains of t_i and q_i.
  - Each **node** of search tree is a problem restriction.
- **Infer:** propagate *indexed linear* constraint and bounds on v_j.
  - **Variable index** is converted to specially structured *element* constraint, which is filtered.
  - Valid **knapsack** cuts are derived and propagated.
- **Relax:** use linear continuous relaxations.
  - Use special purpose relaxation for *element.*
  - **Selection function:** Any optimal solution of relaxation.

Infer (propagate)

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$\boxed{L_j \le v_j \le U_j, \text{ all } j}$$

This is propagated in the usual way

Infer (propagate)

$$v_j = \sum_i z_i, \ \text{all } j$$

$$\text{element}\big(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_i\big), \text{ all } i, j$$

$$\min \sum_j c_j v_j$$

$$\boxed{v_j = \sum_{ik} q_i A_{ijt_i}, \ \text{all } j}$$

$$\boxed{L_j \le v_j \le U_j, \ \text{all } j}$$

This is automatically rewritten as

This is propagated in the usual way

Infer (propagate)

$$v_j = \sum_i z_i, \ \text{all } j$$
$$\text{element}\big(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_i\big), \text{all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form…

Infer (propagate)

$$v_j = \sum_i z_i, \ \text{all } j$$
$$\text{element}\big(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_i\big), \text{all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form,
(b) adding **knapsack cuts** for the valid inequalities:

$$\sum_i \max_{k \in D_{t_i}} \{A_{ijk}\} q_i \geq \underline{v}_j, \ \text{all } j$$
$$\sum_i \min_{k \in D_{t_i}} \{A_{ijk}\} q_i \leq \overline{v}_j, \ \text{all } j$$

$[\underline{v}_j, \overline{v}_j]$ is current domain of v_j

Infer (propagate)

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\big(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_i\big), \text{ all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form,
(b) adding **knapsack cuts** for the valid inequalities:

$$\sum_i \max_{k \in D_{t_i}} \{A_{ijk}\} q_i \geq \underline{v}_j, \text{ all } j$$

$$\sum_i \min_{k \in D_{t_i}} \{A_{ijk}\} q_i \leq \bar{v}_j, \text{ all } j$$

and (c) propagating the knapsack cuts.

$[\underline{v}_j, \bar{v}_j]$ is current domain of v_j

Relax

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \ \text{all } j$$

$$\boxed{L_j \le v_j \le U_j, \ \text{all } j}$$

This is relaxed as

$$\underline{v}_j \le v_j \le \bar{v}_j$$

Relax

$$v_j = \sum_i z_i, \ \text{all } j$$

$$\text{element}\left(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

$$\min \sum_j c_j v_j$$

$$\boxed{v_j = \sum_{ik} q_i A_{ijt_i}, \ \text{all } j}$$

$$\boxed{L_j \le v_j \le U_j, \ \text{all } j}$$

This is relaxed by relaxing *this* and adding the knapsack cuts.

This is relaxed as

$$\underline{v}_j \le v_j \le \bar{v}_j$$

Relax

$$v_j = \sum_i z_i, \ \text{all } j$$

$$\text{element}\big(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_i\big), \text{ all } i, j$$

This is relaxed by replacing each *element* constraint with a **convex hull** relaxation of a disjunctive programming constraint:

$$z_i = \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \quad q_i = \sum_{k \in D_{t_i}} q_{ik}$$

Relax

So the following LP relaxation is solved at each node of the search tree to obtain a lower bound:

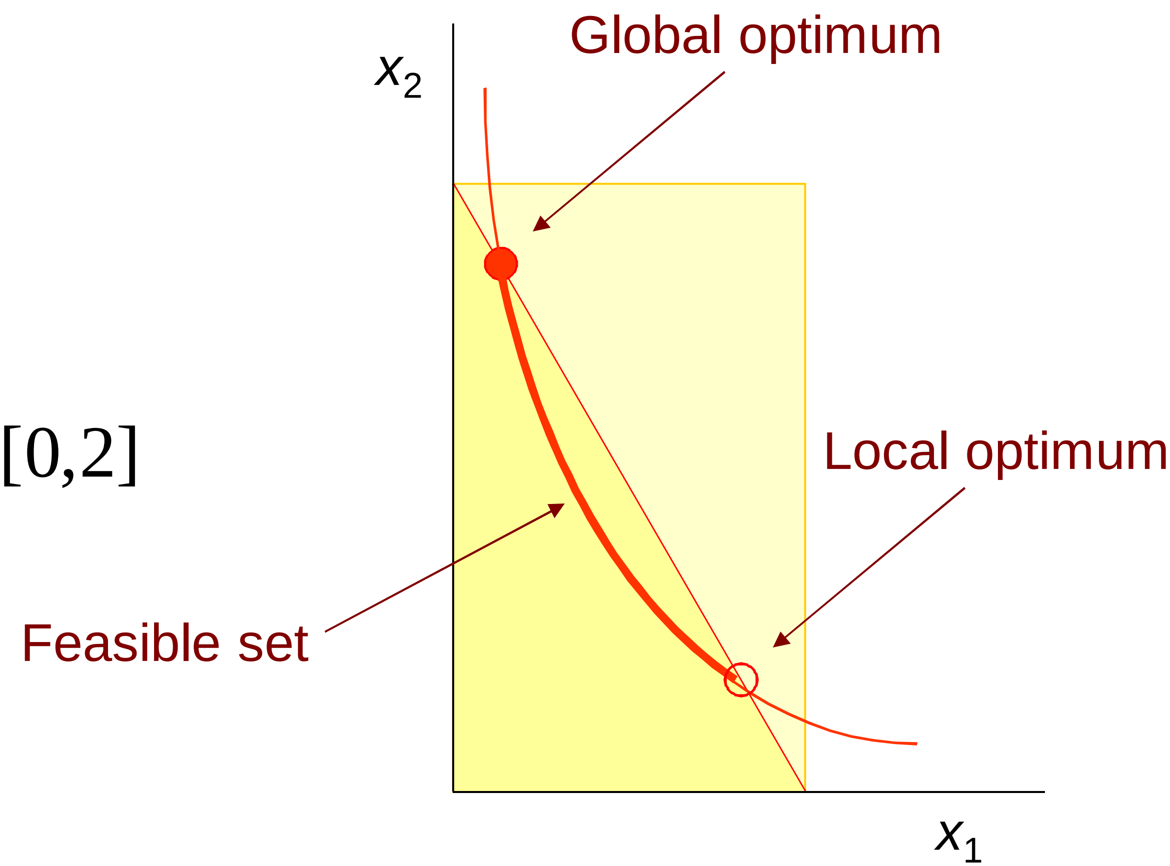
$$
\begin{aligned}
&\min \sum_j c_j v_j \\
&v_j = \sum_i \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \ \text{all } j \\
&q_i = \sum_{k \in D_{t_i}} q_{ik}, \ \text{all } i \\
&\underline{v}_j \le v_j \le \overline{v}_j, \ \text{all } j \\
&\underline{q}_i \le q_i \le \overline{q}_i, \ \text{all } i \\
&\text{knapsack cuts for } \sum_i \max_{k \in D_{t_i}} \{A_{ijk}\} q_i \ge \underline{v}_j, \ \text{all } j \\
&\text{knapsack cuts for } \sum_i \min_{k \in D_{t_i}} \{A_{ijk}\} q_i \le \overline{v}_j, \ \text{all } j \\
&q_{ik} \ge 0, \ \text{all } i, k
\end{aligned}
$$

Branching Methods

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
CP	Created by splitting domain, etc.	Current domains	Any feasible solution of R_k	Domain filtering, propagation
MILP Branch and cut	Created by branching on fractional variables	LP relaxation + cutting planes	Optimal solution of R_k	Cutting planes, preprocessing
Continuous global optimization	**Created by splitting intervals**	**LP or convex NLP relaxation**	**Optimal solution of R_k** .	**Interval propagation, Lagrangean bounding**

Branching Search: Continuous Global Optimization

$$\max x_1 + x_2$$
$$4x_1 x_2 = 1$$
$$2x_1 + x_2 \le 2$$
$$x_1 \in [0,1],\ x_2 \in [0,2]$$



To solve it:

- **Search**: split interval domains of x_1, x_2.
  - Each **node** of search tree is a problem restriction.

To solve it:

- **Search**: split interval domains of x_1, x_2.
  - Each **node** of search tree is a problem restriction.
- **Infer:** Interval propagation, post-relaxation inference.

To solve it:

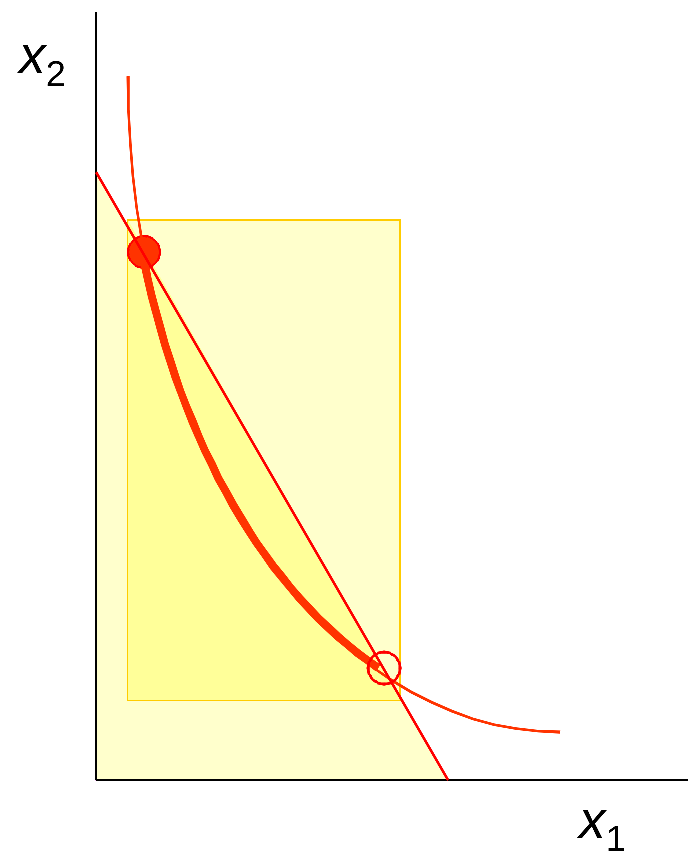
- **Search**: split interval domains of x_1, x_2.
  - Each **node** of search tree is a problem restriction.
- **Infer:** Interval propagation, post-relaxation inference.
  - Post-relaxation inference: Use **Lagrange multipliers** to infer valid inequality for propagation.
  - **Reduced-cost variable** fixing is a special case.

To solve it:

- **Search**: split interval domains of x_1, x_2.
  - Each **node** of search tree is a problem restriction.
- **Infer:** Interval propagation, post-relaxation inference.
  - Post-relaxation inference: Use **Lagrange multipliers** to infer valid inequality for propagation.
  - **Reduced-cost variable** fixing is a special case.
- **Relax:** Use function **factorization** to obtain linear continuous relaxation.
  - **Selection function:** Any optimal solution of relaxation.

Infer (interval propagation)

Propagate intervals
[0,1], [0,2]
through constraints
to obtain
[1/8,7/8], [1/4,7/4]



Relax (function factorization)

Factor complex functions into elementary functions that have known linear relaxations.

Relax (function factorization)

Factor complex functions into elementary functions that have known linear relaxations.

Write $4x_1x_2 = 1$ as $4y = 1$ where $y = x_1x_2$.

This factors $4x_1x_2$ into linear function $4y$ and bilinear function x_1x_2.

Relax (function factorization)

Factor complex functions into elementary functions that have known linear relaxations.

Write $4x_1x_2 = 1$ as $4y = 1$ where $y = x_1x_2$.

This factors $4x_1x_2$ into linear function $4y$ and bilinear function x_1x_2.

Linear function $4y$ is its own linear relaxation.

Relax (function factorization)

Factor complex functions into elementary functions that have known linear relaxations.

Write $4x_1x_2 = 1$ as $4y = 1$ where $y = x_1x_2$.

This factors $4x_1x_2$ into linear function $4y$ and bilinear function x_1x_2.

Linear function $4y$ is its own linear relaxation.

Bilinear function $y = x_1x_2$ has relaxation:

$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$

$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$

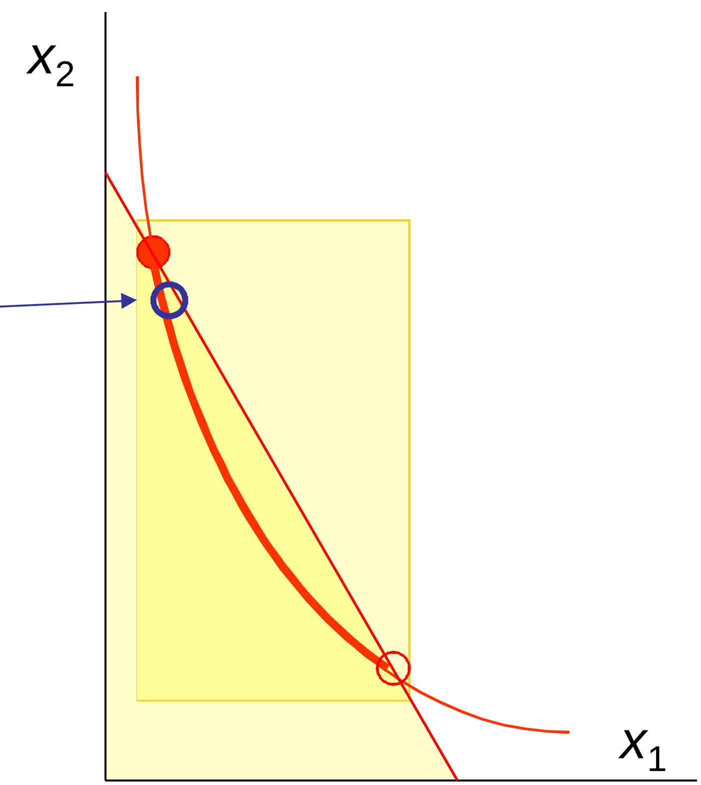
Where domain of x_j is $[\underline{x}_j, \bar{x}_j]$

Relax

The linear relaxation becomes:

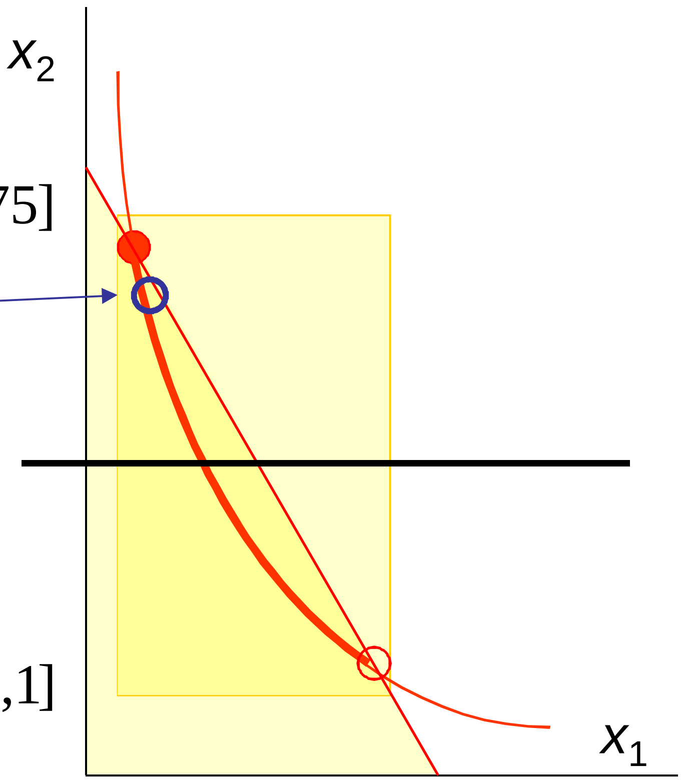
$$
\begin{aligned}
&\min x_1 + x_2 \\
&4y = 1 \\
&2x_1 + x_2 \le 2 \\
&\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2 \\
&\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2 \\
&\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2
\end{aligned}
$$

Relax



Solve linear relaxation.

Relax

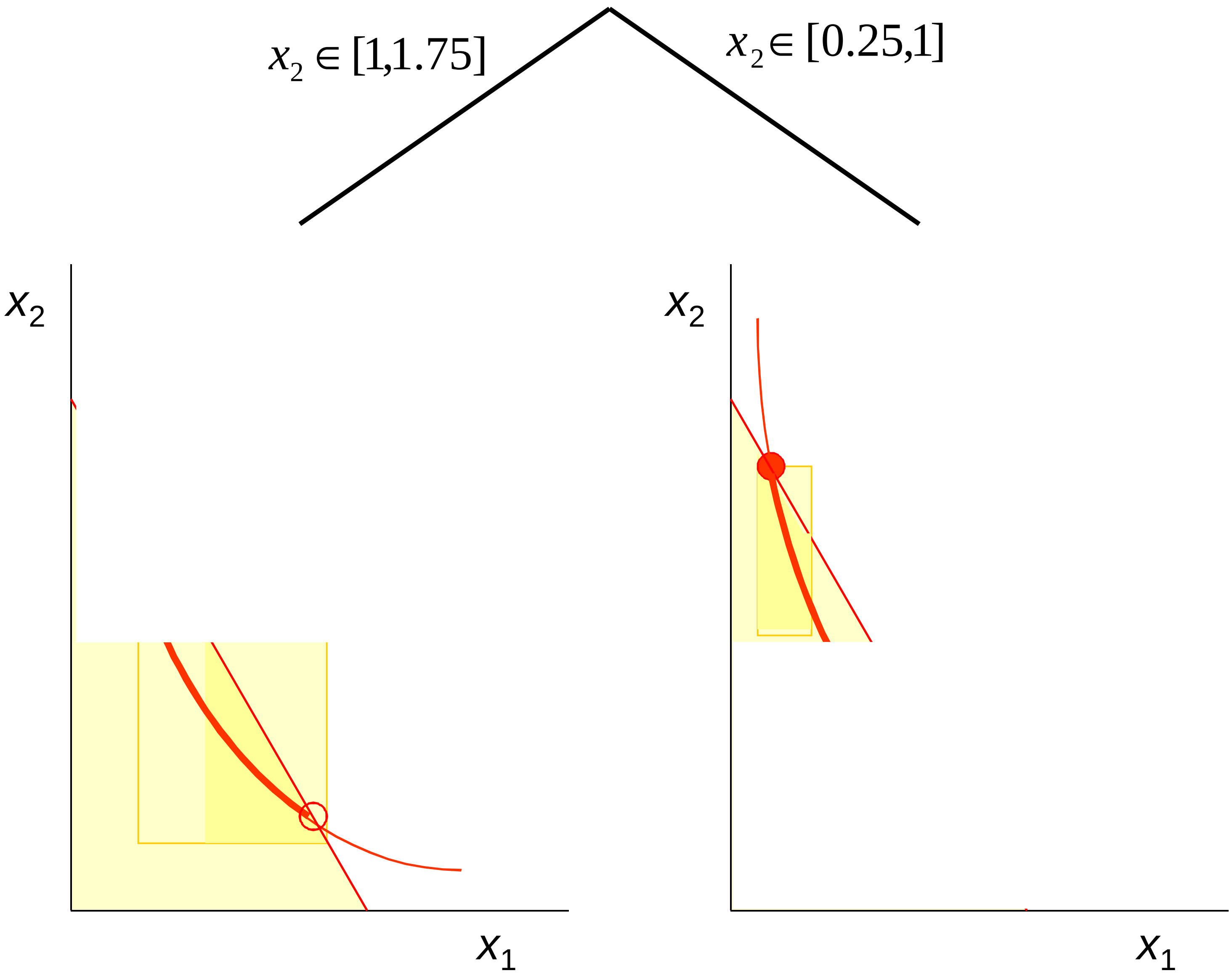


$x_2 \in [1,1.75]$

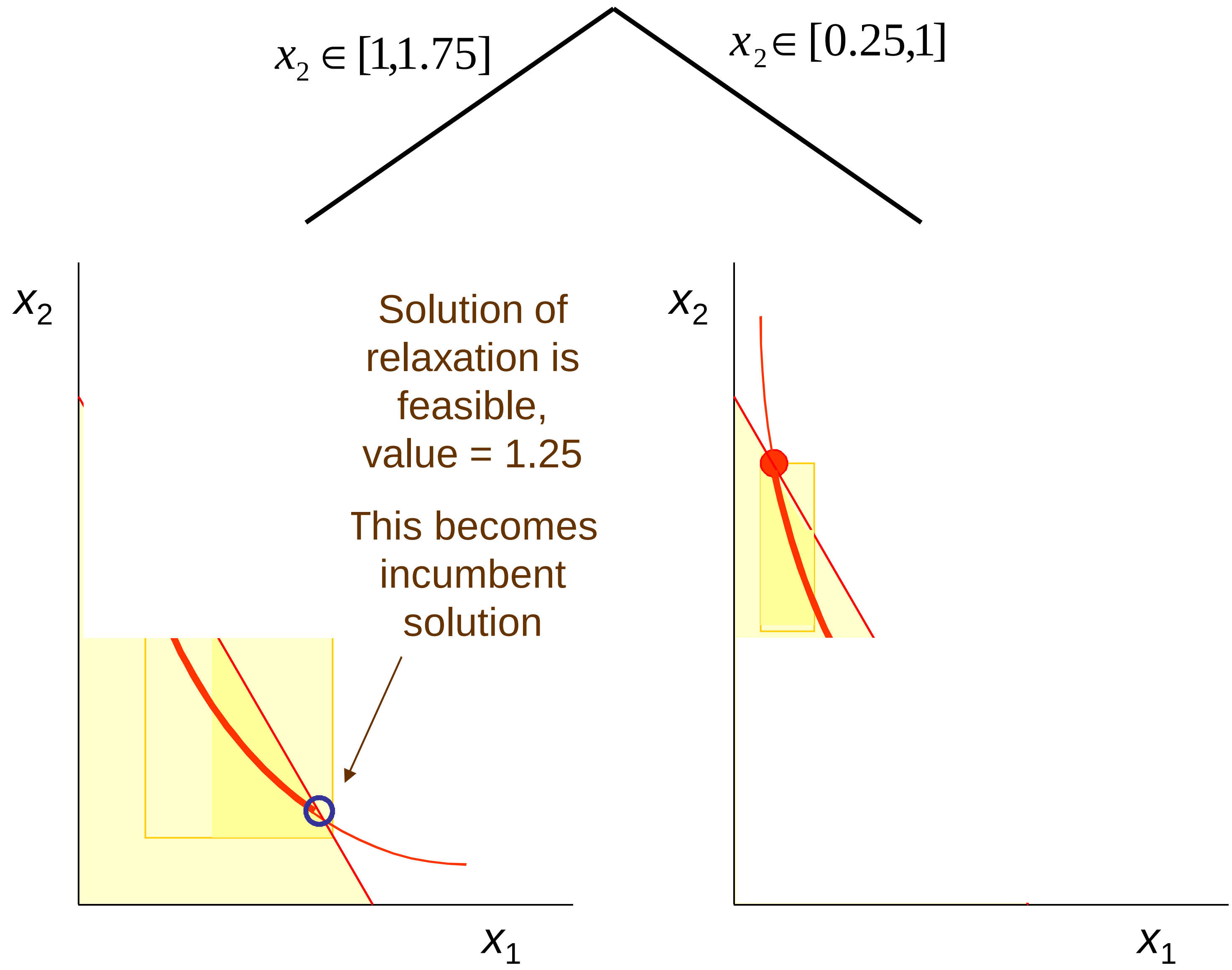
Solve linear relaxation.

Since solution is infeasible,
split an interval and branch.

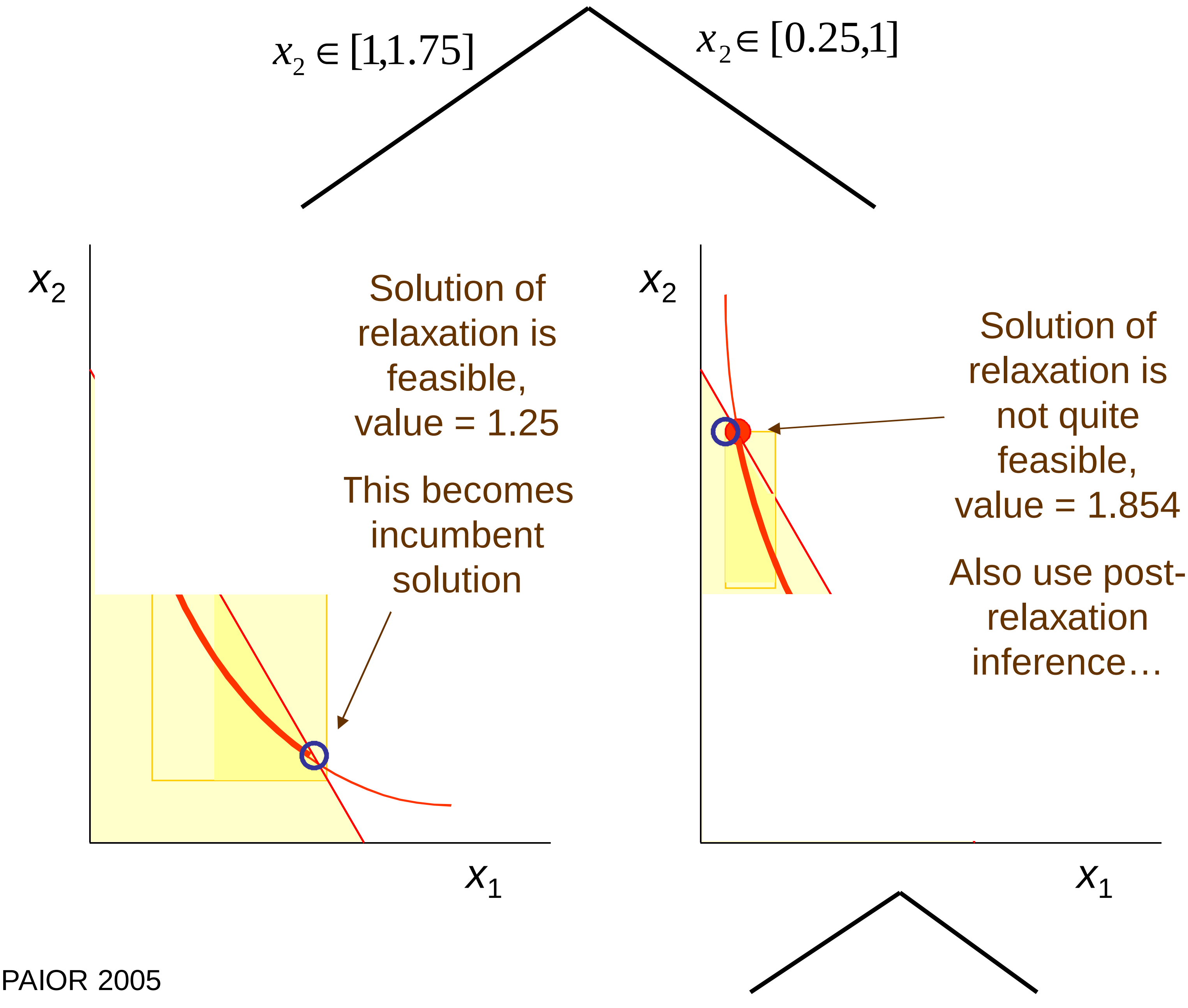
$x_2 \in [0.25,1]$



$x_2 \in [1,1.75]$
$x_2 \in [0.25,1]$
x_2
x_1
x_2
x_1



$x_2 \in [1,1.75]$
$x_2 \in [0.25,1]$
x_2
x_1
Solution of relaxation is feasible, value = 1.25
This becomes incumbent solution
x_2
x_1



$x_2 \in [1,1.75]$
$x_2 \in [0.25,1]$
x_2
x_1
Solution of relaxation is feasible, value = 1.25
This becomes incumbent solution
x_2
x_1
Solution of relaxation is not quite feasible, value = 1.854
Also use post-relaxation inference…

Post-Relaxation Inference

$$
\begin{aligned}
&\min x_1 + x_2 \\
&4y = 1 \\
&2x_1 + x_2 \le 2 \\
&\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2 \\
&\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2 \\
&\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2
\end{aligned}
$$

Associated Lagrange multiplier in solution of relaxation is 1.1

Post-Relaxation Inference

$$\min x_1 + x_2$$

$$4y = 1$$

$$2x_1 + x_2 \le 2$$

Associated Lagrange multiplier in solution of relaxation is 1.1

$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$

$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$

$$\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2$$

This yields a valid inequality for propagation:

$$2x_1 + x_2 \ge 2 - \frac{1.854 - 1.25}{1.1} = 1.451$$

Post-Relaxation Inference

$$\min x_1 + x_2$$

$$4y = 1$$

$$2x_1 + x_2 \le 2$$

Associated Lagrange multiplier in solution of relaxation is 1.1

$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$

$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$

$$\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2$$

This yields a valid inequality for propagation:

$$2x_1 + x_2 \ge 2 - \frac{\boxed{1.854} - 1.25}{1.1} = 1.451$$

Value of relaxation

Post-Relaxation Inference

$$\min x_1 + x_2$$

$$4y = 1$$

$$2x_1 + x_2 \le 2$$

Associated Lagrange multiplier in solution of relaxation is 1.1

$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$

$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$

$$\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2$$

This yields a valid inequality for propagation:

$$2x_1 + x_2 \ge 2 - \frac{\boxed{1.854} - 1.25}{\boxed{1.1}} = 1.451$$

Value of relaxation

Lagrange multiplier

Post-Relaxation Inference

$$
\begin{aligned}
&\min x_1 + x_2 \\
&4y = 1 \\
&2x_1 + x_2 \le 2 \\
&\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2 \\
&\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2 \\
&\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2
\end{aligned}
$$

Associated Lagrange multiplier in solution of relaxation is 1.1

This yields a valid inequality for propagation:

$$2x_1 + x_2 \ge 2 - \frac{\boxed{1.854} - \boxed{1.25}}{\boxed{1.1}} = 1.451$$

Value of relaxation

Lagrange multiplier

Value of incumbent solution

Post-Relaxation Inference

- **Reduced-cost variable fixing** is a special case.

Post-Relaxation Inference

- **Reduced-cost variable fixing** is a special case.
- **Separating cuts** represent another form of post-relaxation inference.

General Framework

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Branching CP, MILP, global optimization	Node of search tree	LP, NLP, domains	Any optimal (feasible) solution of R_k	Domain filtering, cutting planes
Constraint directed **DPL, dynamic backtracking, Benders**	**Add nogoods generated so far**	**Processed nogoods generated so far**	**Solution that results in easy R_k**	**Nogood generation**
Heuristics Local search, GRASP	Neighborhood of current solution	Research topic	Random, best, etc.	?

Constraint-Directed Search

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
DPL for SAT	Add conflict clauses	Processed conflict clauses	Unit clause rule + greedy solution of R_k	Parallel resolution & absorption
Partial order dynamic backtracking	Add nogoods	Processed nogoods	Greedy, consistent with partial order	Parallel resolution & absorption
Logic-based Benders	Subproblem defined by solution of master	Master problem (Benders cuts)	Optimal solution of master	Benders cuts (nogoods)

Constraint-Directed Search

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
DPL for SAT	**Add conflict clauses**	**Processed conflict clauses**	**Unit clause rule + greedy solution of R_k**	**Parallel resolution & absorption**
Partial order dynamic backtracking	Add nogoods	Processed nogoods	Greedy, consistent with partial order	Parallel resolution & absorption
Logic-based Benders	Subproblem defined by solution of master	Master problem (Benders cuts)	Optimal solution of master	Benders cuts (nogoods)

Constraint-Directed Search: DPL for Propositional Satisfiability

DPL
(Davis-Putnam-Loveland)
with clause learning
can be interpreted as
constraint-directed search

$$x_1 \vee x_5 \vee x_6$$
$$x_1 \vee x_5 \vee \bar{x}_6$$
$$x_2 \vee \bar{x}_5 \vee x_6$$
$$x_2 \vee \bar{x}_5 \vee \bar{x}_6$$
$$\bar{x}_1 \vee x_3 \vee x_4$$
$$\bar{x}_2 \vee x_3 \vee x_4$$
$$\bar{x}_1 \vee \bar{x}_3$$
$$\bar{x}_1 \vee \bar{x}_4$$
$$\bar{x}_2 \vee \bar{x}_3$$
$$\bar{x}_2 \vee \bar{x}_4$$

To solve it by branching:

- **Search**: branch on x_j.
  - Each **node** of search tree is a problem restriction.

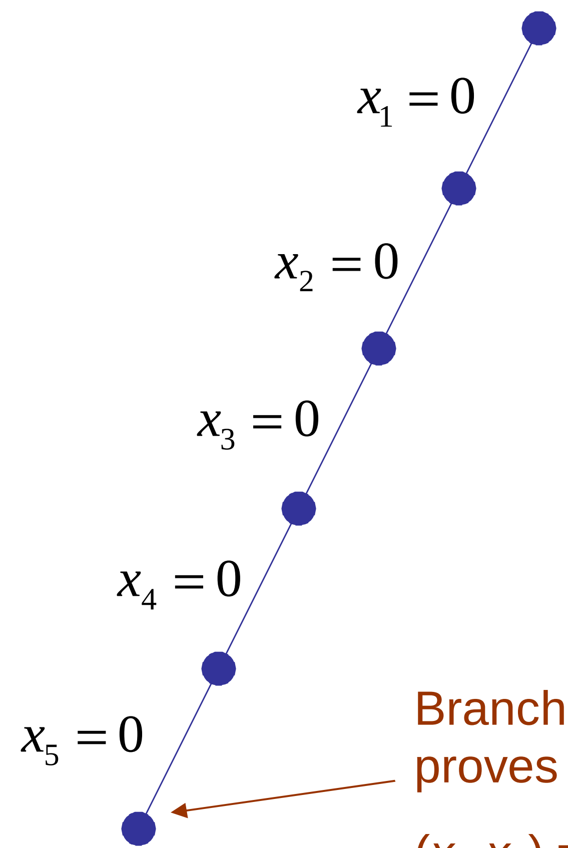
To solve it by branching:

- **Search**: branch on x_j.
  - Each **node** of search tree is a problem restriction.
- **Infer:** clause learning, unit clause rule.

To solve it by branching:

- **Search**: branch on x_j.
  - Each **node** of search tree is a problem restriction.
- **Infer:** clause learning, unit clause rule.
- **Relax:** not used.

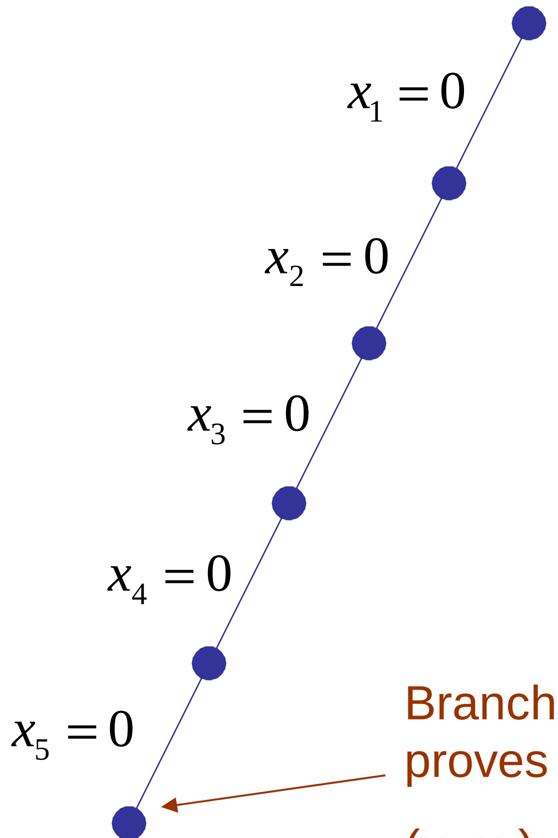
Branching



Branch to here. Unit clause rule proves infeasibility.

$(x_1, x_5) = (0,0)$ creates the conflict.

Branching



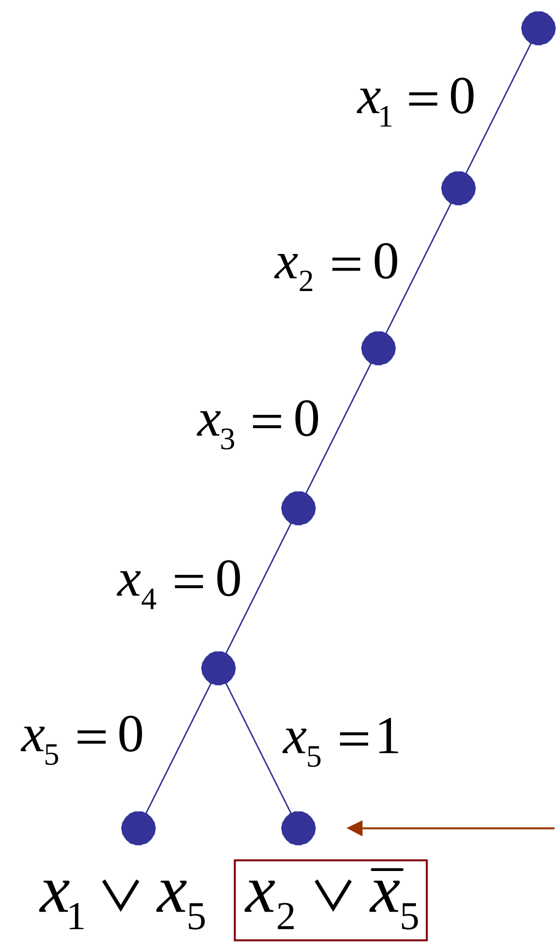
$\boxed{x_1 \vee x_5}$

Branch to here. Unit clause rule proves infeasibility.

$(x_1,x_5) = (0,0)$ creates the conflict.

Add conflict clause to constraint set.

Branching

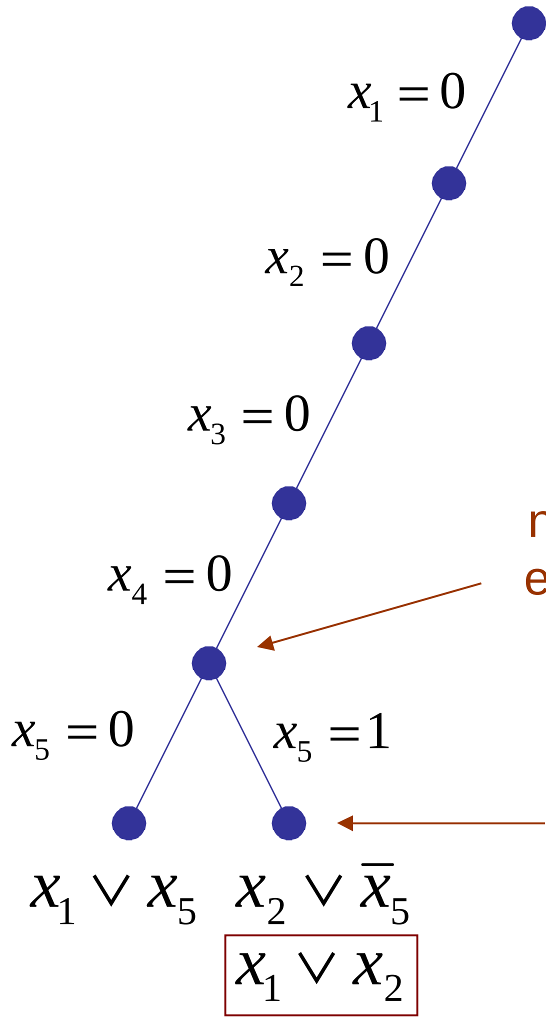


Backtrack and branch to here.

$(x_1,x_5) = (0,1)$ creates the conflict.

Generate conflict clause

Branching



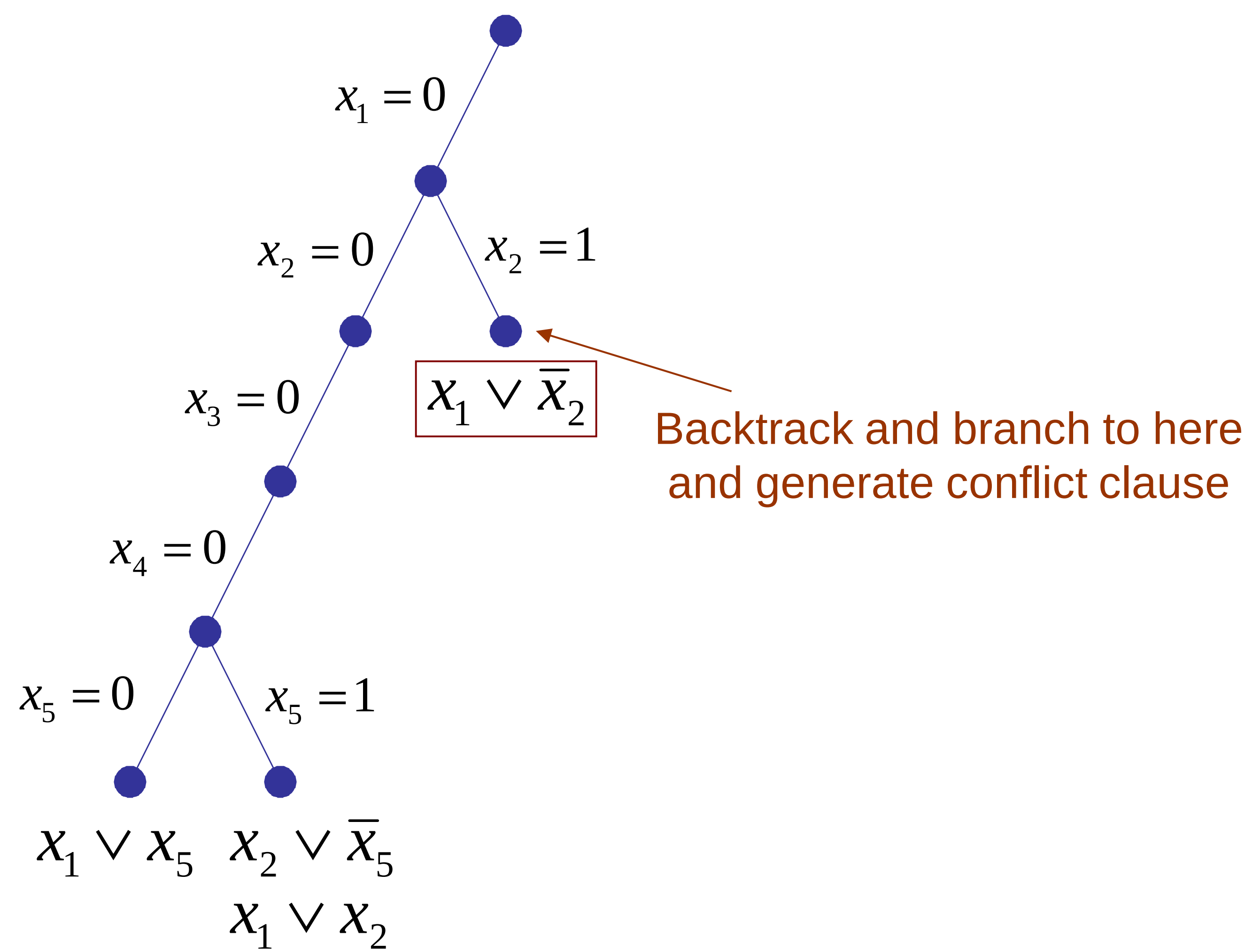
Backtrack to here and note that $(x_1, x_2) = (0,0)$ is enough to create conflict. Generate new conflict clause

Backtrack and branch to here.

$(x_1, x_5) = (0,1)$ creates the conflict.

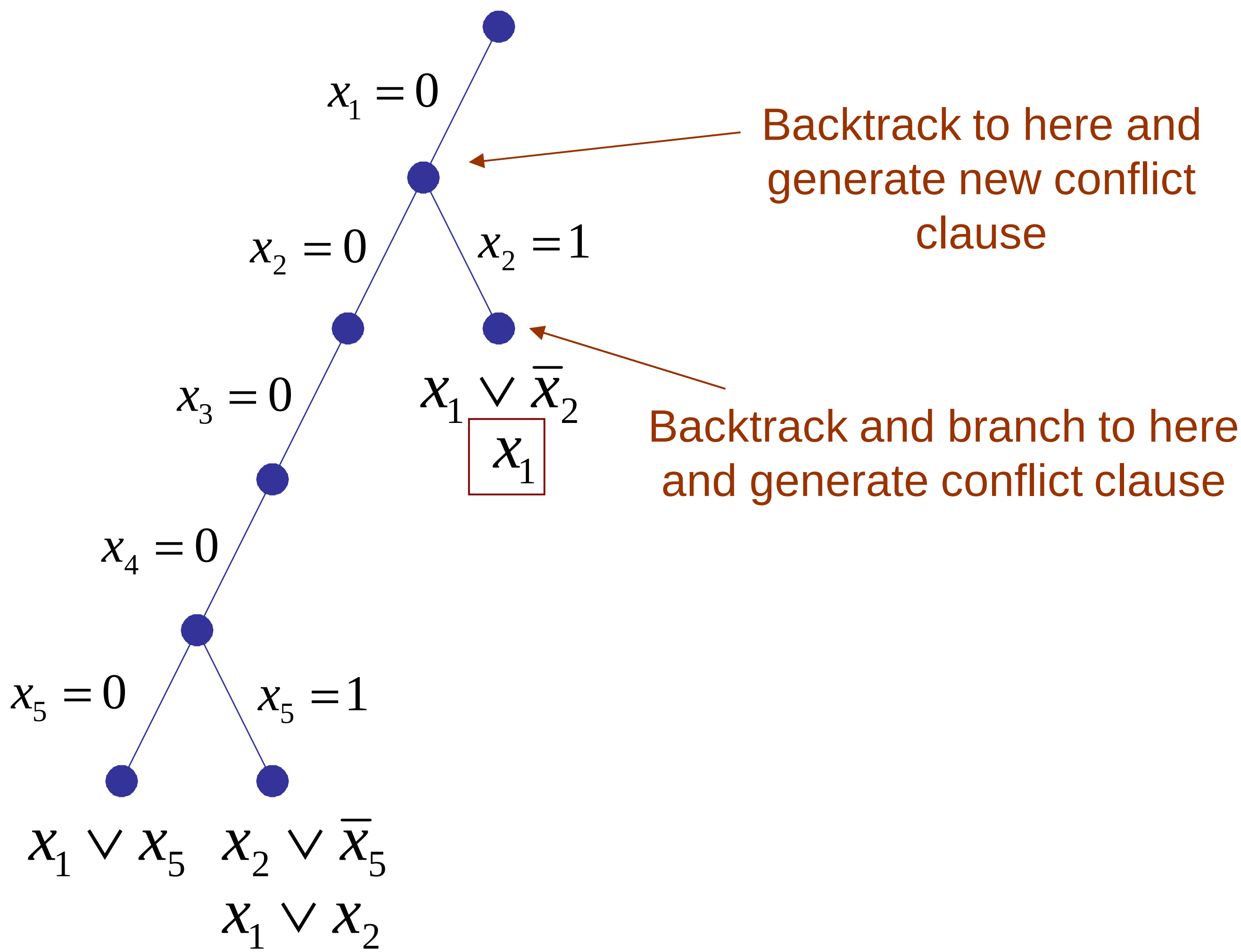
Generate conflict clause

Branching



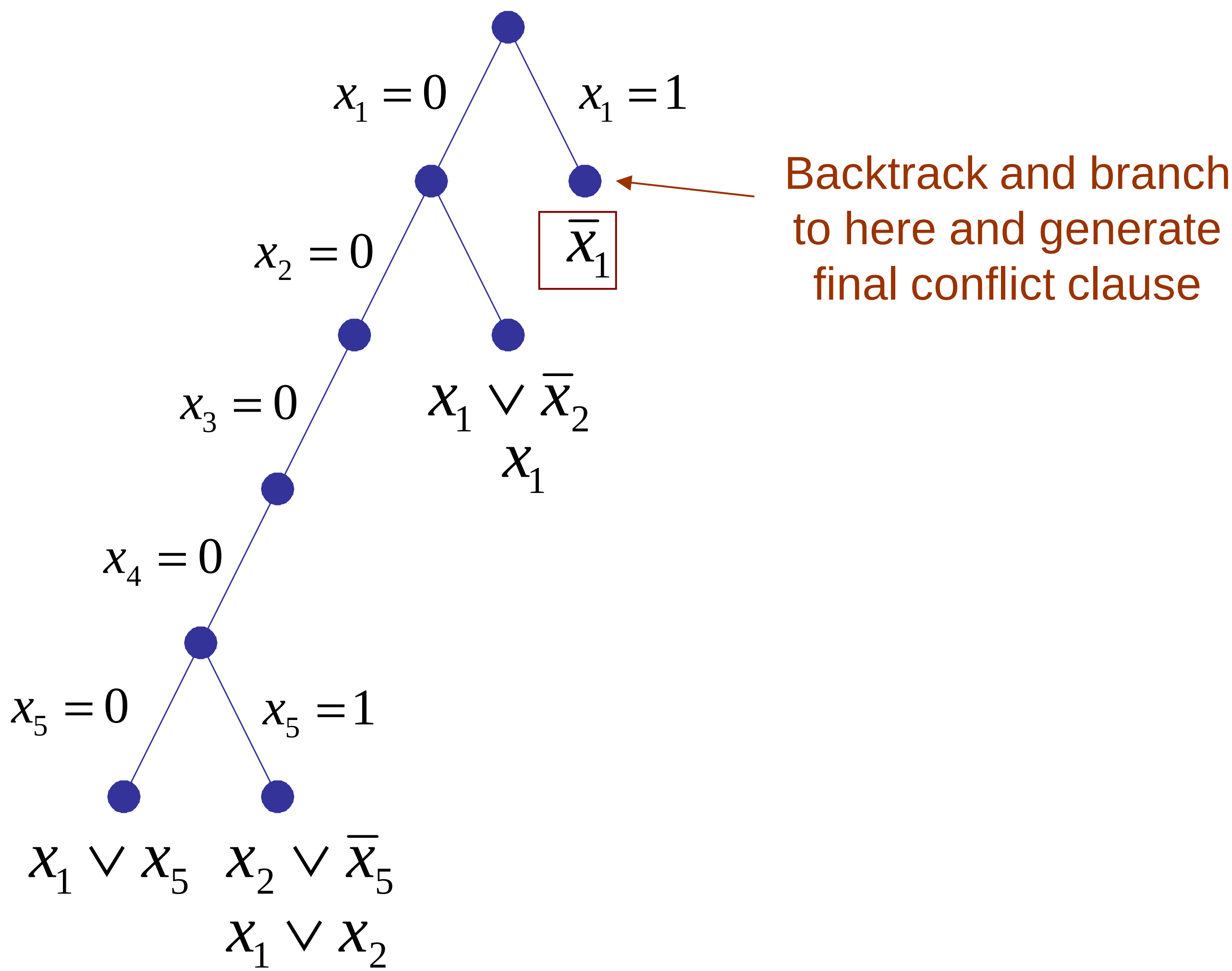
$x_1 = 0$
$x_2 = 0$
$x_2 = 1$
$x_3 = 0$
$x_4 = 0$
$x_5 = 0$
$x_5 = 1$
$\boxed{x_1 \vee \bar{x}_2}$
$x_1 \vee x_5$
$x_2 \vee \bar{x}_5$
$x_1 \vee x_2$
Backtrack and branch to here
and generate conflict clause

Branching



$x_1 = 0$
$x_2 = 0$
$x_2 = 1$
$x_3 = 0$
$x_4 = 0$
$x_5 = 0$
$x_5 = 1$
$x_1 \vee \bar{x}_2$
$\boxed{x_1}$
$x_1 \vee x_5$
$x_2 \vee \bar{x}_5$
$x_1 \vee x_2$
Backtrack to here and generate new conflict clause
Backtrack and branch to here and generate conflict clause

Branching



$x_1 = 0$
$x_1 = 1$
$\boxed{\bar{x}_1}$
Backtrack and branch to here and generate final conflict clause
$x_2 = 0$
$x_3 = 0$
$x_1 \vee \bar{x}_2$
x_1
$x_4 = 0$
$x_5 = 0$
$x_5 = 1$
$x_1 \vee x_5$
$x_2 \vee \bar{x}_5$
$x_1 \vee x_2$

To solve it by constraint-directed search:

- **Search**: generate problem restrictions.
  - Each **leaf node** of search tree is a problem restriction.

To solve it by constraint-directed search:

- **Search**: generate problem restrictions.
  - Each **leaf node** of search tree is a problem restriction.
- **Infer:** generate nogoods.
  - Process nogoods in current relaxation with **parallel resolution**.

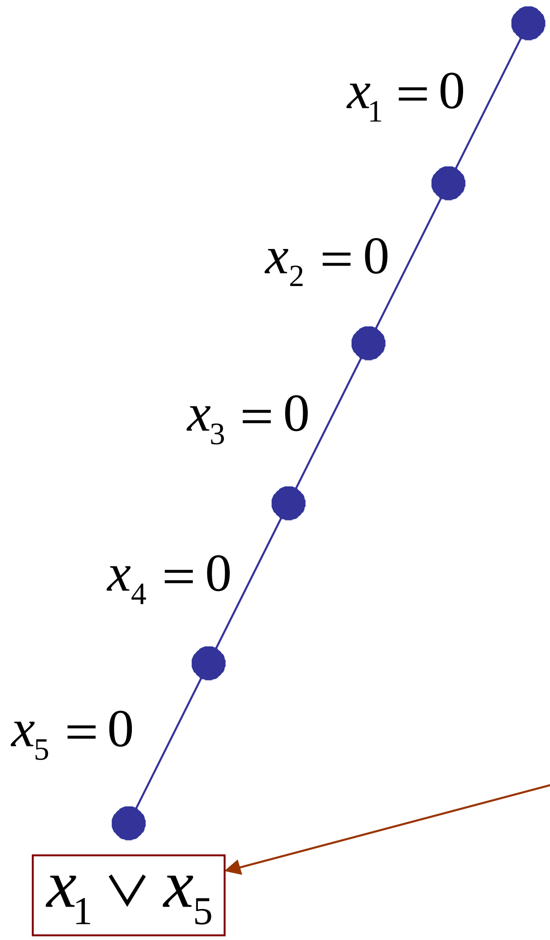
To solve it by constraint-directed search:

- **Search**: generate problem restrictions.
  - Each **leaf node** of search tree is a problem restriction.
- **Infer:** generate nogoods.
  - Process nogoods in current relaxation with **parallel resolution**.
- **Relax:** Relaxation R_k consists of current processed nogoods.

To solve it by constraint-directed search:

- **Search**: generate problem restrictions.
  - Each **leaf node** of search tree is a problem restriction.
- **Infer:** generate nogoods.
  - Process nogoods in current relaxation with **parallel resolution**.
- **Relax:** Relaxation R_k consists of current processed nogoods.
  - **Selection function**: Mimic chronological backtracking; apply unit clause rule.

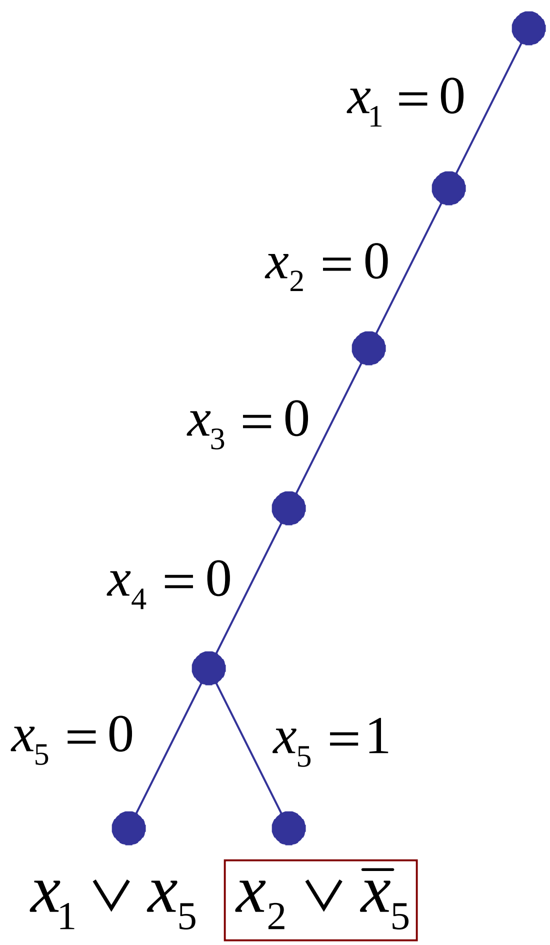
Constraint-Directed Search



k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$		

Conflict clause appears as nogood induced by solution of R_k.

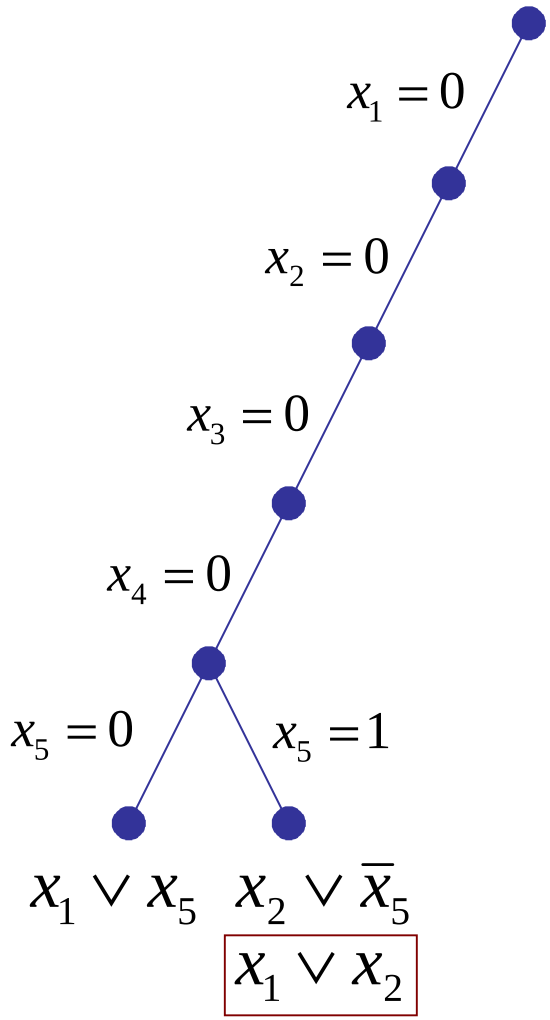
Constraint-Directed Search



Consists of **processed** nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$

Constraint-Directed Search



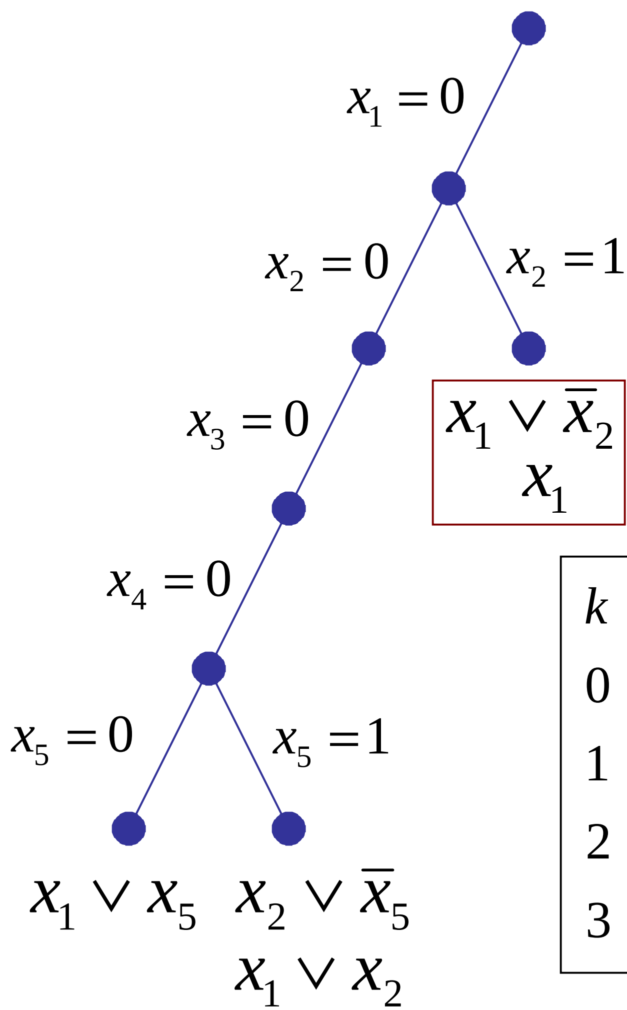
Consists of **processed** nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$
2	$\boxed{x_1 \vee x_2}$		

$x_1 \vee x_5$, $x_2 \vee \bar{x}_5$ **parallel-resolve** to yield $x_1 \vee x_2$

$x_1 \vee x_2$ **parallel-absorbs** $x_1 \vee x_5$, $x_2 \vee \bar{x}_5$

Constraint-Directed Search

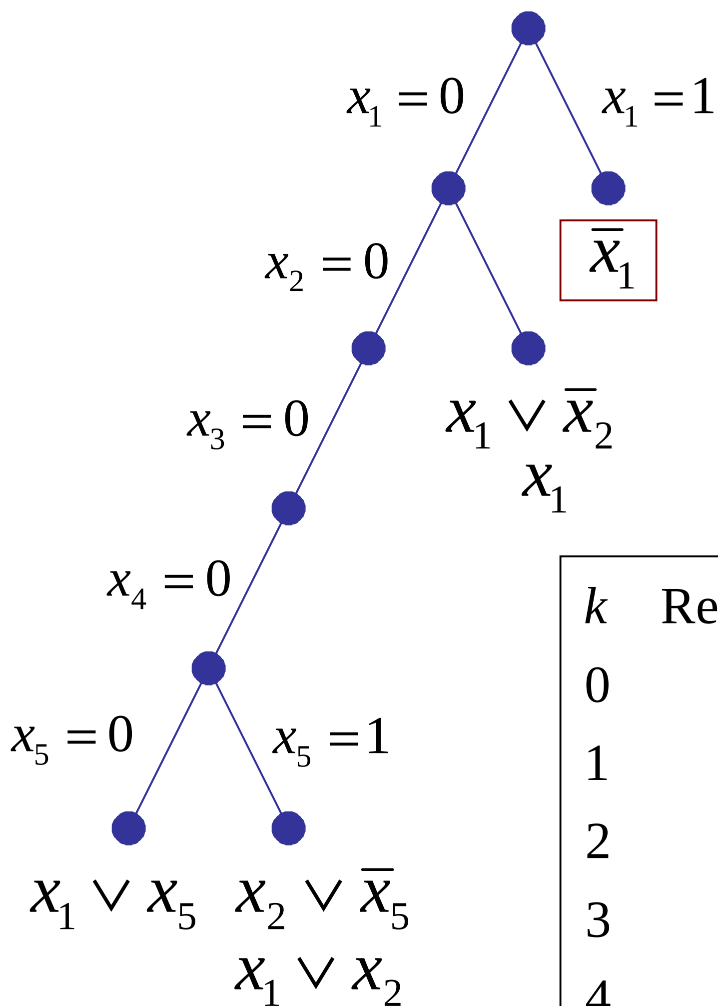


k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$
2	$x_1 \vee x_2$	$(0,1,\cdot,\cdot,\cdot,\cdot)$	$x_1 \vee \bar{x}_2$
3	$\boxed{x_1}$		

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$ parallel-resolve to yield x_1

Constraint-Directed Search



k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$
2	$x_1 \vee x_2$	$(0,1,\cdot,\cdot,\cdot,\cdot)$	$x_1 \vee \bar{x}_2$
3	x_1	$(1,\cdot,\cdot,\cdot,\cdot,\cdot)$	$\bar{x}_1$
4	$\boxed{\varnothing}$		

Search terminates

Constraint-Directed Search

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
DPL for SAT	Add conflict clauses	Processed conflict clauses	Unit clause rule + greedy solution of R_k	Parallel resolution & absorption
Partial order dynamic backtracking	**Add nogoods**	**Processed nogoods**	**Greedy, consistent with partial order**	**Parallel resolution & absorption**
Logic-based Benders	Subproblem defined by solution of master	Master problem (Benders cuts)	Optimal solution of master	Benders cuts (nogoods)

Constraint-Directed Search: Partial Order Dynamic Backtracking

Solve same problem as before

$$x_1 \vee x_5 \vee x_6$$
$$x_1 \vee x_5 \vee \bar{x}_6$$
$$x_2 \vee \bar{x}_5 \vee x_6$$
$$x_2 \vee \bar{x}_5 \vee \bar{x}_6$$
$$\bar{x}_1 \vee x_3 \vee x_4$$
$$\bar{x}_2 \vee x_3 \vee x_4$$
$$\bar{x}_1 \vee \bar{x}_3$$
$$\bar{x}_1 \vee \bar{x}_4$$
$$\bar{x}_2 \vee \bar{x}_3$$
$$\bar{x}_2 \vee \bar{x}_4$$

To solve it:

- **Search**: generate problem restrictions.

To solve it:

- **Search**: generate problem restrictions.
- **Infer:** generate nogoods.
  - Process nogoods in current relaxation with **parallel resolution**.

To solve it:

- **Search**: generate problem restrictions.
- **Infer:** generate nogoods.
  - Process nogoods in current relaxation with **parallel resolution**.
- **Relax:** Relaxation R_k consists of current processed nogoods.
  - **Selection function**: Solution of R_k must **conform to** current nogoods. Also apply unit clause rule.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee \boxed{x_1}$
1	$x_1 \vee x_5$		

Arbitrarily choose one variable to be **last**

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$\boxed{x_5} \vee \boxed{x_1}$
1	$x_1 \vee x_5$		

Other variables are **penultimate**

Arbitrarily choose one variable to be **last**

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	
2			

Since x_5 is **penultimate** in at least one nogood, it must **conform** to nogoods.

It must take value **opposite** its sign in the nogoods.

x_5 will have the **same sign** in all nogoods where it is penultimate.

This allows **more freedom** than chronological backtracking.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \boxed{\bar{x}_1}$
2			

Choice of **last** variable is arbitrary but must be consistent with **partial order** implied by previous choices.

Since x_5 is **penultimate** in at least one nogood, it must **conform** to nogoods.

It must take value **opposite** its sign in the nogoods.

x_5 will have the **same sign** in all nogoods where it is penultimate.

This allows **more freedom** than chronological backtracking.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \boxed{\bar{x}_1}$
2	$\boxed{x_5}$		

Choice of **last** variable is arbitrary but must be consistent with **partial order** implied by previous choices.

$$x_5 \vee x_1$$
$$x_5 \vee \bar{x}_1$$

Parallel-resolve to yield x_5

Since x_5 is **penultimate** in at least one nogood, it must **conform** to nogoods.

It must take value **opposite** its sign in the nogoods.

x_5 will have the **same sign** in all nogoods where it is penultimate.

This allows **more freedom** than chronological backtracking.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \bar{x}_1$
2	x_5	$(\cdot,0,\cdot,\cdot,1,\cdot)$	$\bar{x}_5 \vee x_2$
3	$\boxed{\left\{\begin{matrix} x_5 \\ \bar{x}_5 \vee x_2 \end{matrix}\right\}}$		

x_5 does not parallel-resolve with $\bar{x}_5 \vee x_2$

because x_5 is not last in both clauses

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \bar{x}_1$
2	x_5	$(\cdot,0,\cdot,\cdot,1,\cdot)$	$\bar{x}_5 \vee x_2$
3	$\left\{ \begin{matrix} x_5 \\ \bar{x}_5 \vee x_2 \end{matrix} \right\}$	$(\cdot,1,\cdot,\cdot,1,\cdot)$	$\bar{x}_2$
4	$\varnothing$		

Search terminates

Must conform

Constraint-Directed Search

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
DPL for SAT	Add conflict clauses	Processed conflict clauses	Unit clause rule + greedy solution of R_k	Parallel resolution & absorption
Partial order dynamic backtracking	Add nogoods	Processed nogoods	Greedy, consistent with partial order	Parallel resolution & absorption
Logic-based Benders	**Subproblem defined by solution of master**	**Master problem (Benders cuts)**	**Optimal solution of master**	**Benders cuts (nogoods)**

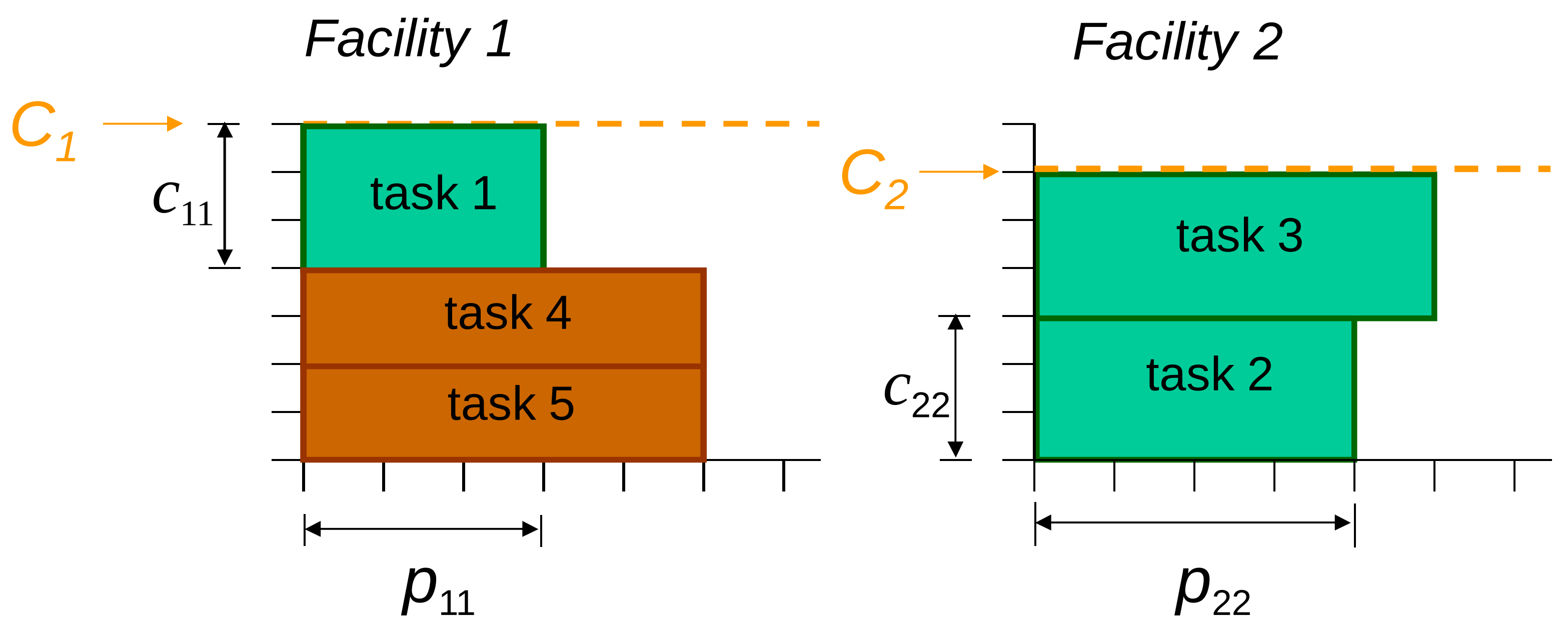
Constraint-Directed Search: Logic-Based Benders Decomposition

Planning and scheduling problem:

- Allocate tasks to facilities.
- Schedule tasks assigned to each facility.
  - Subject to deadlines.
  - Facilities may run at different speeds and incur different costs.
- Cumulative scheduling
  - Several tasks may run simultaneously on a facility.
  - But total resource consumption must never exceed limit.

Planning and Scheduling

p_{ij} = processing time of task j on facility i
c_{ij} = resource consumption of task j on facility i
C_i = resources available on facility i



Total resource consumption $\leq C_i$ at all times.

Planning and Scheduling

y_j = facility assigned to task j

$$\min \sum_j c_{y_j j}$$

$$\text{cumulative}\begin{pmatrix} (t_j \mid y_j = i) \\ (p_{ij} \mid y_j = i) \\ (c_{ij} \mid y_j = i) \\ C_i \end{pmatrix}, \quad \text{all } i$$

$$0 \le t_j \le d_j - p_{y_j j}, \quad \text{all } j$$

Planning and Scheduling

Cost of processing task *j* on facility *i*

y_j = facility assigned to task *j*

$$\min \sum_j c_{y_j j}$$

$$\text{cumulative}\begin{pmatrix} (t_j \mid y_j = i) \\ (p_{ij} \mid y_j = i) \\ (c_{ij} \mid y_j = i) \\ C_i \end{pmatrix}, \quad \text{all } i$$

$$0 \le t_j \le d_j - p_{y_j j}, \quad \text{all } j$$

Planning and Scheduling

Cost of processing task *j* on facility *i*

y_j = facility assigned to task *j*

start times of tasks assigned to facility *i*

$$\min \sum_j c_{y_j j}$$

$$\text{cumulative}\begin{pmatrix} (t_j \mid y_j = i) \\ (p_{ij} \mid y_j = i) \\ (c_{ij} \mid y_j = i) \\ C_i \end{pmatrix}, \quad \text{all } i$$

$$0 \le t_j \le d_j - p_{y_j j}, \quad \text{all } j$$

Observe resource limit on each facility

Planning and Scheduling

Cost of processing task *j* on facility *i*

y_j = facility assigned to task *j*

$$\min \sum_j c_{y_j j}$$

start times of tasks assigned to facility *i*

$$\text{cumulative}\begin{pmatrix} (t_j \mid y_j = i) \\ (p_{ij} \mid y_j = i) \\ (c_{ij} \mid y_j = i) \\ C_i \end{pmatrix}, \quad \text{all } i$$

Observe resource limit on each facility

$$0 \le t_j \le d_j - p_{y_j j}, \quad \text{all } j$$

Observe time windows

To solve it:

- **Search**: enumerate assignments of tasks to facilities.
  - Each assignment defines a problem restriction (scheduling **subproblem**).

To solve it:

- **Search**: enumerate assignments of tasks to facilities.
  - Each assignment defines a problem restriction (scheduling **subproblem**).
- **Infer:** generate nogoods.
  - Nogoods (**Benders cuts**) exclude assignments that have no feasible schedule.

To solve it:

- **Search**: enumerate assignments of tasks to facilities.
  - Each assignment defines a problem restriction (scheduling **subproblem**).
- **Infer:** generate nogoods.
  - Nogoods (**Benders cuts**) exclude assignments that have no feasible schedule.
- **Relax:** Relaxation R_k consists of Benders cuts generated so far .

To solve it:

- **Search**: enumerate assignments of tasks to facilities.
  - Each assignment defines a problem restriction (scheduling **subproblem**).
- **Infer:** generate nogoods.
  - Nogoods (**Benders cuts**) exclude assignments that have no feasible schedule.
- **Relax:** Relaxation R_k consists of Benders cuts generated so far .
  - **Selection function**: Any optimal solution of R_k (= **master problem**).

Restrict (Subproblem)

For a given assignment of tasks to facilities,
find a feasible schedule

Solve by **constraint programming**

Given assignment

$$\left\{ \begin{array}{l} \text{cumulative}\begin{pmatrix} (t_j \mid \bar{y}_j = i) \\ (p_{ij} \mid \bar{y}_j = i) \\ (c_{ij} \mid \bar{y}_j = i) \\ C_i \end{pmatrix} \\ 0 \le t_j \le d_j \end{array} \right\}, \quad \text{all } i$$

Infer (Benders cut = nogood)

$$\left\{ \begin{array}{l} \text{cumulative}\begin{pmatrix} (t_j \mid \bar{y}_j = i) \\ (p_{ij} \mid \bar{y}_j = i) \\ (c_{ij} \mid \bar{y}_j = i) \\ C_i \end{pmatrix} \\ 0 \le t_j \le d_j \end{array} \right\}, \quad \text{all } i$$

Let J_{ih} = {tasks assigned to facility i in iteration h}.

If there is no feasible schedule, create Benders cut:

$$y_j \neq i \text{ for some } j \in J_{hi}$$

Relax (Master problem)

Solve by MILP.

$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\sum_i x_{ij} = 1, \quad \text{all } j$$

$$\sum_{j \in J_{hi}} (1 - x_{ij}) \geq 1, \quad \text{all } h, i$$

$$x_{ij} \in \{0,1\}$$

Let $x_{ij} = 1$ when $y_j = i$

Task j is assigned to one facility

Benders cuts: $x_{ij} = 0$ for some $j \in J_{hi}$

Stop when subproblem is feasible (original problem is feasible)…

…**or** when master problem is infeasible (original problem is infeasible).

General Framework

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Branching CP, MILP, global optimization	Node of search tree	LP, NLP, domains	Any optimal (feasible) solution of R_k	Domain filtering, cutting planes
Constraint directed DPL, dynamic backtracking, Benders	Add nogoods generated so far	Processed nogoods generated so far	Solution that results in easy R_k	Nogood generation
Heuristics **Local search, GRASP**	**Neighborhood of current solution**	**Research topic**	**Random, best, etc.**	**?**

Heuristic Methods

Solution Method	*Restriction* P_k	*Relaxation* R_k	*Selection Function* $s(R_k)$	*Inference*
Simulated annealing	Neighborhood of current solution	P_k	Random solution from nbhd	None
Tabu search	Neighborhood minus tabu list	P_k	Best solution in nbhd	Items in tabu list
GRASP	Neighborhood of partial solution	Problem specific	Solve R_k only at leaf node	None

Heuristic Methods

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Simulated annealing	**Neighborhood of current solution**	$\boldsymbol{P_k}$	**Random solution from nbhd**	**None**
Tabu search	Neighborhood minus tabu list	P_k	Best solution in nbhd	Items in tabu list
GRASP	Neighborhood of partial solution	Problem specific	Solve R_k only at leaf node	None

Heuristics:
Simulated Annealing

To solve it:

- **Search**: enumerate neighborhoods (restrictions).

Heuristics:
Simulated Annealing

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** none

Heuristics: Simulated Annealing

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** none
- **Relax:** Same as restriction.

Heuristics:
Simulated Annealing

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** none
- **Relax:** Same as restriction.
  - **Selection function**: Random solution x in neighborhood.

Heuristics: Simulated Annealing

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** none
- **Relax:** Same as restriction.
  - **Selection function**: Random solution x in neighborhood.
  - Next restriction:
    - neighborhood of x if x is better than previous solution;

Heuristics:
Simulated Annealing

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** none
- **Relax:** Same as restriction.
  - **Selection function**: Random solution x in neighborhood.
  - Next restriction:
    - neighborhood of x if x is better than previous solution;
    - otherwise neighborhood of x with probability p,
      current neighborhood with probability $1 - p$.

Heuristic Methods

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Simulated annealing	Neighborhood of current solution	P_k	Random solution from nbhd	None
Tabu search	**Neighborhood minus tabu list**	$\boldsymbol{P_k}$	**Best solution in nbhd**	**Items in tabu list**
GRASP	Neighborhood of partial solution	Problem specific	Solve R_k only at leaf node	None

Heuristics:
Tabu Search

To solve it:

- **Search**: enumerate neighborhoods (restrictions).

Heuristics:
Tabu Search

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** item in tabu list (functions as nogood)

Heuristics:
Tabu Search

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** item in tabu list (functions as nogood)
- **Relax:** Same as restriction.

Heuristics:
Tabu Search

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** item in tabu list (functions as nogood)
- **Relax:** Same as restriction.
  - **Selection function**: Best solution x in neighborhood that is consistent with tabu list.

Heuristics:
Tabu Search

To solve it:

- **Search**: enumerate neighborhoods (restrictions).
- **Infer:** item in tabu list (functions as nogood)
- **Relax:** Same as restriction.
  - **Selection function**: Best solution x in neighborhood that is consistent with tabu list.
  - Next restriction: neighborhood of x.

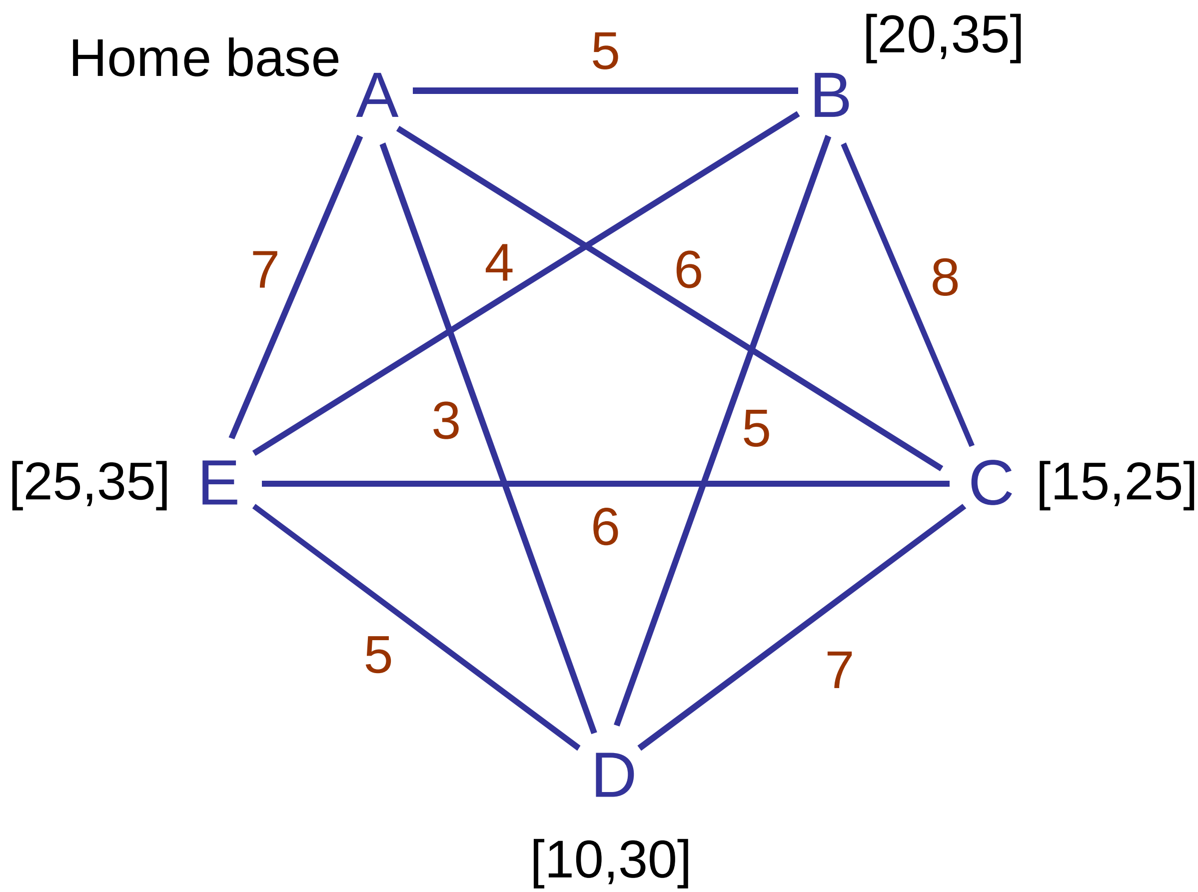
Heuristic Methods

Solution Method	*Restriction P_k*	*Relaxation R_k*	*Selection Function $s(R_k)$*	*Inference*
Simulated annealing	Neighborhood of current solution	P_k	Random solution from nbhd	None
Tabu search	Neighborhood minus tabu list	P_k	Best solution in nbhd	Items in tabu list
GRASP	**Neighborhood of partial solution**	**Problem specific**	**Solve R_k only at leaf node**	**None**

Heuristics:
Generalized GRASP

Greedy Randomized Adaptive Search Procedure

TSP with Time Windows



Heuristics: Generalized GRASP

To solve it:

- **Search**: enumerate neighborhoods of partial solutions
  - Neighboring solution is created by selecting customer to visit next.

Heuristics: Generalized GRASP

To solve it:

- **Search**: enumerate neighborhoods of partial solutions
  - Neighboring solution is created by selecting customer to visit next.
- **Infer:** none

Heuristics:
Generalized GRASP

To solve it:

- **Search**: enumerate neighborhoods of partial solutions
  - Neighboring solution is created by selecting customer to visit next.
- **Infer:** none
- **Relax:** Same as restriction.

Heuristics:
Generalized GRASP

To solve it:

- **Search**: enumerate neighborhoods of partial solutions
  - Neighboring solution is created by selecting customer to visit next.
- **Infer:** none
- **Relax:** Same as restriction.
  - **Selection function**:
    - **Greedy phase**: Select next customer to visit in greedy fashion.

Heuristics: Generalized GRASP

To solve it:

- **Search**: enumerate neighborhoods of partial solutions
  - Neighboring solution is created by selecting customer to visit next.
- **Infer:** none
- **Relax:** Same as restriction.
  - **Selection function**:
    - **Greedy phase**: Select next customer to visit in greedy fashion.
    - **Local search phase**: Randomly backtrack and select next customer in random fashion.

Generalized GRASP

Sequence
of customers
visited

A · · · · A

Basically,

GRASP =
greedy solution +
local search

Begin with greedy
assignments that
can be viewed as
creating
“branches”

Generalized GRASP

Greedy phase

A · · · · A

AD · · · A

Visit customer than can be served earliest from A

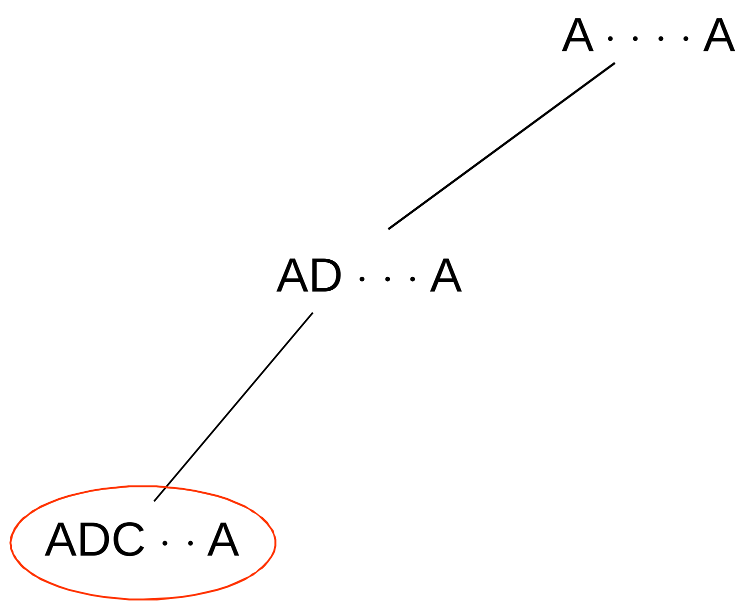
Basically,

GRASP = greedy solution + local search

Begin with greedy assignments that can be viewed as creating "branches"

Generalized GRASP

Greedy
phase



Next, visit
customer than can
be served earliest
from D

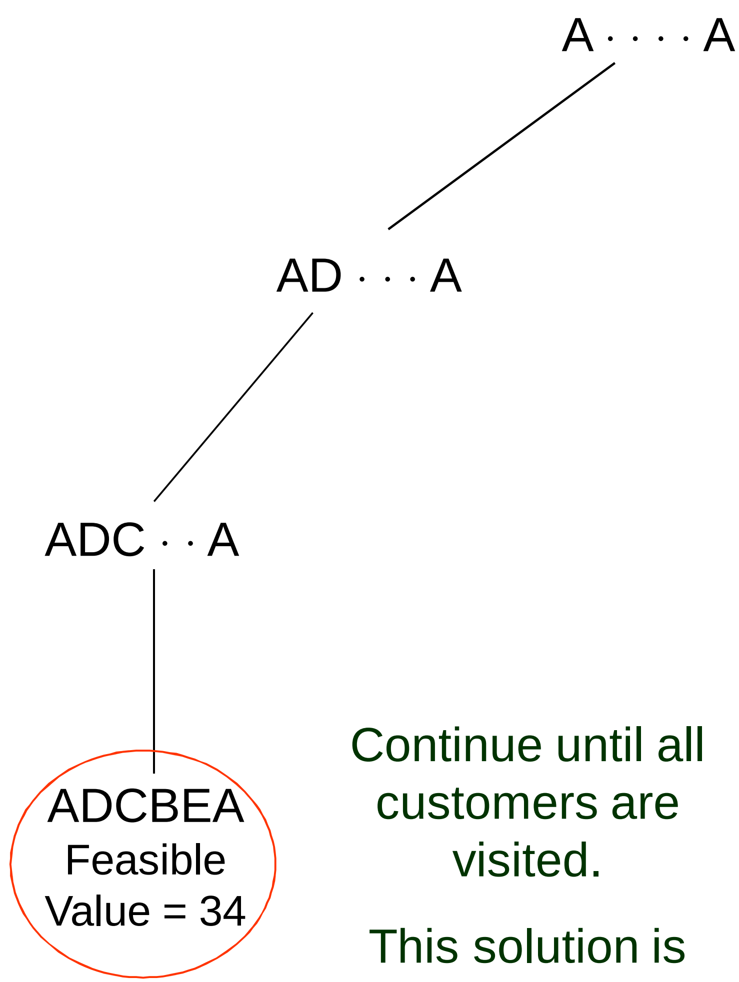
Basically,

GRASP =
greedy solution +
local search

Begin with greedy
assignments that
can be viewed as
creating
“branches”

Generalized GRASP

Greedy phase



Continue until all customers are visited.

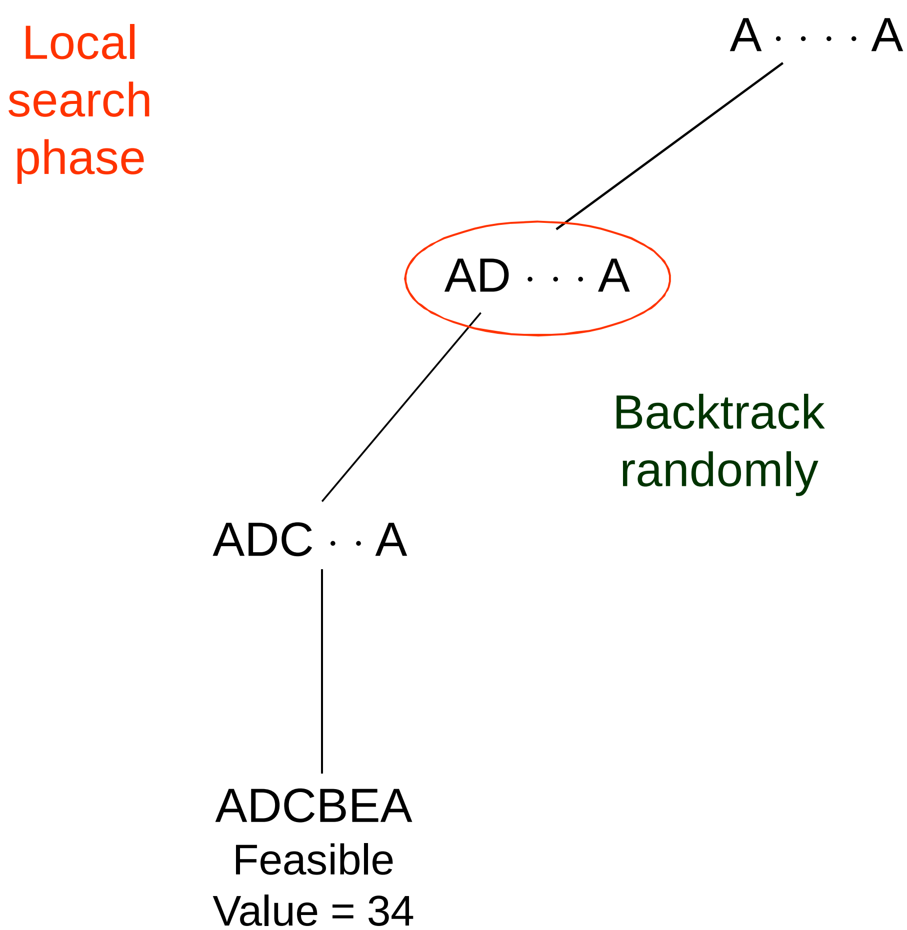
This solution is feasible. Save it.

Basically,

GRASP = greedy solution + local search

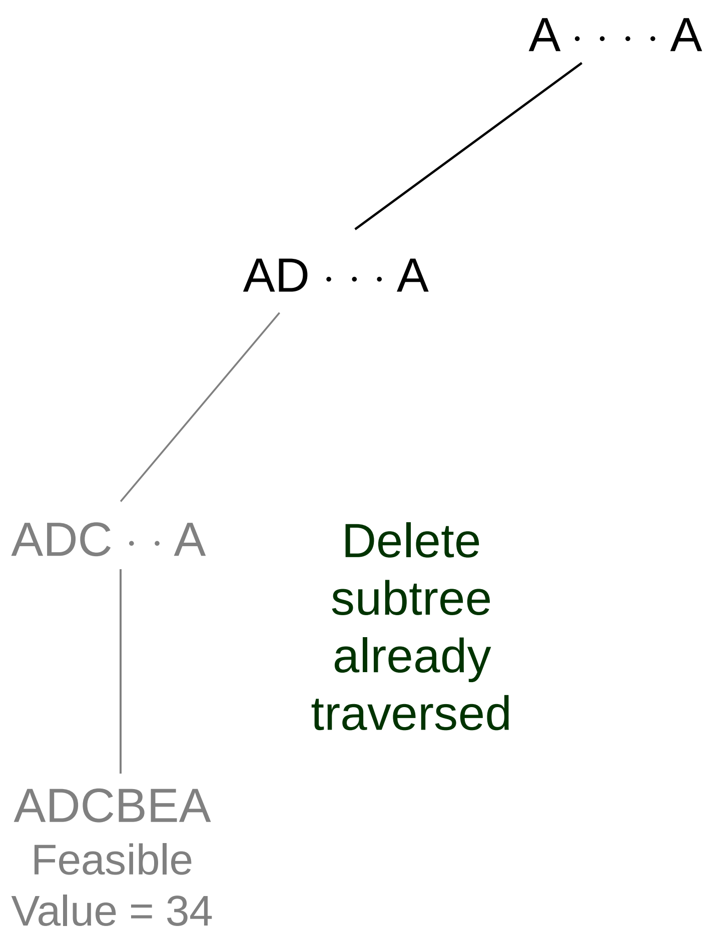
Begin with greedy assignments that can be viewed as creating “branches”

Generalized GRASP



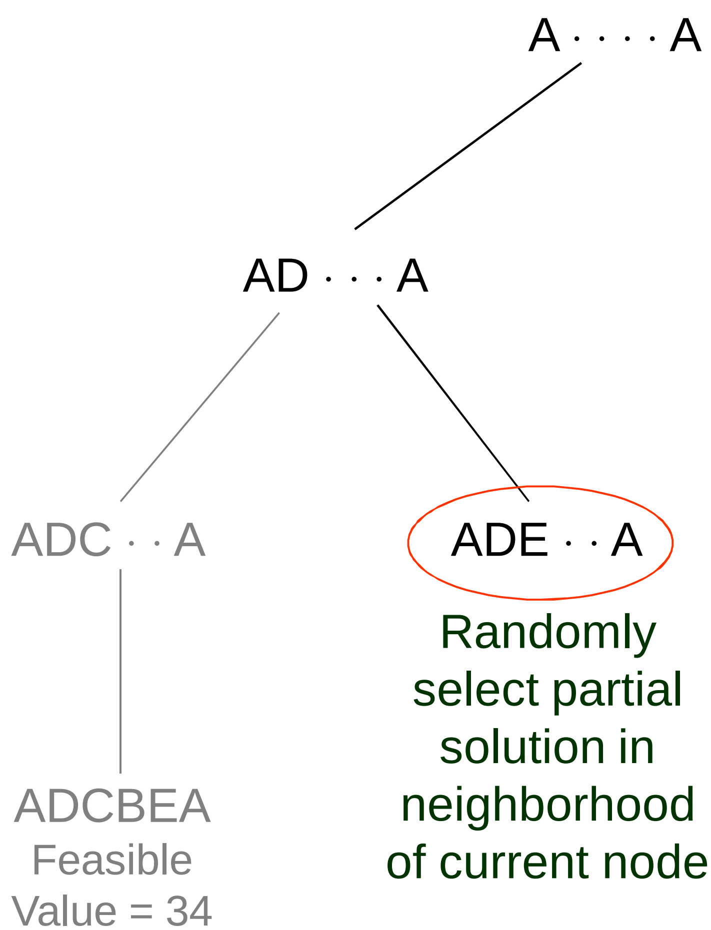
Generalized GRASP

Local
search
phase

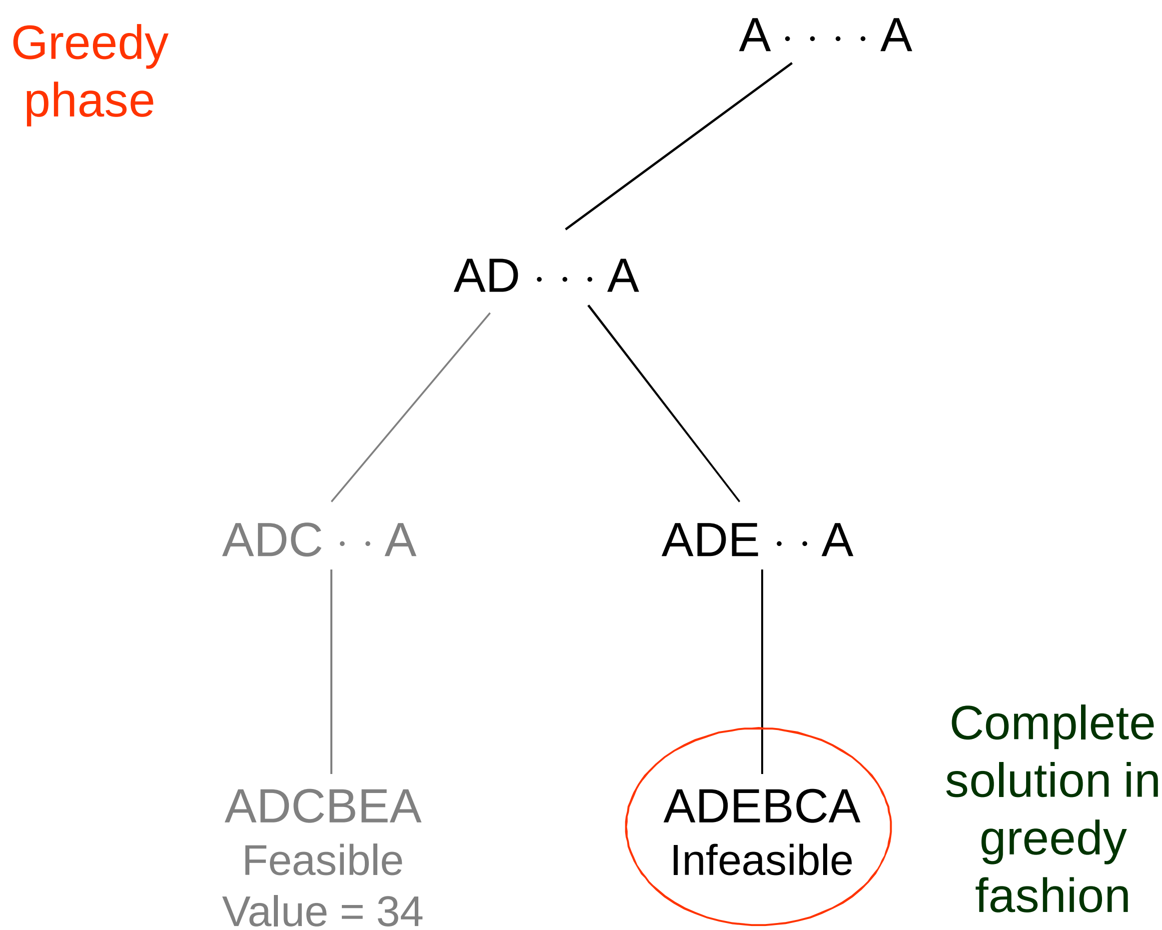


Generalized GRASP

Local search phase

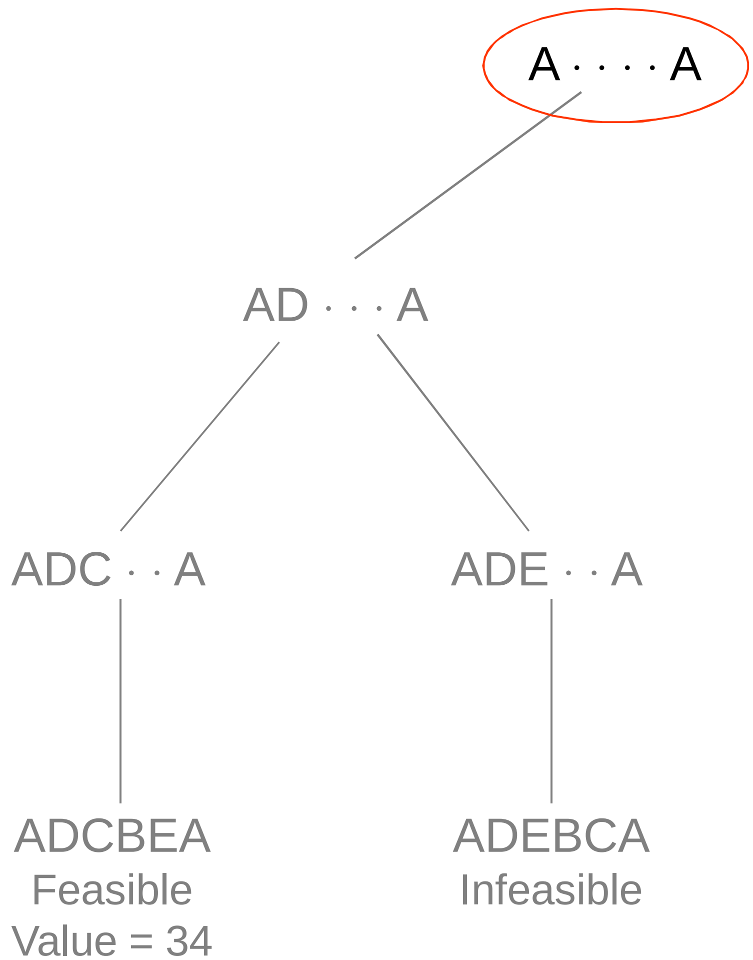


Generalized GRASP



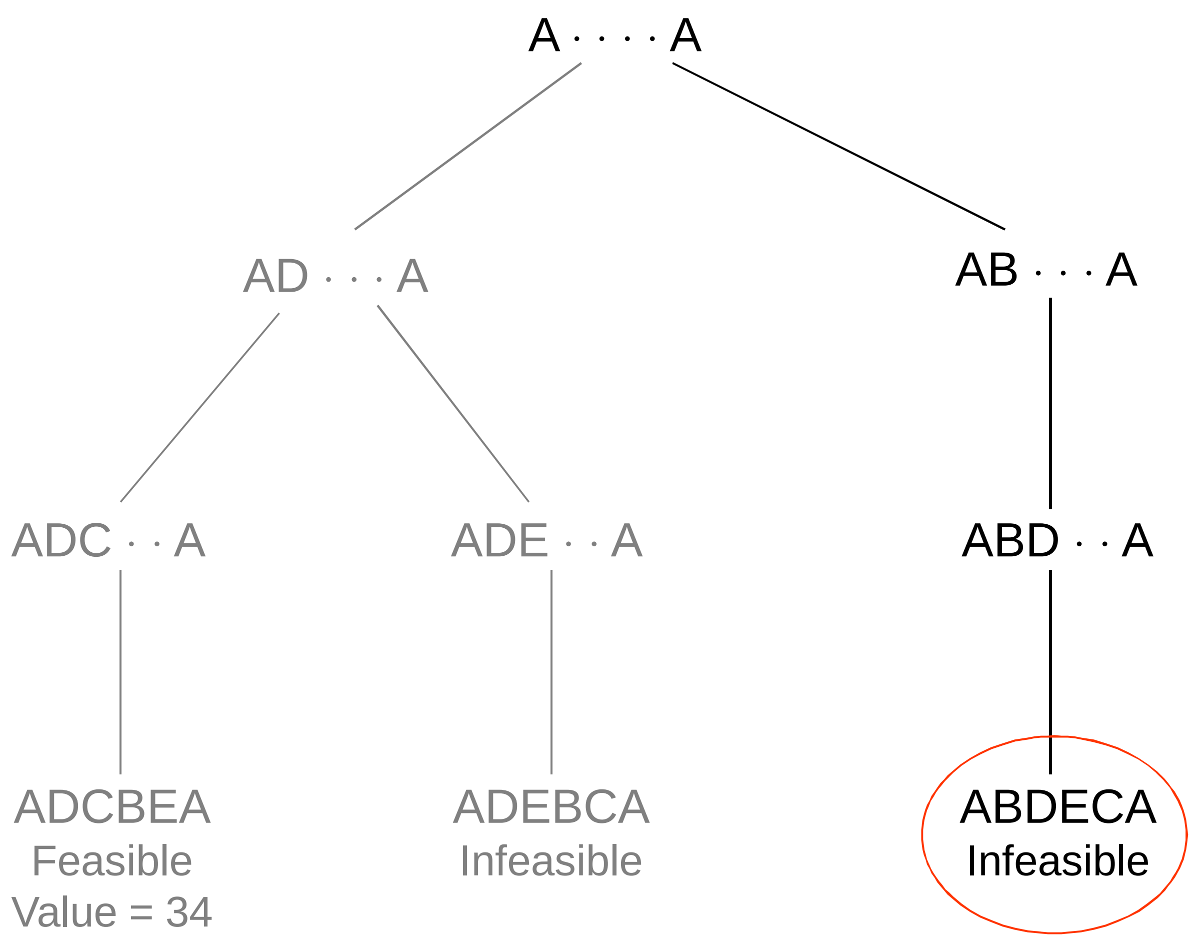
Generalized GRASP

Local
search
phase



Randomly
backtrack

Generalized GRASP



Continue in similar fashion

Local search
is generalized GRASP
in which the “branching” tree
has 2 levels

To add a relaxation to generalized GRASP:

Suppose that customers $x_0, x_1, \ldots, x_k$ have been visited so far.

Let t_{ij} = travel time from customer i to j.

Then total travel time of completed route is bounded below by

$$\boxed{T} + \sum_{j \notin \{x_0,\ldots,x_k\}} \boxed{\min\left\{ t_{x_k j},\ \min_{i \notin \{j, x_0,\ldots,x_k\}} \{t_{ij}\} \right\}} + \boxed{\min_{j \notin \{x_0,\ldots,x_k\}} \{t_{j0}\}}$$

Earliest time vehicle can leave customer *k*

Min time from customer *j*'s predecessor to *j*

Min time from last customer back to home

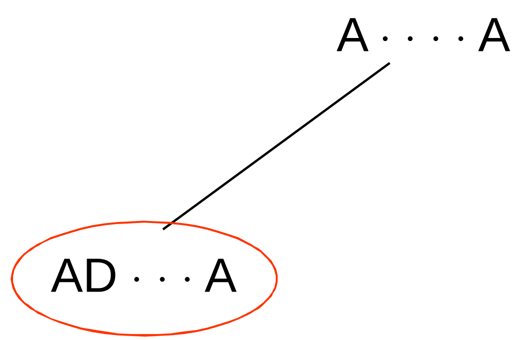
Heuristics: Generalized GRASP with relaxation

To solve it:

- **Search**: enumerate neighborhoods of partial solutions
  - Neighboring solution is created by selecting customer to visit next.
- **Infer:** None.
- **Relax:** As just described.
  - **Selection function**:
    - **Greedy phase**: Select next customer to visit in greedy fashion.
    - **Local search phase**: Randomly backtrack and select next customer in random fashion.

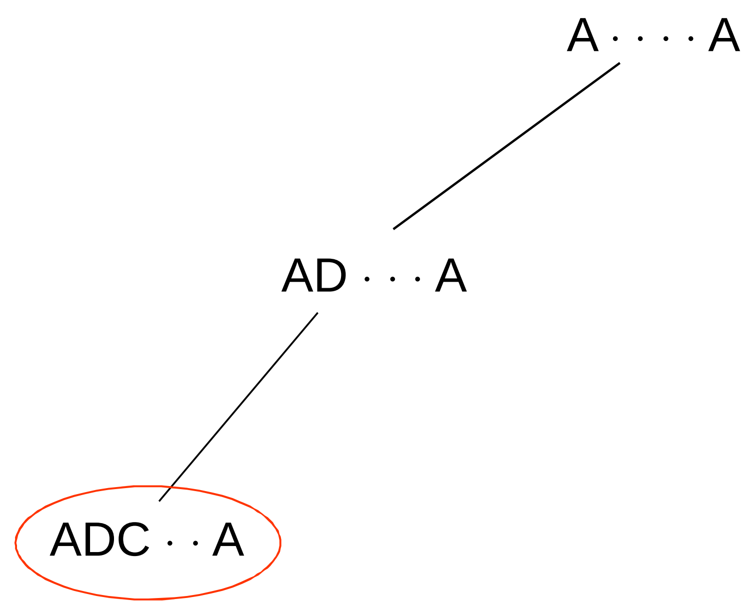
Generalized GRASP with relaxation

Greedy phase



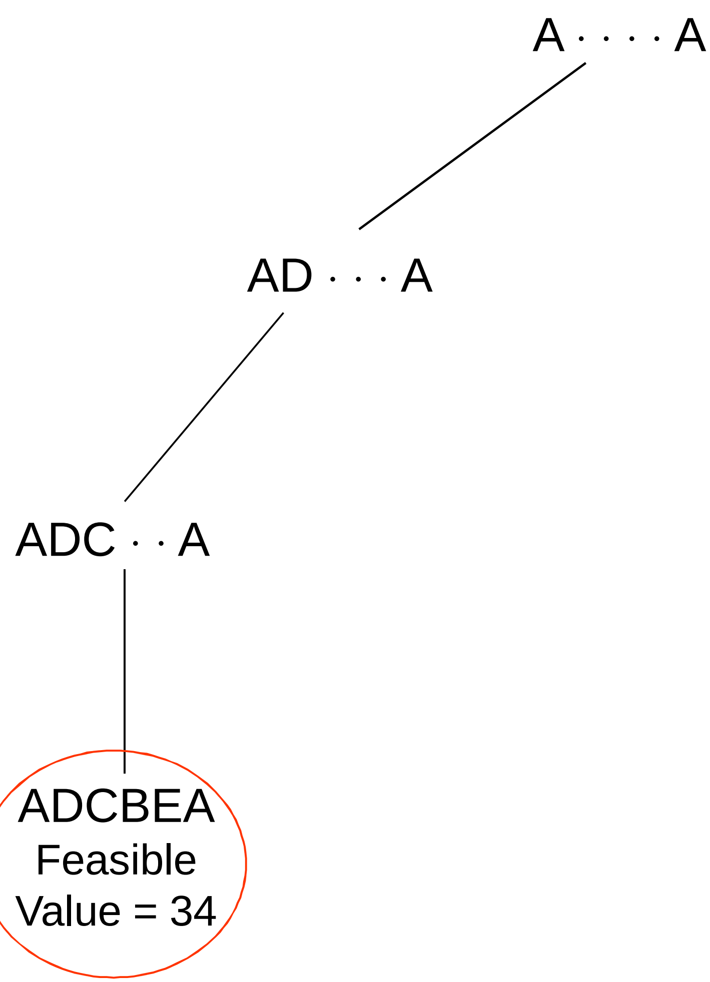
Generalized GRASP

Greedy
phase

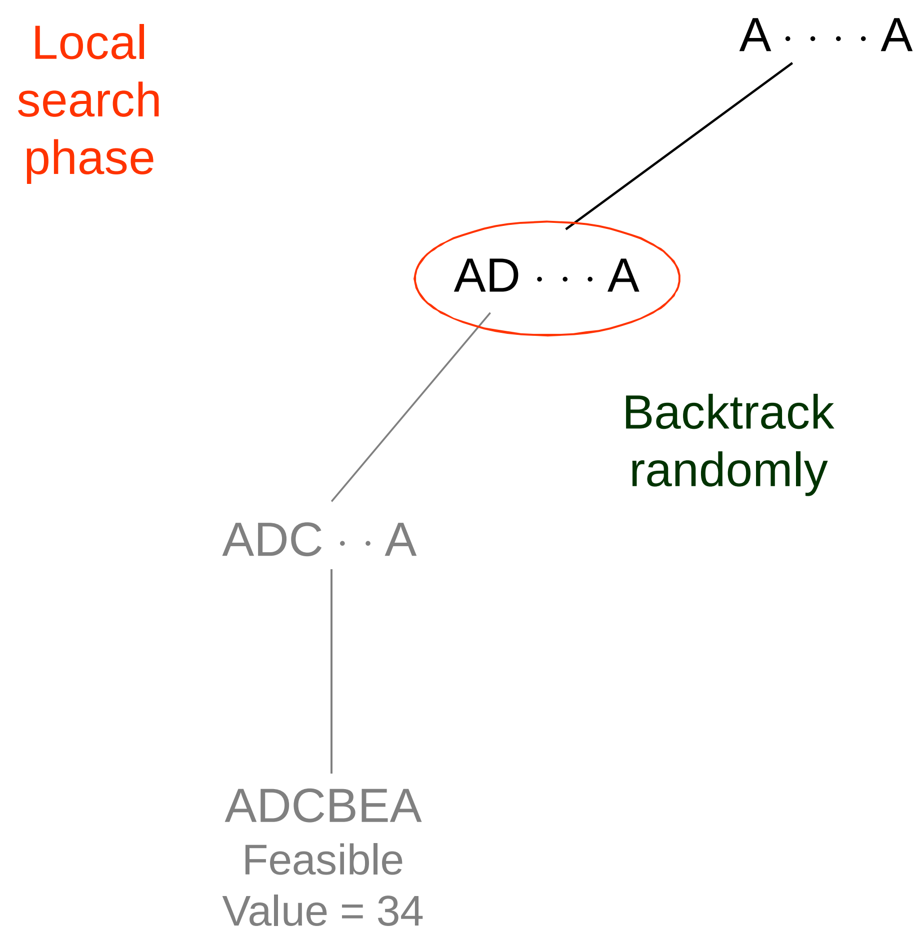


Generalized GRASP

Greedy phase

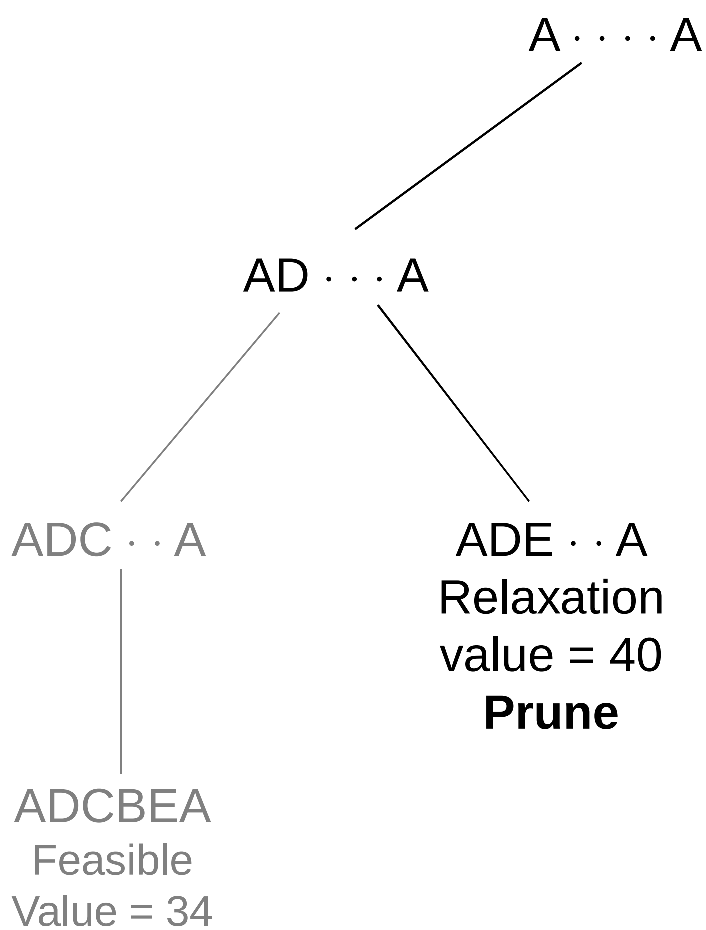


Generalized GRASP

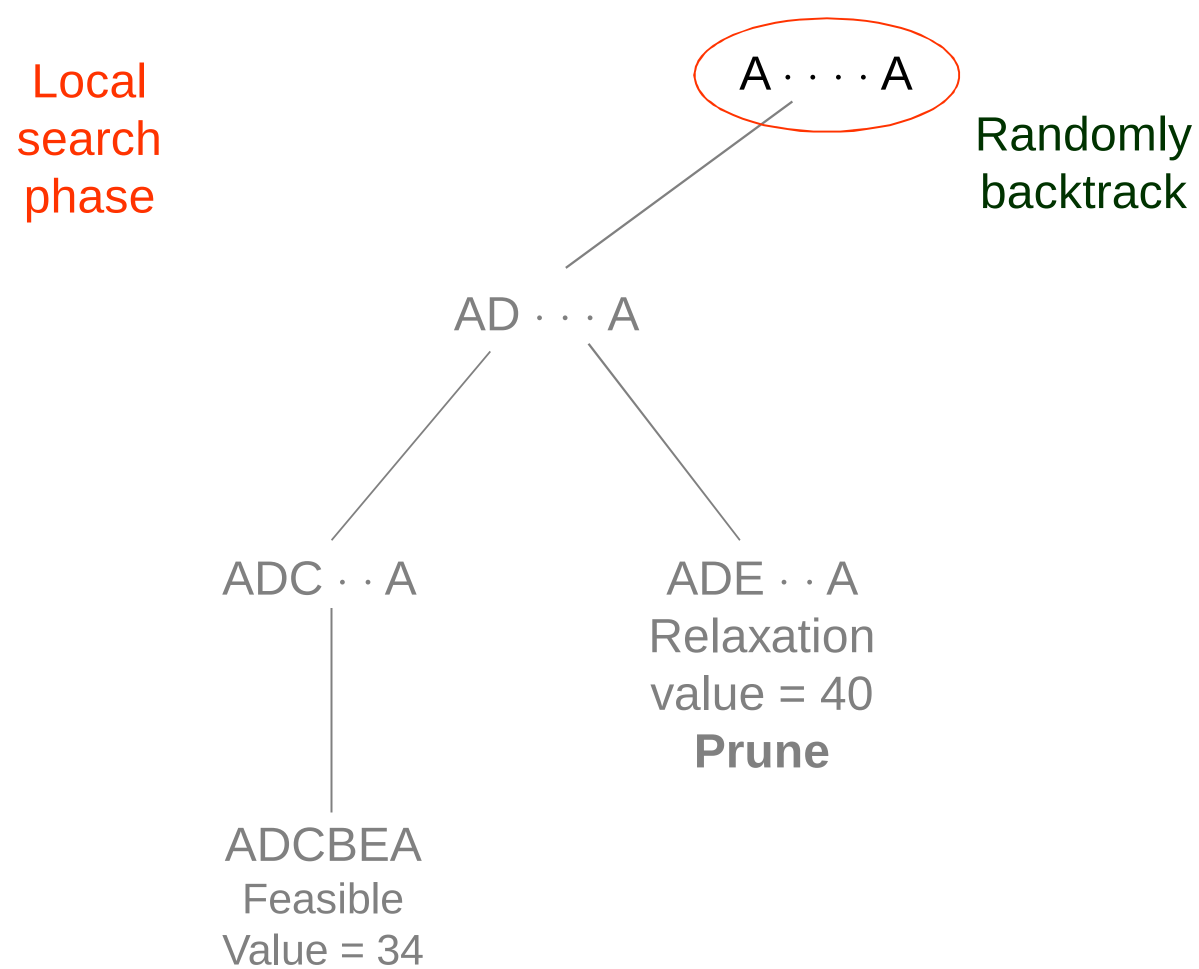


Generalized GRASP

Local search phase

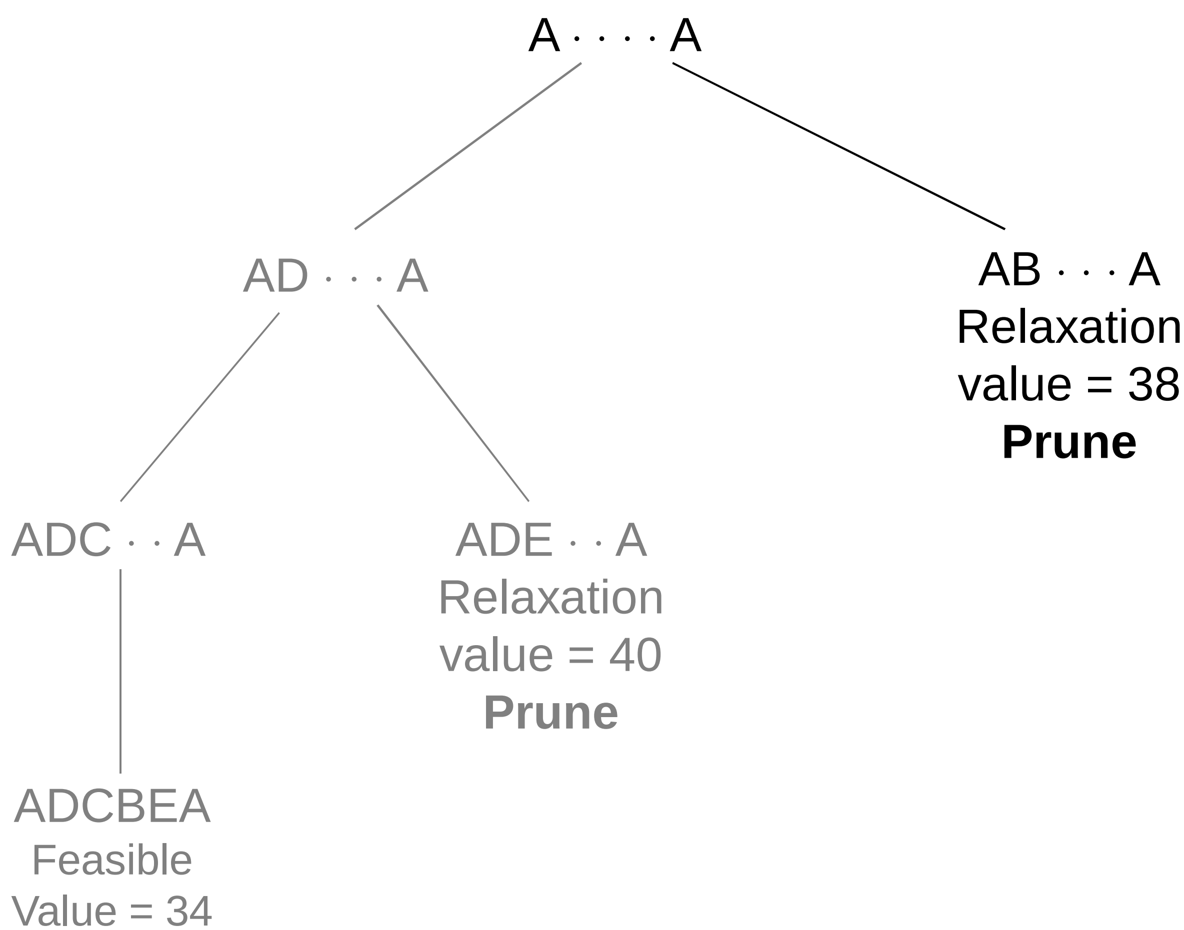


Generalized GRASP



Generalized GRASP

Local
search
phase



Product endorsement:

SIMPL
is a partial implementation of this approach
(CP-AI-OR 2004)

We expect to post on the web this fall.