A Primal-Dual Framework for Combinatorial Problem Solving

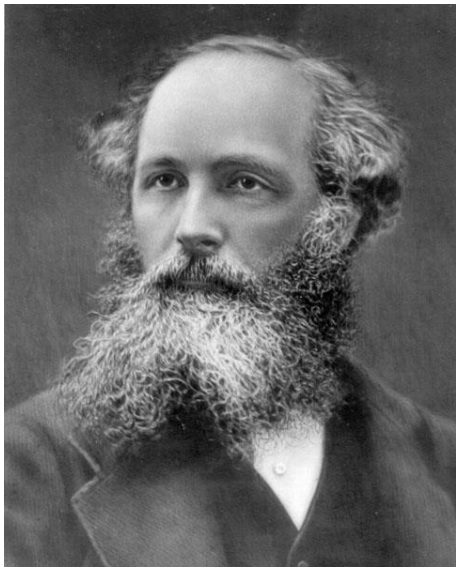
John Hooker

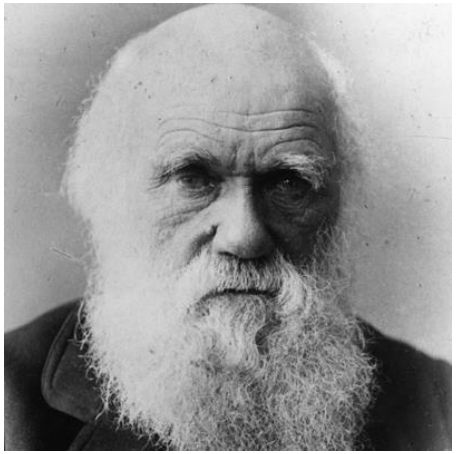
Carnegie Mellon University

Workshop on Seeking Feasibility in Combinatorial Problems
CPAIOR 2013

A Word about Unifying Theories

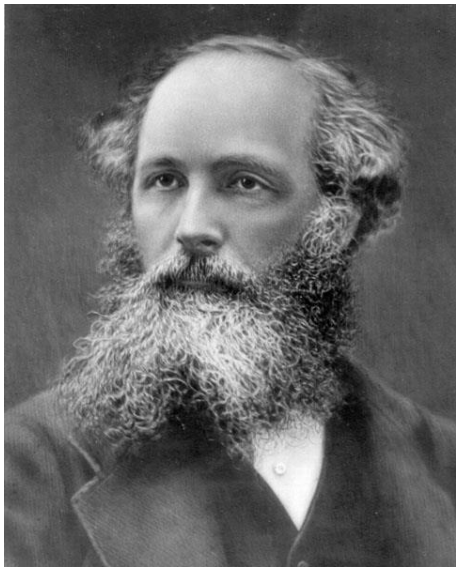
- **Synthesis** – 19th century
  - Maxwell's equations
  - Evolution by natural selection
- **Analysis** – 20th century
  - Except in reductive schemes (DNA, TOE in physics)
  - Abstraction is **not** synthesis (Bourbaki school)
  - AI (General Problem Solver) is not synthesis.
- **Return** to synthesis in 21st century?

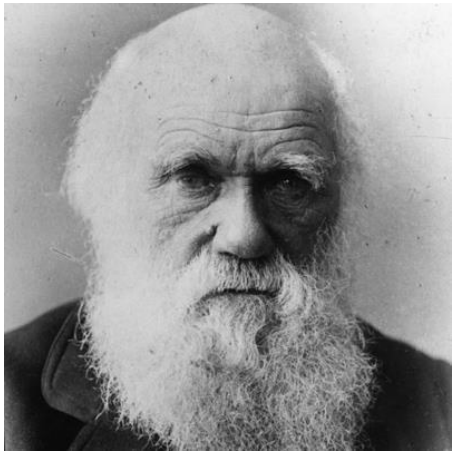




A Word about Unifying Theories

- **Primal/dual** – a unifying principle for combinatorial problem solving.



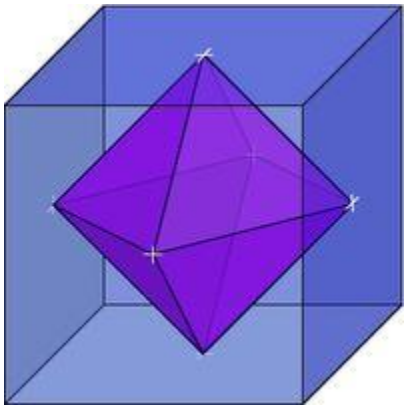


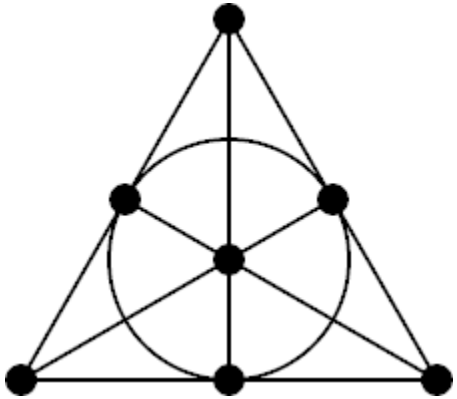
Primal/Dual Framework

- **Primal problem**
  - Find good feasible solutions
- **Dual problem**
  - Prove infeasibility (or optimality)
- **Primal/dual method**
  - Solve primal and dual simultaneously
  - Exchange information
  - Accelerates solution of both primal and dual problem.

Duality

- Most mathematical duals are symmetric
  - Dual of dual = original
- Polar duality of polytopes
  - Facets / vertices
- Duality in projective geometry
  - Points / lines
- Duality of vector spaces
  - Row rank / column rank





Fano plane

Optimization Duals

- Generally not symmetric:
  - Surrogate dual
  - Lagrangean dual
  - Superadditive dual
  - LP dual is an exception
- All are **inference duals** and **relaxation duals**.

Exact and Heuristic Methods

- **Heuristic methods** are key to solving the primal problem.
  - Exact and heuristic methods can be **unified**.
  - They have **common primal/dual structure**.
- Advantages of unification
  - Allows transfer of **inference** and **relaxation** techniques from exact methods to heuristics.
  - And transfer of **local search** ideas to exact methods.
    - For example, strong branching in MIP can be viewed as a local search method for solving a dual problem.

Exact and Heuristic Methods

- Another advantage of unification
  - There is no reason *a priori* that one heuristic method should work better than another.
    - "No free lunch" theorem.

Exact and Heuristic Methods

- Another advantage of unification
  - There is no reason *a priori* that one heuristic method should work better than another.
    - "No free lunch" theorem.
  - Solution methods must exploit problem structure.
    - "Full employment" theorem.

Exact and Heuristic Methods

- Another advantage of unification
  - There is no reason *a priori* that one heuristic method should work better than another.
    - "No free lunch" theorem.
  - Solution methods must exploit problem structure.
    - "Full employment" theorem.
  - **Inference and relaxation methods exploit problem structure**.

Outline

- Primal-dual framework
  - Inference dual
  - Relaxation dual
  - Constraint-directed search
  - DPLL
- Exact Methods
  - Simplex
  - Branch and bound
  - Benders decomposition
- Heuristic Methods
  - Local search
  - GRASP
  - Tabu search
  - Genetic algorithms
  - Ant colony optimization
  - Particle swarm optimization
- Summing up

Outline

- Caveat…
  - This is a high-level talk.
    - Don’t worry too much about technical details.

Outline

- **Primal-dual framework**
  - **Inference dual**
  - **Relaxation dual**
  - **Constraint-directed search**
  - **DPLL**
- Exact Methods
  - Simplex
  - Branch and bound
  - Benders decomposition
- Heuristic Methods
  - Local search
  - GRASP
  - Tabu search
  - Genetic algorithms
  - Ant colony optimization
  - Particle swarm optimization
- Summing up

Inference Dual

- Find the tightest bound on objective function that can be deduced from the constraints.
  - Using a specified method of logical deduction.

Primal: $\min_{x \in S} \{ f(x) \}$

Inference dual: $\max_{v,P} \left\{ v \,\middle|\, (x \in S) \overset{P}{\Rightarrow} (f(x) \geq v) \right\}$

where *P* belongs to a proof family

Inference Dual

- For example, LP dual:

Primal $$\min_{x \geq 0} \{ cx \mid Ax \geq b \}$$

Dual $$\max_{v,P} \left\{ v \mid (Ax \geq b) \overset{P}{\Rightarrow} (cx \geq b) \right\}$$

where proof P is nonnegative linear combination

i.e. $uAx \geq ub$ dominates $cx \geq v$ for $u \geq 0$

Inference Dual

- For example, LP dual:

Primal $\min_{x \geq 0} \{ cx \mid Ax \geq b \}$

Dual $\max_{v,P} \left\{ v \mid (Ax \geq b) \overset{P}{\Rightarrow} (cx \geq b) \right\}$

where proof P is nonnegative linear combination

i.e. $uAx \geq ub$ dominates $cx \geq v$ for $u \geq 0$

i.e. $uA \leq c$ and $ub \geq v$. This yields

Classical dual $\max_{u \geq 0} \{ ub \mid uA \leq c \}$

Inference Dual

- Standard optimization duals are inference duals that use different inference methods.
  - LP dual
    - Nonnegative linear combination + domination
  - Surrogate dual
    - Same, but for NLP, IP
  - Lagrangean dual
    - Same, but with stronger form of domination
  - Subadditive dual
    - Subadditive homogeneous function + domination

Relaxation Dual

- Find a relaxation that gives the tightest bound on the objective function.
  - Relaxation is parameterized by **dual variables**.

Primal: $\min_{x \in S} \{ f(x) \}$

Relaxation dual: $\max_{u \in U} \{ \theta(u) \}$

where $\theta(u) = \min_{x \in S'(u)} \{ f'(x,u) \}$

Relaxation of primal, parameterized by *u*

Relaxation Dual

- Example: Lagrangean dual.

Primal: $\min_{x \in S} \{ f(x) \mid g(x) \leq 0 \}$

Relaxation dual: $\max_{u \in U} \{ \theta(u) \}$

where $\theta(u) = \min_{x \in S} \{ f(x) + u g(x) \}$

Primal-Dual Algorithm

- Enumerate restrictions
  - Branching tree nodes
  - Benders subproblems
  - Local search neighborhoods
- Derive bounds to prune search
  - From inference or relaxation dual.
  - For example, LP bounds in MIP.

Primal-Dual Algorithm

- Key issue: How restrictive are the restrictions?
  - Tighten restrictions until they can be solved
    - As in branching.
  - Relax restrictions until solution quality improves
    - As in large neighborhood search

Primal-Dual Algorithm

- Let inference dual guide the search
  - Constraint-directed search
  - Solution of inference dual provides nogood constraint, as in:
    - Branching (perhaps with conflict analysis)
    - SAT algorithms with clause learning
    - Benders decomposition
    - Dynamic backtracking
    - Tabu search

Primal-Dual Algorithm

- Let relaxation dual guide the search
  - Solution of relaxation suggests how to tighten it.
  - As when branching on fractional variables.

Constraint-Directed Search

- Start with example: SAT.
  - Solving with DPLL.
    - Use branching + unit clause rule:

$$(\bar{x}_1 \vee x_2 \vee x_3 \vee \bar{x}_4) \text{ and } \bar{x}_2 \Rightarrow (\bar{x}_1 \vee x_3 \vee \bar{x}_4)$$

  - Dual solution at infeasible node is unit clause **proof**.
    - Identify branches that play a role in the proof.
    - This yields a **nogood constraint** (conflict clause).

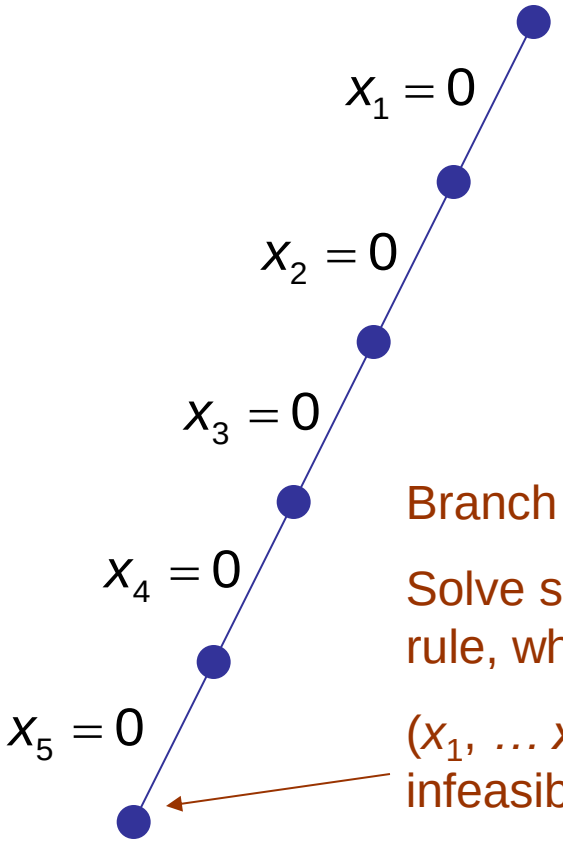
Constraint-Directed Search

- The problem:

  Find a satisfying solution.

$$
\begin{array}{l}
x_1 \vee x_5 \vee x_6 \\
x_1 \vee x_5 \vee \overline{x}_6 \\
x_2 \vee \overline{x}_5 \vee x_6 \\
x_2 \vee \overline{x}_5 \vee \overline{x}_6 \\
\overline{x}_1 \vee x_3 \vee x_4 \\
\overline{x}_2 \vee x_3 \vee x_4 \\
\overline{x}_1 \vee \overline{x}_3 \\
\overline{x}_1 \vee \overline{x}_4 \\
\overline{x}_2 \vee \overline{x}_3 \\
\overline{x}_2 \vee \overline{x}_4
\end{array}
$$

DPLL

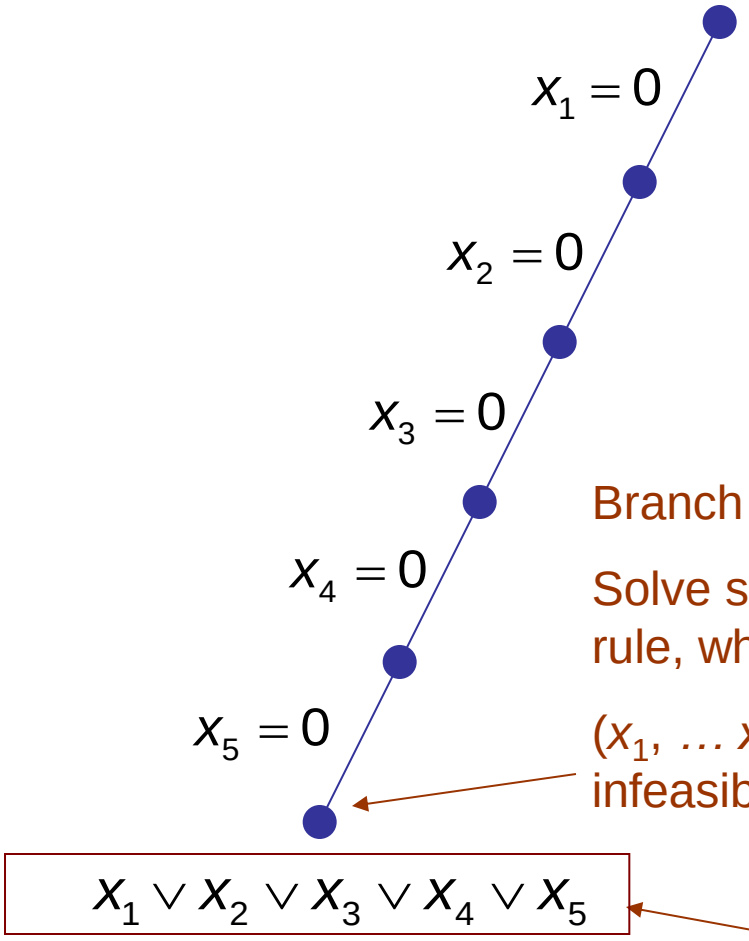


Branch to here.

Solve subproblem with unit clause rule, which proves infeasibility.

(x_1, … x_5) = (0, …, 0) creates the infeasibility.

DPLL



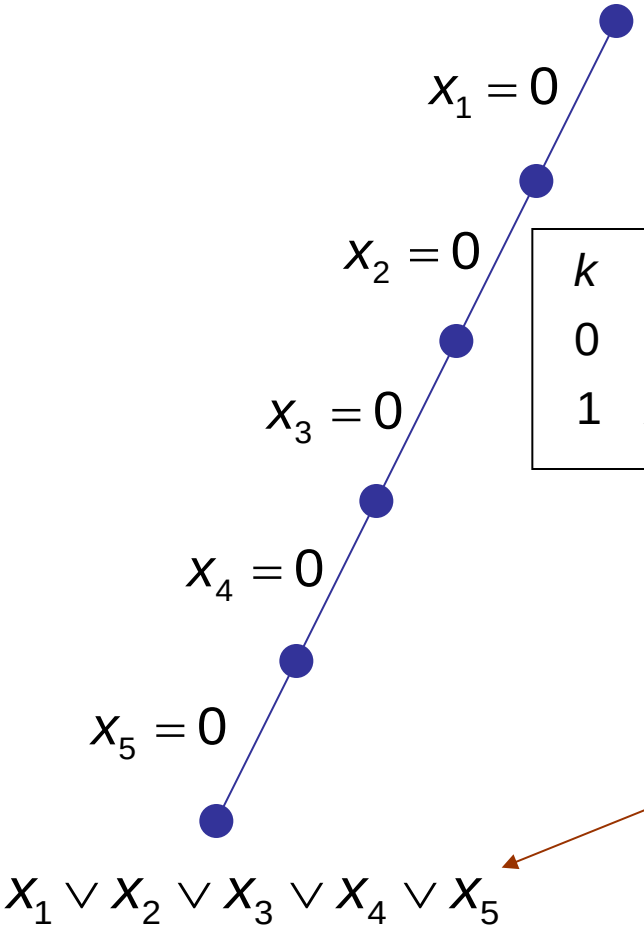
Branch to here.

Solve subproblem with unit clause rule, which proves infeasibility.

(x_1, … x_5) = (0, …, 0) creates the infeasibility.

Generate nogood.

DPLL

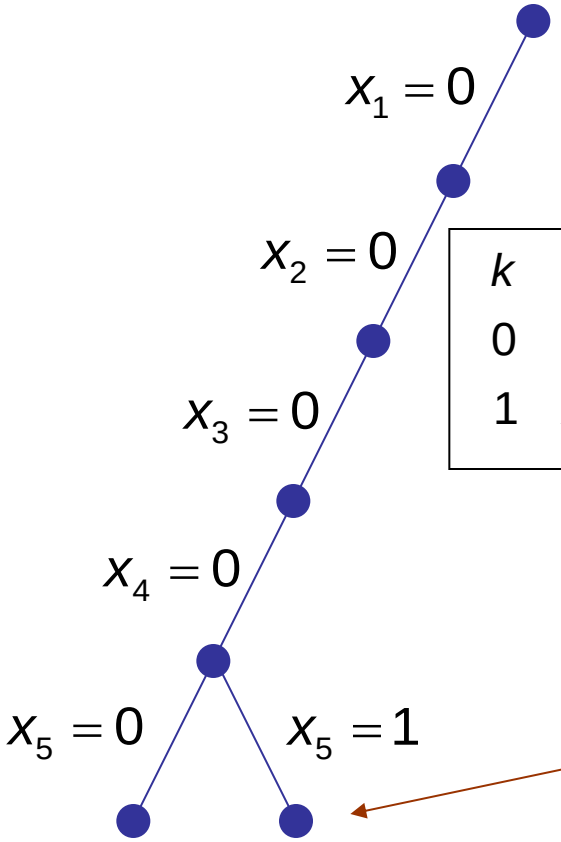


Consists of processed nogoods in iteration *k*

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$		

Conflict clause appears as nogood induced by solution of R_k.

DPLL

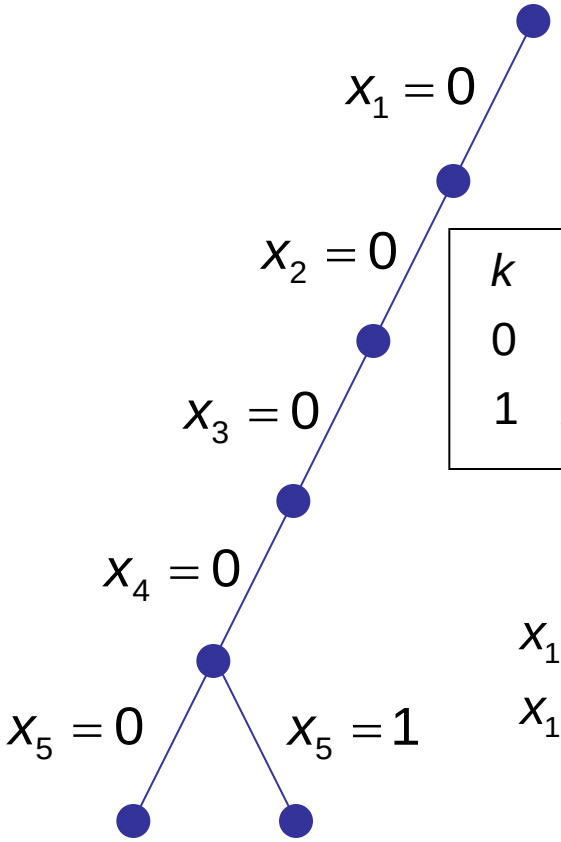


Consists of processed nogoods in iteration *k*

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

Go to solution that solves relaxation, with priority to 0

DPLL



Consists of processed nogoods in iteration *k*

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

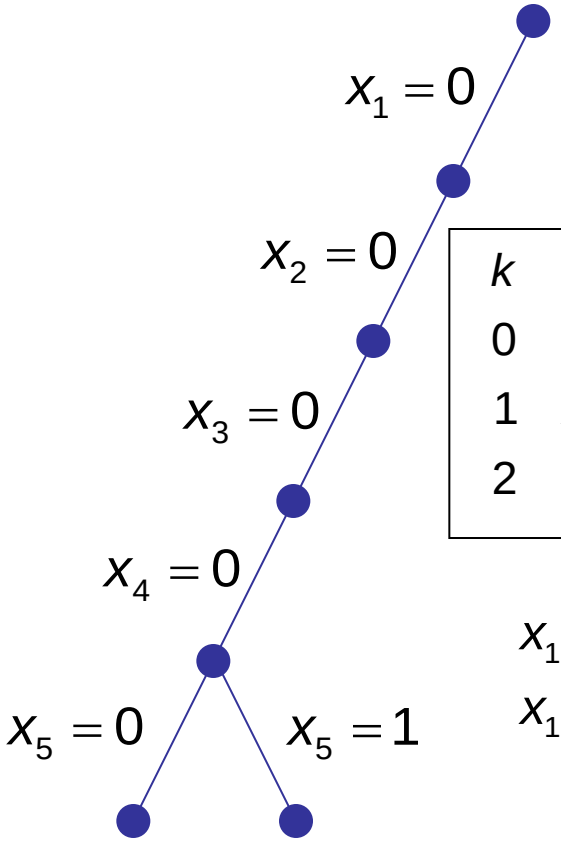
Process nogood set with **parallel resolution**

$x_1 \vee x_2 \vee x_3 \vee x_4$ **parallel-absorbs**

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

DPLL



Consists of processed nogoods in iteration *k*

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$
2	$x_1 \vee x_2 \vee x_3 \vee x_4$		

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

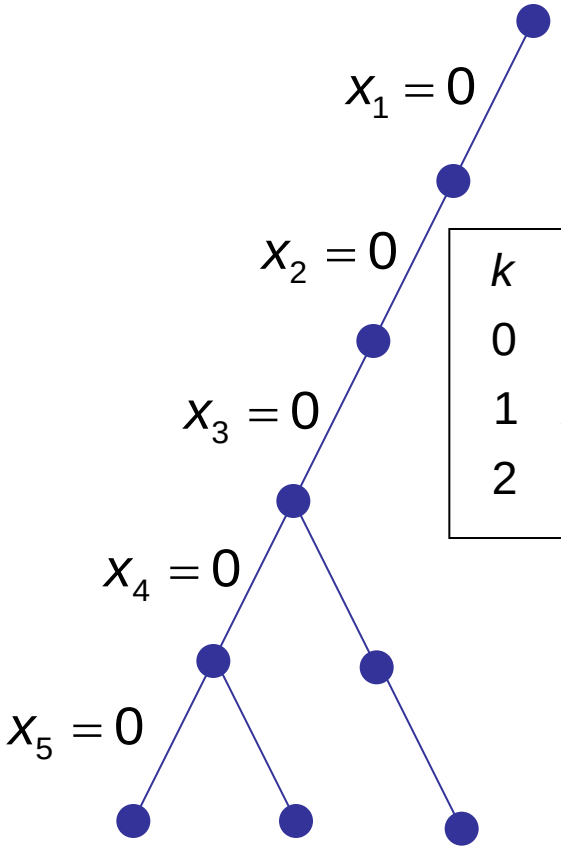
Process nogood set with **parallel resolution**

$x_1 \vee x_2 \vee x_3 \vee x_4$ **parallel-absorbs**

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

DPLL



Consists of processed nogoods in iteration *k*

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$
2	$x_1 \vee x_2 \vee x_3 \vee x_4$	$(0,0,0,1,0,\cdot)$	

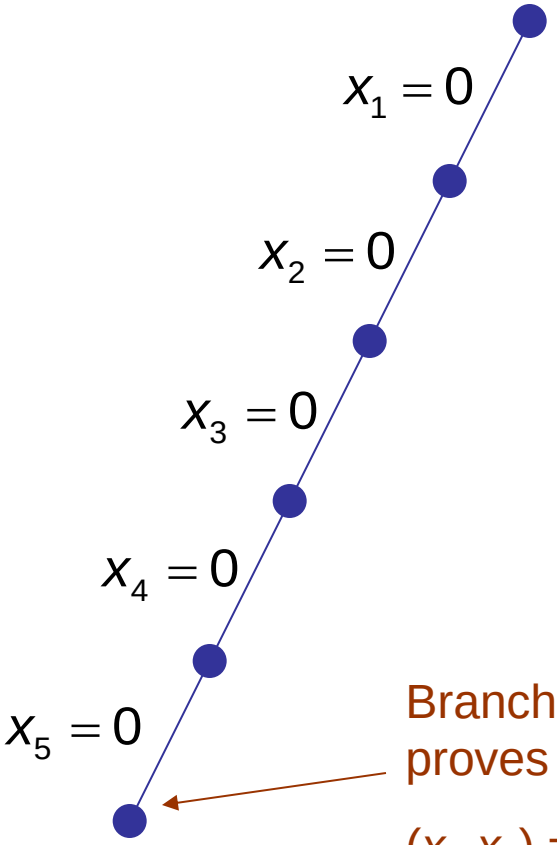
Solve relaxation again, continue.

So backtracking is nogood-based search with parallel resolution

Constraint-Directed Search

- Use stronger nogoods = conflict clauses.
  - Nogoods rule out only branches that play a role in unit clause proof.

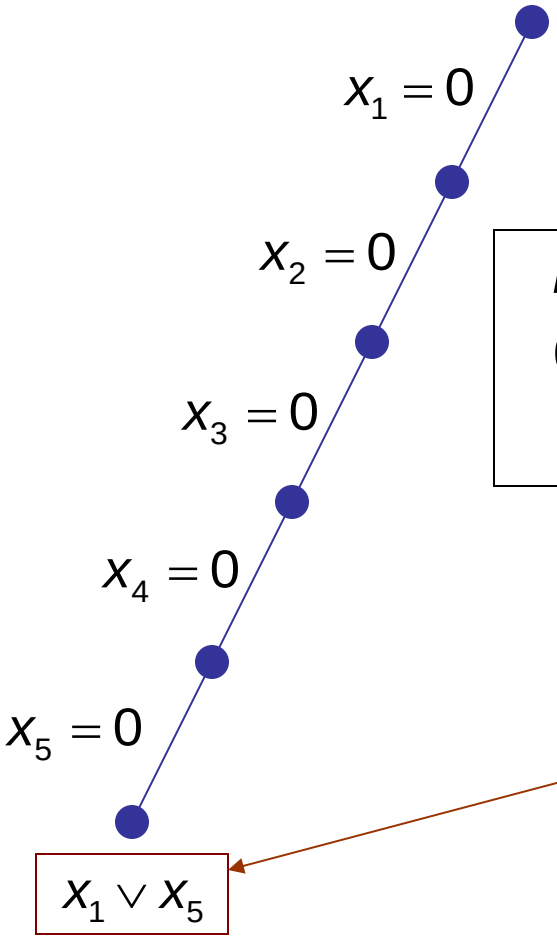
DPLL with conflict clauses



Branch to here. Unit clause rule proves infeasibility.

(x_1,x_5) = (0,0) is only premise of unit clause proof.

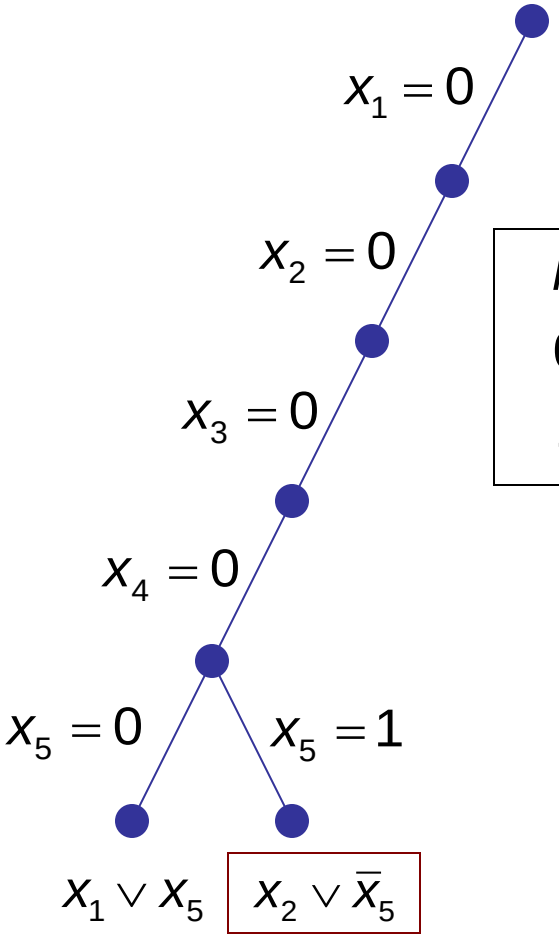
DPLL with conflict clauses



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.

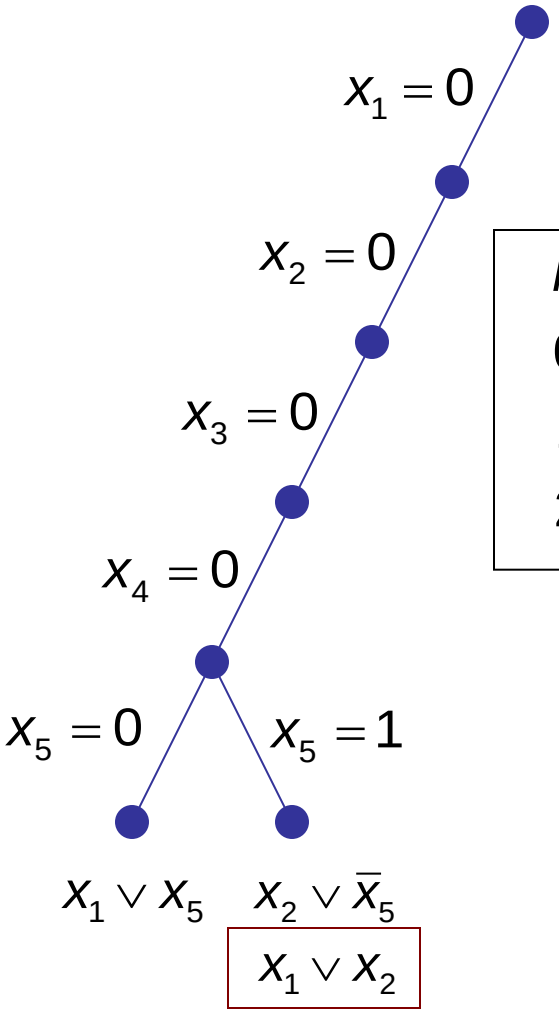
DPLL with conflict clauses



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	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$

DPLL with conflict clauses



Consists of processed nogoods

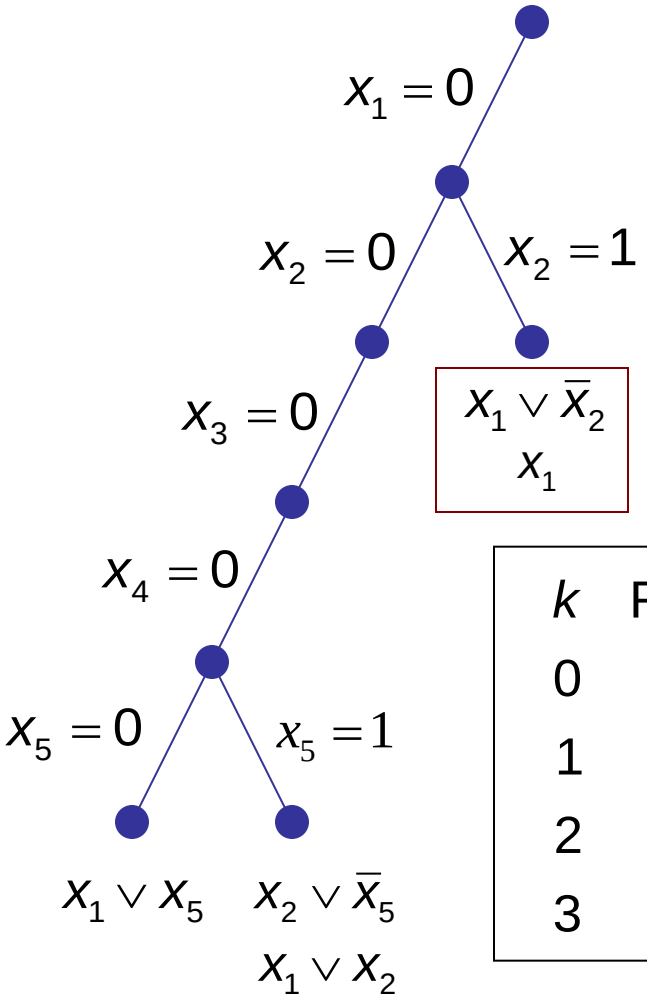
k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$
2	$x_1 \vee x_2$		

$x_1 \vee x_5$
$x_2 \vee \bar{x}_5$

parallel-resolve to yield

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

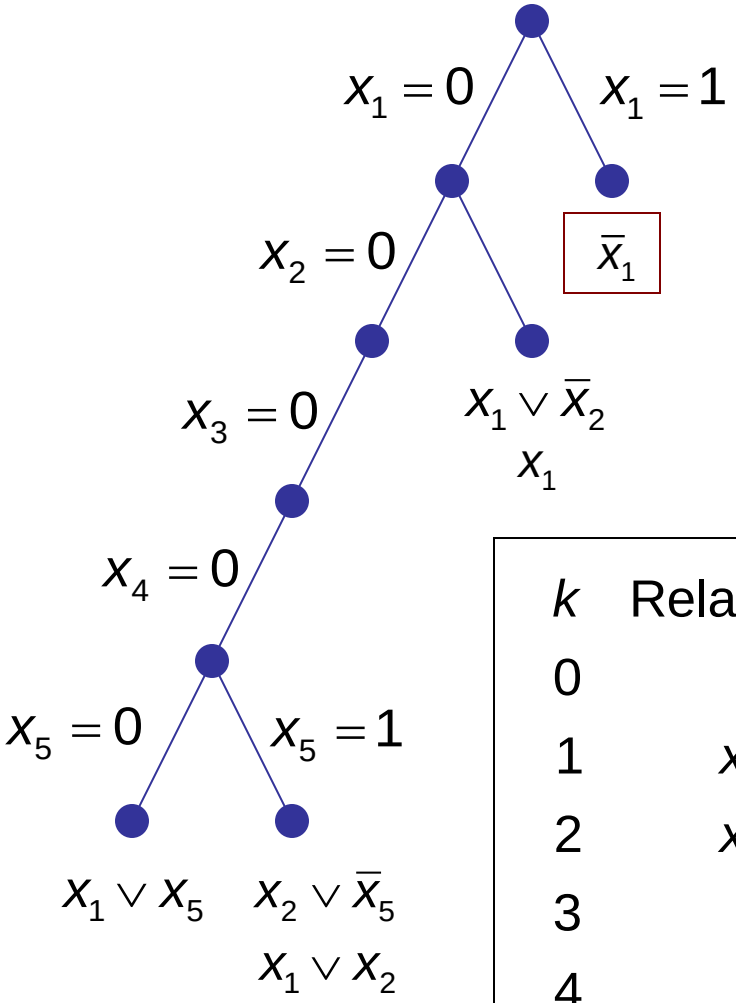
DPLL with conflict clauses



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		

$x_1 \vee x_2$
$x_1 \vee \bar{x}_2$ parallel-resolve to yield x_1

DPLL with conflict clauses



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	$\varnothing$		

Search terminates

Constraint-Directed Search

- Suppose we search over partial solutions.
  - Let $x = (y,z)$. In each iteration, fix y to $\bar{y}$.
    - Partition $x = (y,z)$ can change in each iteration.
  - This defines subproblem:

$$\min_{(\bar{y},z)\in S} \{f(\bar{y},z)\}$$

Constraint-Directed Search

- Suppose we search over partial solutions.
  - Let $x = (y,z)$. In each iteration, fix y to $\bar{y}$.
    - Partition $x = (y,z)$ can change in each iteration.
  - This defines subproblem:

$$\min_{(\bar{y},z)\in S} \{f(\bar{y},z)\}$$

  - Let (v^*,P^*) solve the subproblem dual:

$$\max_{v,P} \left\{ v \;\middle|\; ((\bar{y},z)\in S) \overset{P}{\Rightarrow} (f(\bar{y},z) \geq v) \right\}$$

  - Then we have a nogood constraint (e.g., Benders cut):

$$v \geq \begin{cases} v^* & \text{if } y = \bar{y} \\ \text{LB}(y) & \text{if } y \neq \bar{y} \end{cases}$$

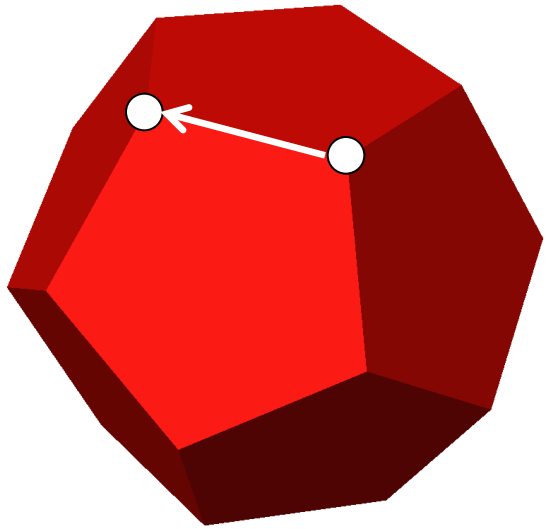
Bound obtained from proof P^* for fixed values y

Outline

- Primal-dual framework
  - Inference dual
  - Relaxation dual
  - Constraint-directed search
  - DPLL
- **Exact Methods**
  - **Simplex**
  - **Branch and bound**
  - **Benders decomposition**
- Heuristic Methods
  - Local search
  - GRASP
  - Tabu search
  - Genetic algorithms
  - Ant colony optimization
  - Particle swarm optimization
- Summing up

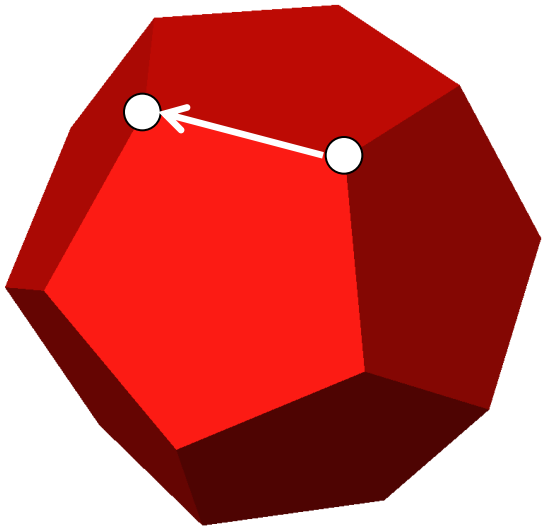
Exact Methods: Simplex

- Searches over problem restrictions
  - Namely, a neighborhood (edge) of each iterate.
  - A local search that happens to be complete, due to choice of neighborhood.



Exact Methods: Simplex

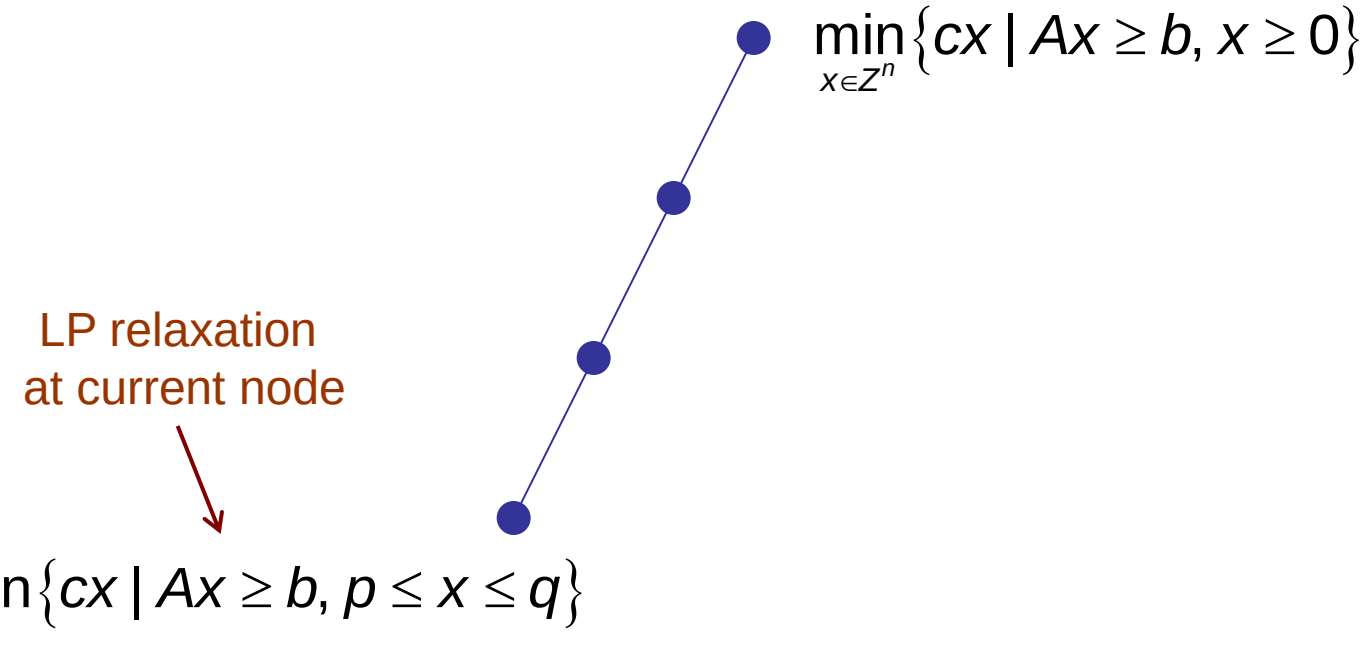
- Searches over problem restrictions
  - Namely, a neighborhood (edge) of each iterate.
  - A local search that happens to be complete, due to choice of neighborhood.
- Dual of **subproblem** (edge) can provide reduced costs.
  - These select the next neighborhood.
  - Requires slightly nonstandard analysis.



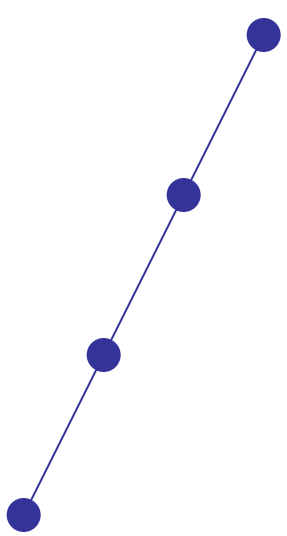
Exact Methods: Branch and Bound

- Branch and bound with conflict analysis.
  - Now used in some solvers.
  - Conflict analysis yields a nogood constraint.
  - Permits **backjumping** as in DPLL.
  - An old idea in AI, new in OR.

Branch and Bound with Conflict Analysis



Branch and Bound with Conflict Analysis



$$\min_{x \in Z^n} \{cx \mid Ax \geq b, x \geq 0\}$$

LP relaxation
at current node

$$\min_{x} \{cx \mid Ax \geq b, p \leq x \leq q\}$$

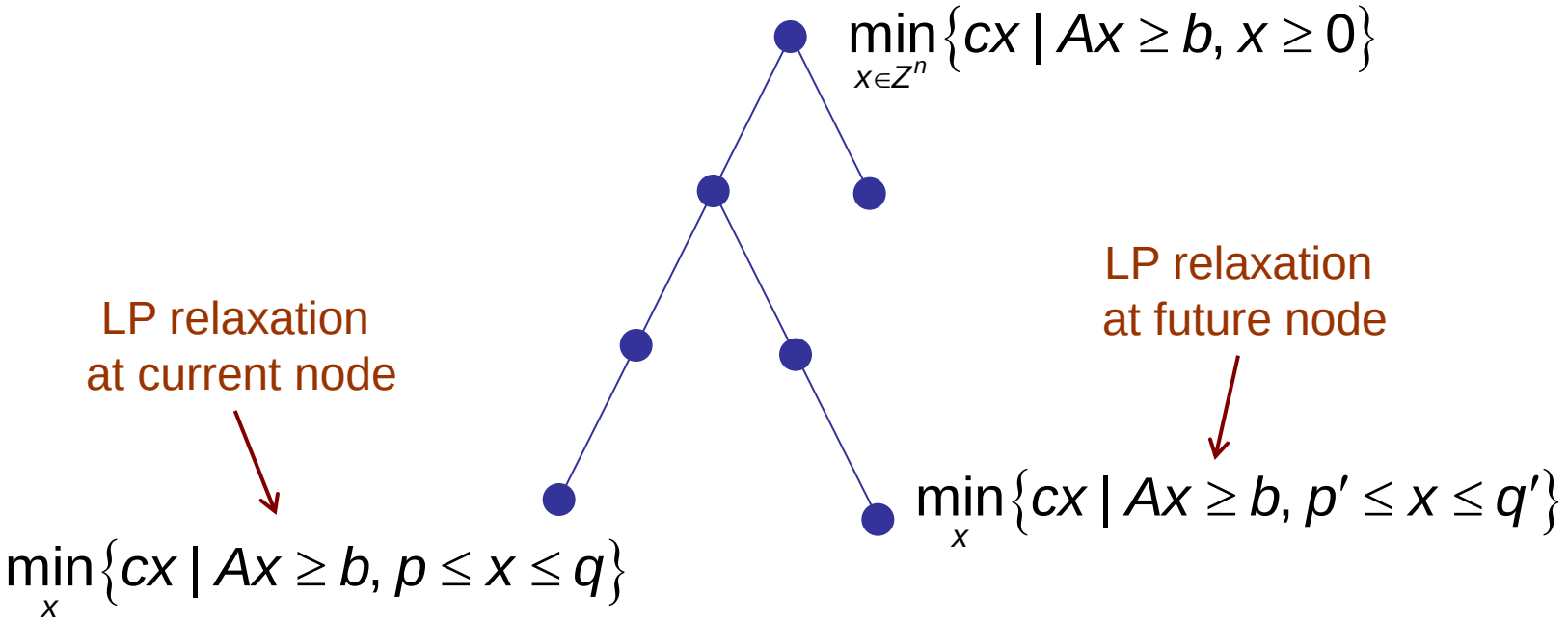
If LP is infeasible, we have

$$uA + \sigma - \tau \leq 0$$

$$ub + \sigma p - \tau q > 0$$

where u, σ, τ are dual variables.

Branch and Bound with Conflict Analysis



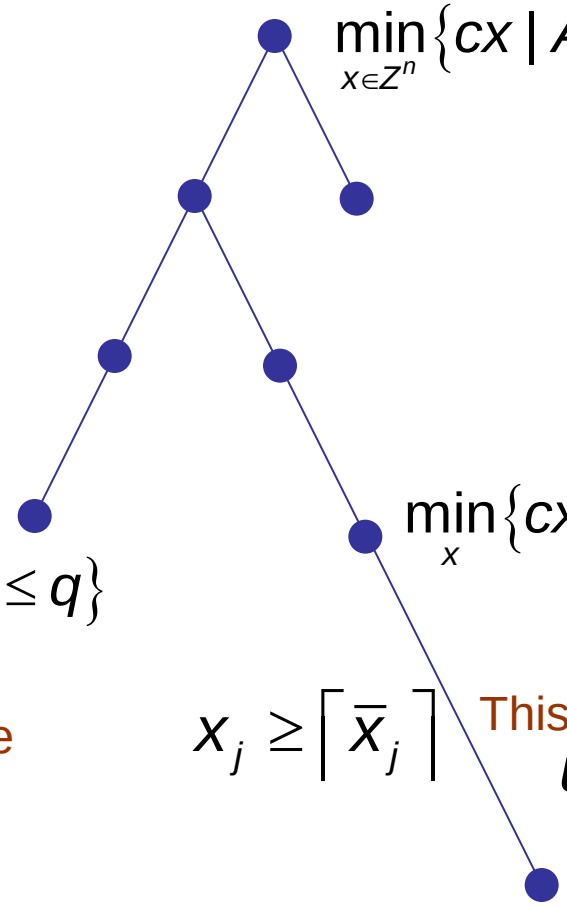
If LP is infeasible, we have

$$uA + \sigma - \tau \leq 0$$

$$ub + \sigma p - \tau q > 0$$

where u, σ, τ are dual variables.

Branch and Bound with Conflict Analysis



$$\min_{x \in Z^n} \{cx \mid Ax \geq b, x \geq 0\}$$

LP relaxation at current node

$$\min_{x} \{cx \mid Ax \geq b, p \leq x \leq q\}$$

LP relaxation at future node

$$\min_{x} \{cx \mid Ax \geq b, p' \leq x \leq q'\}$$

$$x_j \geq \lceil \bar{x}_j \rceil$$

This branch is feasible only if

$$ub + \sigma p' - \tau q' \leq 0$$

when

$$p'_j = \lceil \bar{x}_j \rceil$$

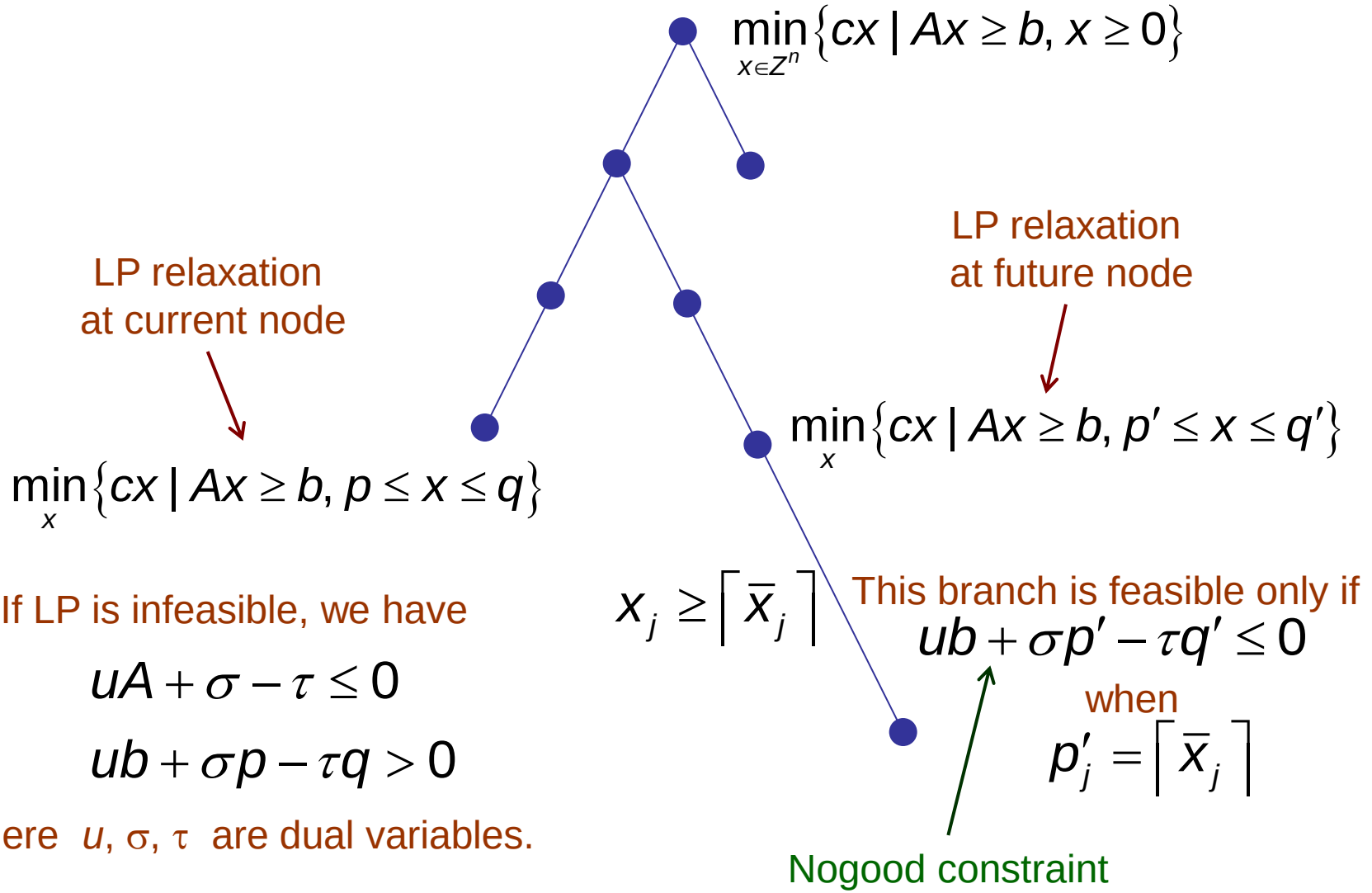
If LP is infeasible, we have

$$uA + \sigma - \tau \leq 0$$

$$ub + \sigma p - \tau q > 0$$

where u, σ, τ are dual variables.

Branch and Bound with Conflict Analysis



Exact Methods: Branch and Bound

- To convert to heuristic method…
  - Let nogood constraints alone guide next trial solution.
  - Drop older nogood constraints.
  - Becomes a form of tabu search.

Exact Methods: Branch and Bound

- Branch and bound as dual local search.
  - Let v_i^* be LP value at bounding nodes i of current tree T.
  - T parameterizes a relaxation:

$$\theta(T) = \min_i \left\{ v_i^* \right\}$$

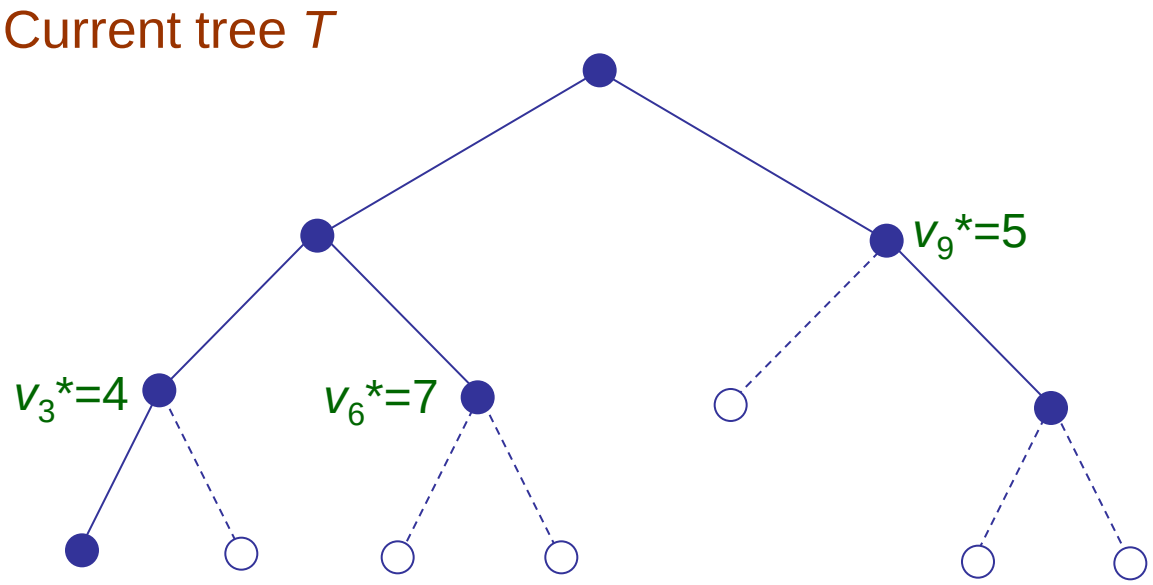
  - **Relaxation dual** is

$$\max_T \theta(T)$$

    - Complete T solves the dual problem.

  - We can maximize $\theta(T)$ by local search (hill-climbing)
    - as when solving Lagrangean dual (for which θ is concave)

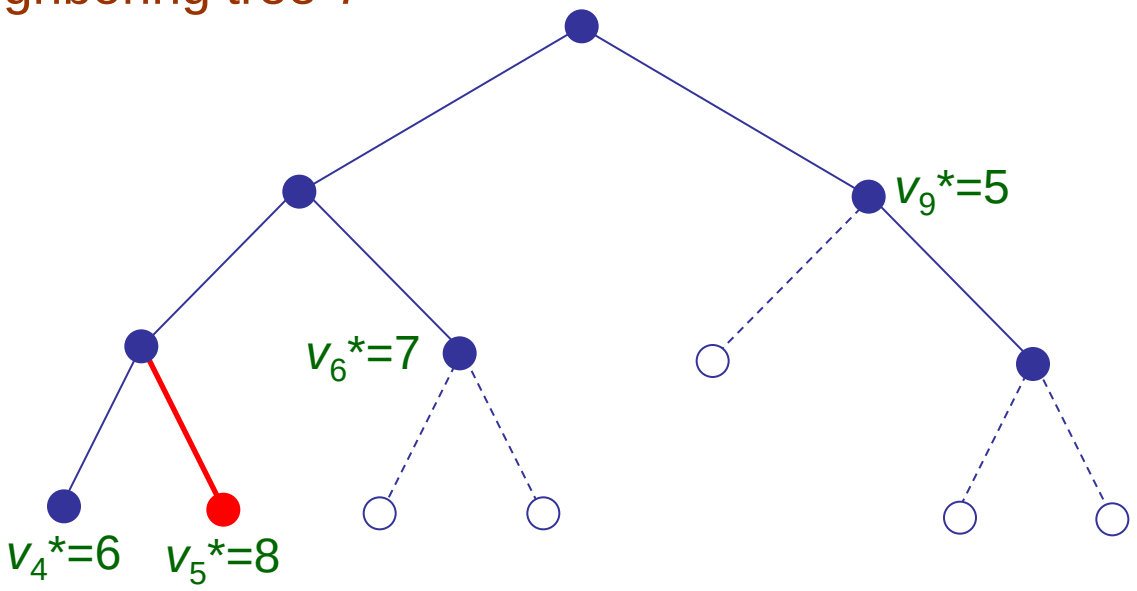
Dual Local Search



$\theta(T) = \min \{ 4, 7, 5 \} = 4$

Dual Local Search

Neighboring tree T'

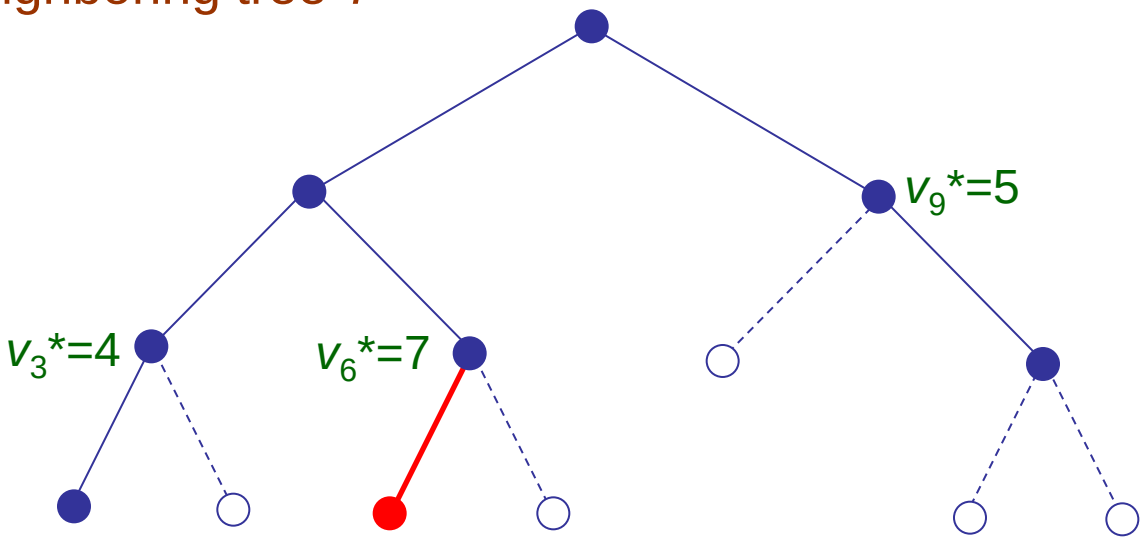


$\theta(T') = \min \{ 6, 8, 7, 5 \} = 5$

Increase of 1

Dual Local Search

Neighboring tree T'

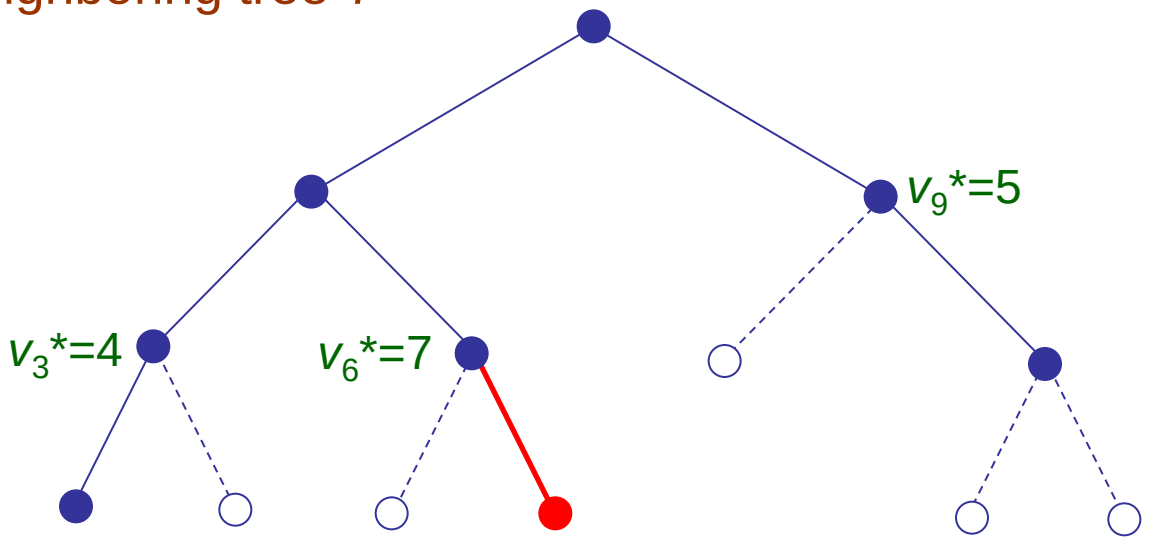


$\theta(T') = \min \{ 4, 7, 5 \} = 4$

No change

Dual Local Search

Neighboring tree T'

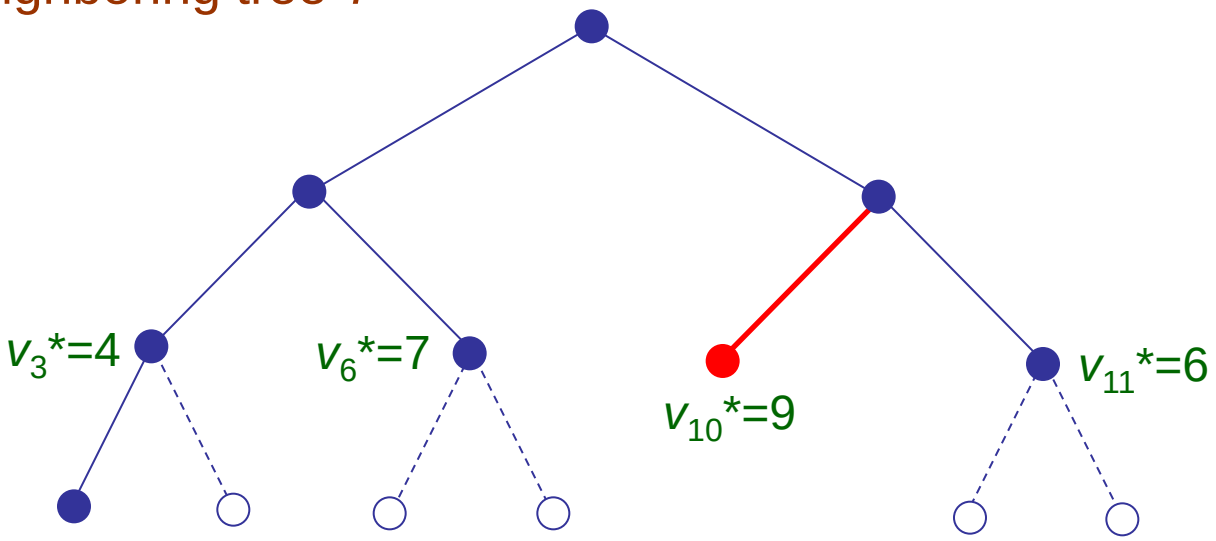


$\theta(T') = \min \{ 4, 7, 5 \} = 4$

No change

Dual Local Search

Neighboring tree T'

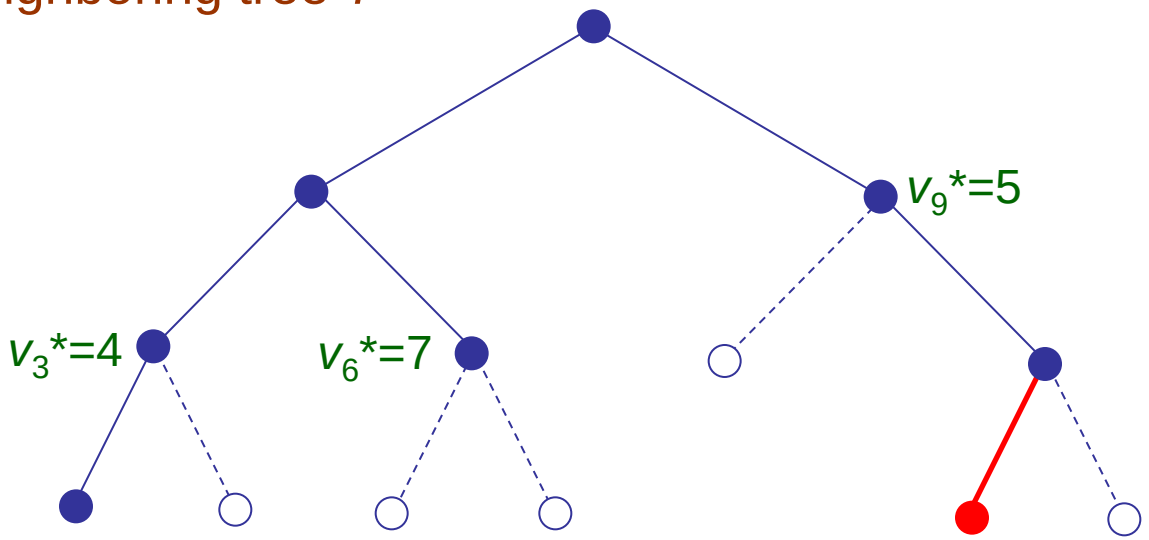


$\theta(T') = \min \{ 4, 7, 9, 6 \} = 4$

No change

Dual Local Search

Neighboring tree T'

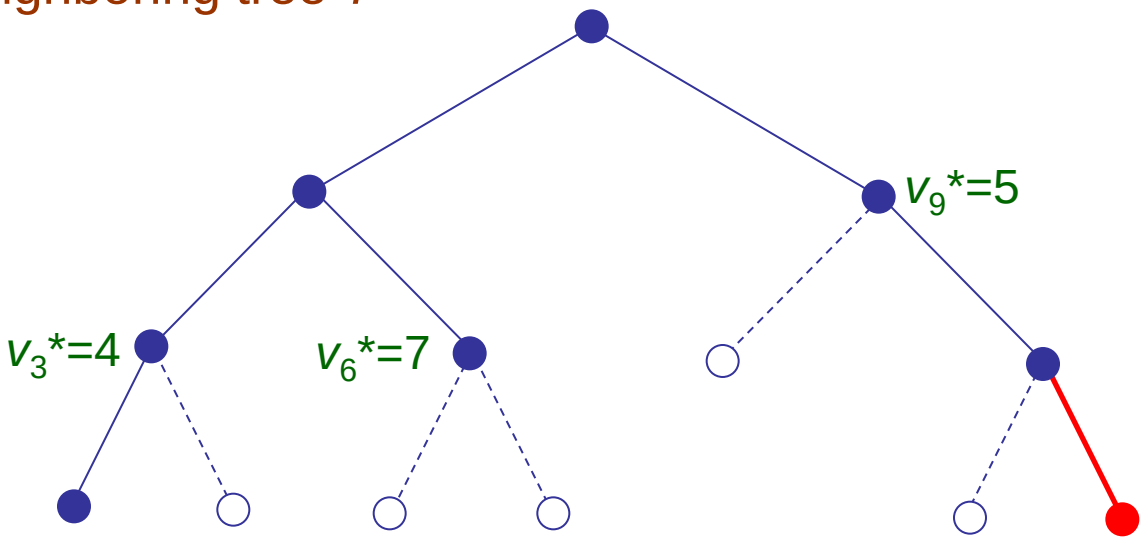


$\theta(T') = \min \{ 4, 7, 5 \} = 4$

No change

Dual Local Search

Neighboring tree T'



$\theta(T') = \min \{ 4, 7, 5\} = 4$

No change

Exact Methods: Branch and Bound

- Strong branching
  - Selects neighboring tree by adding node with best LP value
  - This usually has no effect on the bound $\theta(T)$.

Exact Methods: Branch and Bound

- Strong branching
  - Selects neighboring tree by adding node with best LP value
  - This usually has no effect on the bound $\theta(T)$.
- Primal vs dual search
  - The search tree traditionally solves primal and dual problem simultaneously.
  - Primal and dual call for different search methods.
  - Recent solvers use primal heuristics – good idea.
  - Now let the tree focus on the dual.

Benders Decomposition

- In Benders, every restriction is formed by fixing the **same set** of variables.
  - These are the master problem variables
  - Restriction is the resulting subproblem

Benders Decomposition

- In Benders, every restriction is formed by fixing the **same set** of variables.
  - These are the master problem variables
  - Restriction is the resulting subproblem
- It is another form of constraint-directed search
  - Nogood constraints are Benders cuts.
  - As always, they are obtained from inference dual of restriction (subproblem).
- This generalizes Benders to almost any subproblem.
  - Not just LP as in classical Benders.

Benders Decomposition

- Example: Facility assignment and scheduling, to minimize makespan
  - Master problem assigns jobs to facilities.
  - Subproblem schedules jobs in each facility.
- Use resource-constrained scheduling in each facility…

Benders Decomposition

Constraint for resource-constrained scheduling:

$$\text{cumulative}\big((s_1,s_2,s_3),(p_1,p_2,p_3),(c_1,c_2,c_3),C\big)$$

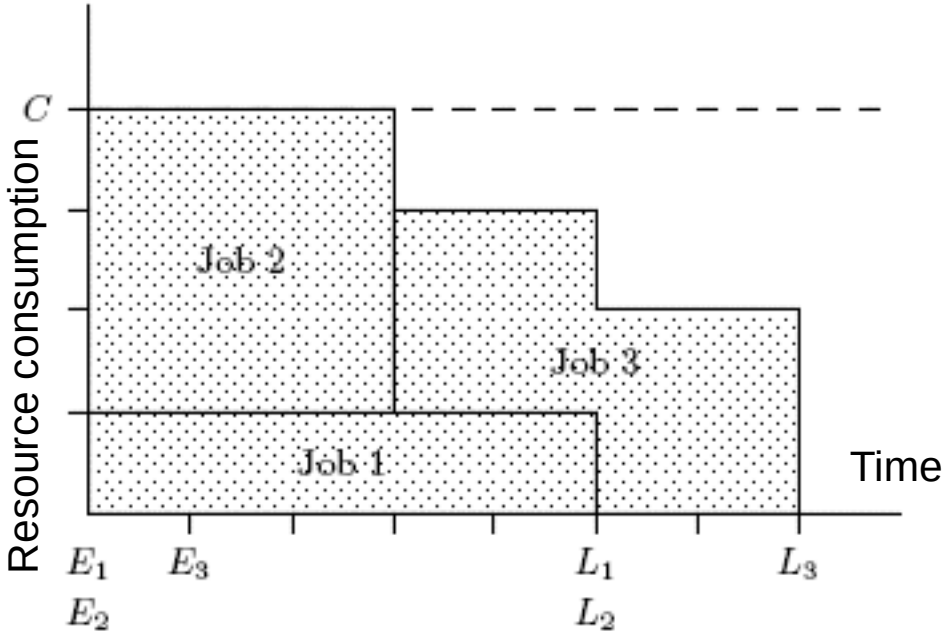
Start times (variables)

Processing times

Resource consumption

Resource limit

A feasible solution:



Benders Decomposition

- Model to minimize makespan:.

$$\min M$$

$$M \geq s_j + p_{x_j j}, \quad \text{all } j$$

$$r_j \leq s_j \leq d_j - p_{x_j j}, \quad \text{all } j$$

$$\text{cumulative}\Big((s_j \mid x_j = i),(p_{ij} \mid x_j = i),(c_{ij} \mid x_j = i)\Big), \quad \text{all } i$$

Start time of job *j*

Facility assigned to job *j*

Benders Decomposition

Subproblem for each facility *i*, given an assignment $\bar{x}$ from master

$$\min M$$

$$M \geq s_j + p_{\bar{x}_j j}, \quad \text{all } j$$

$$r_j \leq s_j \leq d_j - p_{\bar{x}_j j}, \quad \text{all } j$$

$$\text{cumulative}\left((s_j \mid \bar{x}_j = i),(p_{ij} \mid \bar{x}_j = i),(c_{ij} \mid \bar{x}_j = i)\right)$$

Simple Benders cut: Analyze subproblem optimality proof (i.e., inference dual solution) to identify critical subset of jobs that yield min makespan:

$$M \geq M_{ik} \sum_{j \in J_{ik}} p_{ij}(1 - y_{ij})$$

Min makespan on facility *i* in iteration *k*

Critical set of jobs assigned to facility *i* in iteration *k*

=1 if job *j* assigned to facility *i* ($xj = i$)

Benders Decomposition

Subproblem for each facility i, given an assignment $\bar{x}$ from master

$$\min M$$

$$M \geq s_j + p_{\bar{x}_j j}, \quad \text{all } j$$

$$r_j \leq s_j \leq d_j - p_{\bar{x}_j j}, \quad \text{all } j$$

$$\text{cumulative}\left((s_j \mid \bar{x}_j = i),(p_{ij} \mid \bar{x}_j = i),(c_{ij} \mid \bar{x}_j = i)\right)$$

Stronger Benders cut (all release times the same):

$$M \geq M_{ik}\left(\sum_{j \in J_{ik}} p_{ij}(1 - y_{ij}) + \max_{j \in J_{ik}}\{d_j\} - \min_{j \in J_{ik}}\{d_j\}\right)$$

Min makespan on facility i in iteration k

=1 if job j assigned to facility i ($xj = i$)

Deadline for job j

Benders Decomposition

The master problem is

$$\min M$$

$$M \geq M_{ik}\left(\sum_{j \in J_{ik}} p_{ij}(1-y_{ij}) + \max_{j \in J_{ik}}\{d_j\} - \min_{j \in J_{ik}}\{d_j\} \right), \quad \text{all } k$$

Benders cuts

Relaxation of subproblem

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

Outline

- Primal-dual framework
  - Inference dual
  - Relaxation dual
  - Constraint-directed search
  - DPLL
- Exact Methods
  - Simplex
  - Branch and bound
  - Benders decomposition
- **Heuristic Methods**
  - **Local search**
  - **GRASP**
  - **Tabu search**
  - **Genetic algorithms**
  - **Ant colony optimization**
  - **Particle swarm optimization**
- Summing up

Heuristic Methods: Local Search

- Really two kinds of search
  - Search within neighborhoods
    - The “local” part.
  - Search over neighborhoods.
    - A search over problem restrictions.
    - Each neighborhood is feasible set of restriction.

Heuristic Methods: Local Search

- (Very) large neighborhood search
  - Adjust neighborhood size (or structure) for good solution quality.
    - While remaining tractable.
  - Search-over-restrictions provides different perspective.
    - For adjusting neighborhood size.
    - For example, fix some of the variables (as in exact methods).

Heuristic Methods: Local Search

- Example: TSP
  - Search neighborhoods of current tour.
- Traditional: *k*-opt heuristic
  - Adjust neighborhood size by controlling *k*.
  - Lin-Kernighan heuristic is special case.
- Alternative : Circuit formulation
  - Fix *k* variables.
  - Adjust neighborhood size by controlling *k*.

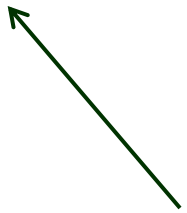
Heuristic Methods: Local Search

- Circuit formulation of TSP:

$$\min \sum_i c_{ix_i}$$

$$\text{circuit}(x_1, \ldots, x_n)$$

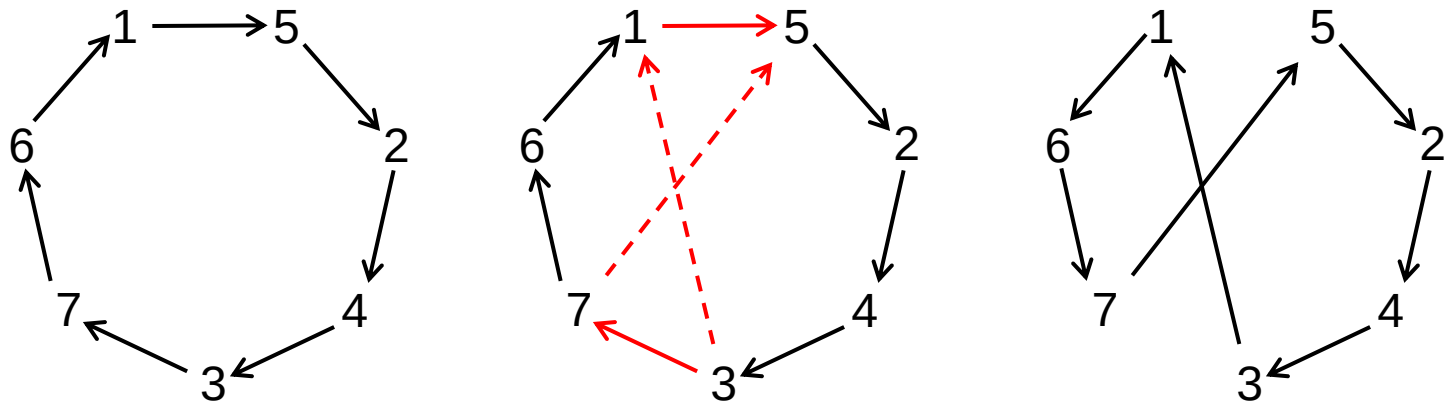
where x_i = city after city i



Requires that x_1. …, x_n describe a hamiltonian cycle

Heuristic Methods: Local Search

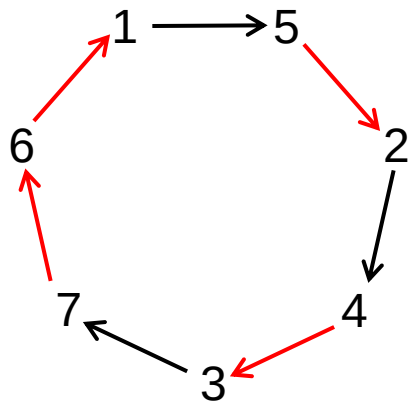
- 2-opt neighborhood
  - Obtained by edge swaps:



  - 14 neighbors

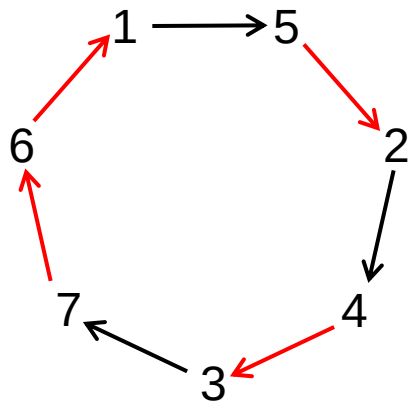
Heuristic Methods: Local Search

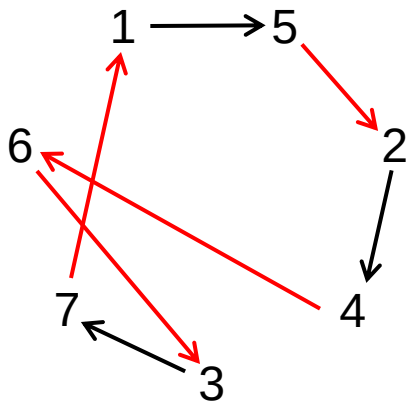
- Circuit neighborhood
  - Current solution $x = (5,4,7,3,2,1,6)$. Fix $(x_1,x_2,x_3) = (5,4,7)$.



Heuristic Methods: Local Search

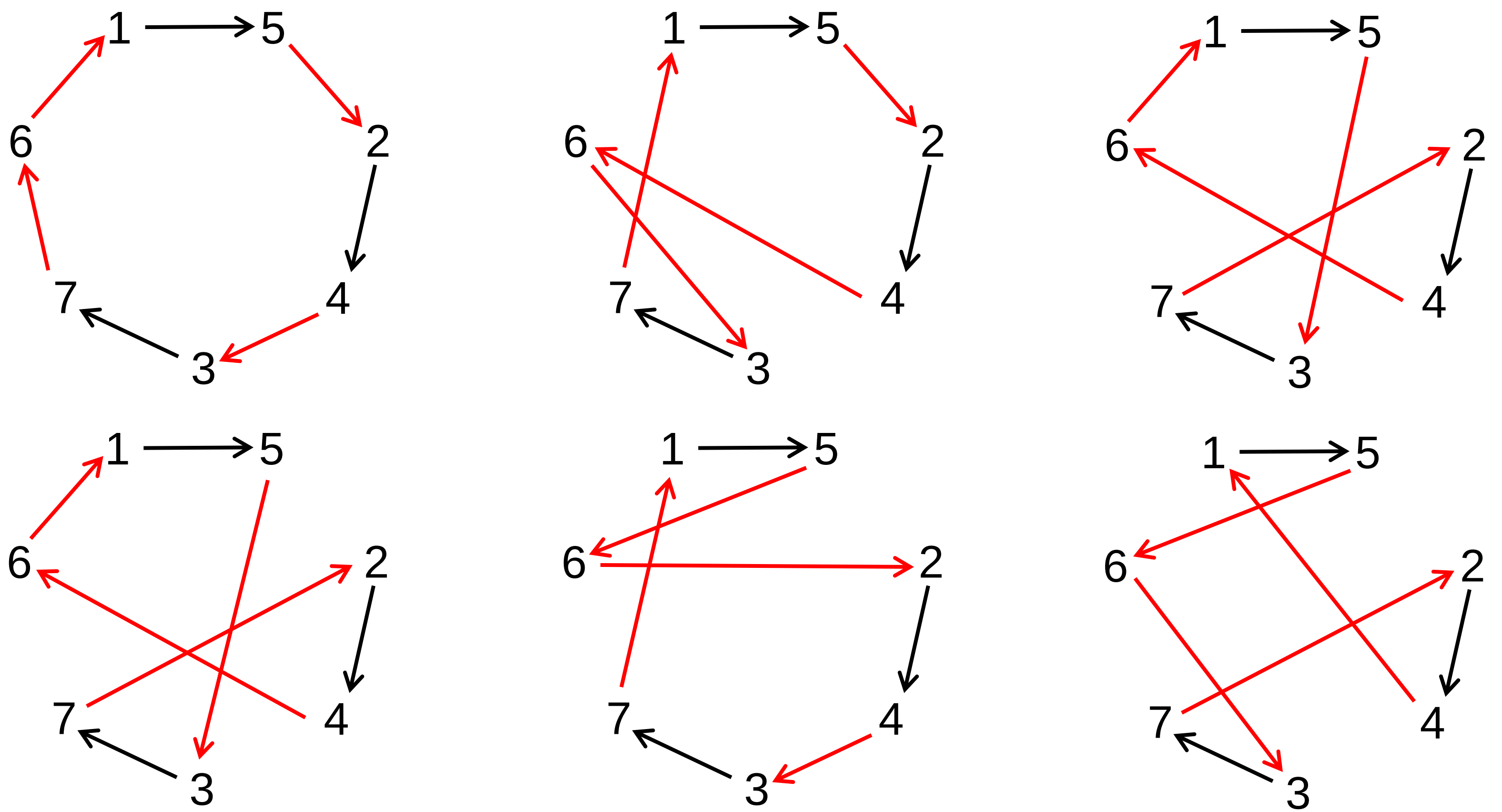
- Circuit neighborhood
  - Current solution $x = (5,4,7,3,2,1,6)$. Fix $(x_1,x_2,x_3) = (5,4,7)$.





Heuristic Methods: Local Search

- Circuit neighborhood
  - Current solution $x = (5,4,7,3,2,1,6)$. Fix $(x_1,x_2,x_3) = (5,4,7)$.



Heuristic Methods: Local Search

- Neighborhood – permutations of chains
  - Fix $(x_1,x_2,x_3,x_4) = (5,4,7,3)$: 2 neighbors
  - Fix $(x_1,x_2,x_3) = (5,4,7)$: 6 neighbors
  - Fix $(x_1,x_2,) = (5,4)$: 24 neighbors
  - Fix $(x_1,) = (5)$: 120 neighbors
- Local search algorithm:
  - Find best solution in neighborhood.
  - Fix a different set of variables to define next neighborhood.
- Exact algorithm is special case
  - Fix no variables.

Heuristic Methods: Local Search

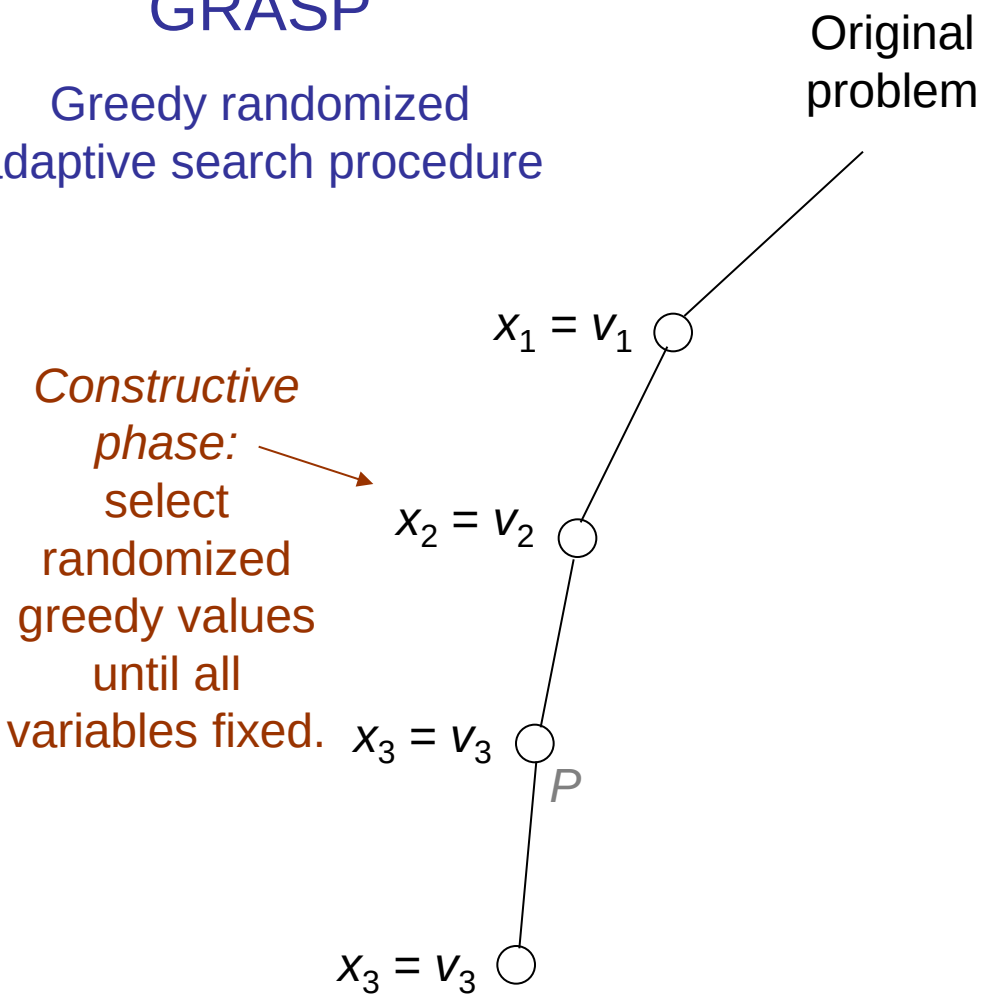
- How to use bounding
  - To control neighborhhood size.
  - Increase neighborhood size until **relaxation** of restricted problem has better value than best known solution.

Heuristic Methods: GRASP

- Greedy randomized adaptive search procedure
  - Greedy constructive phase alternates with local search phase.
- Can be interpreted as incomplete branching
  - Allows a natural generalization…
  - and bounding as in branch and bound.

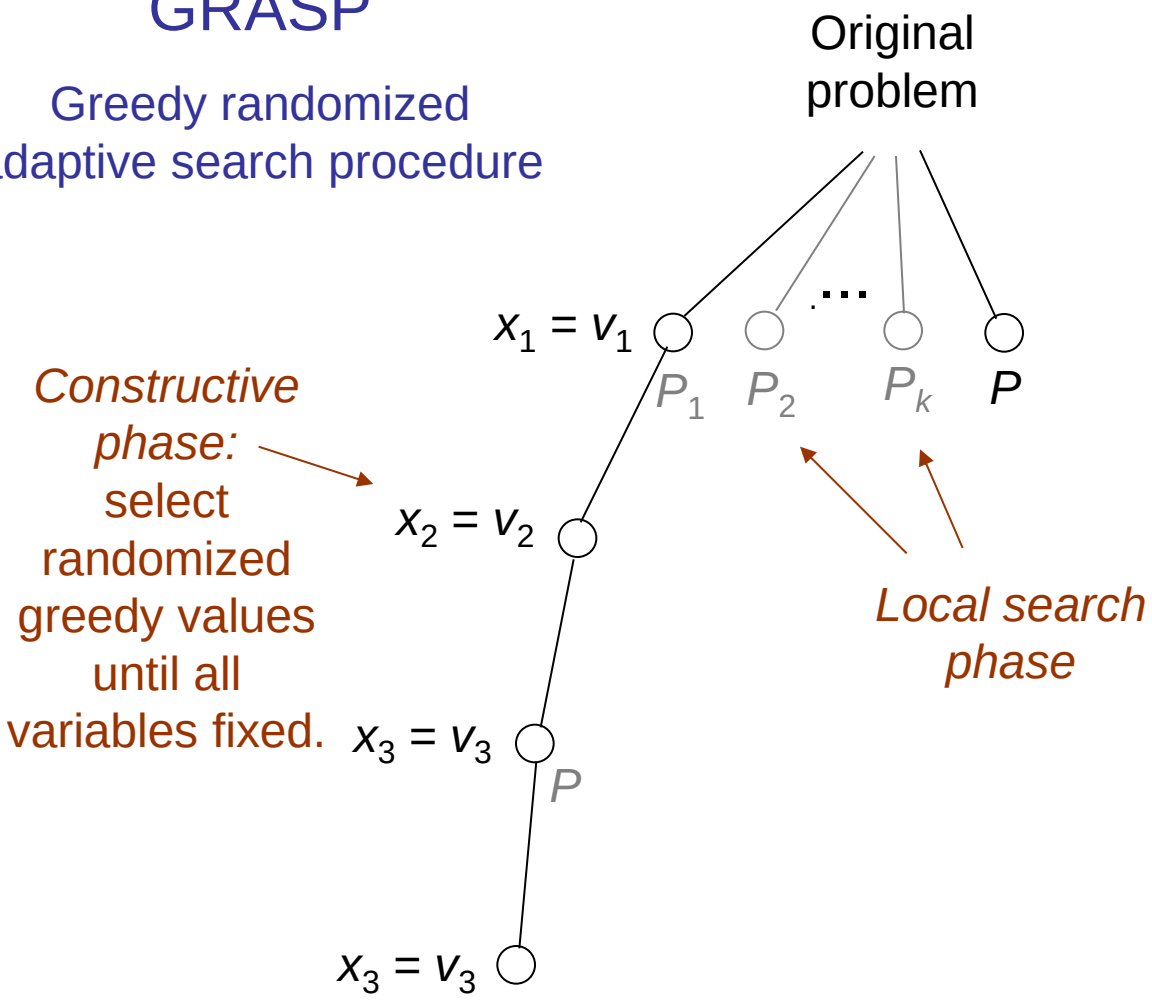
Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure



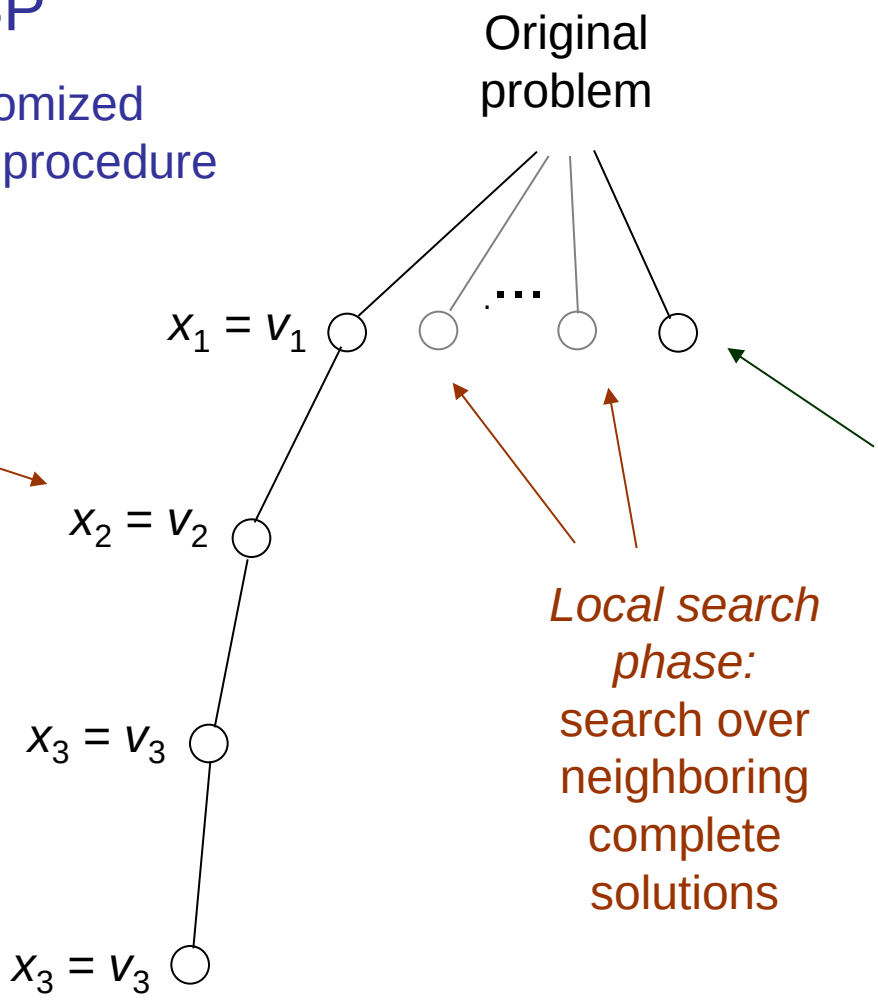
Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure



Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure



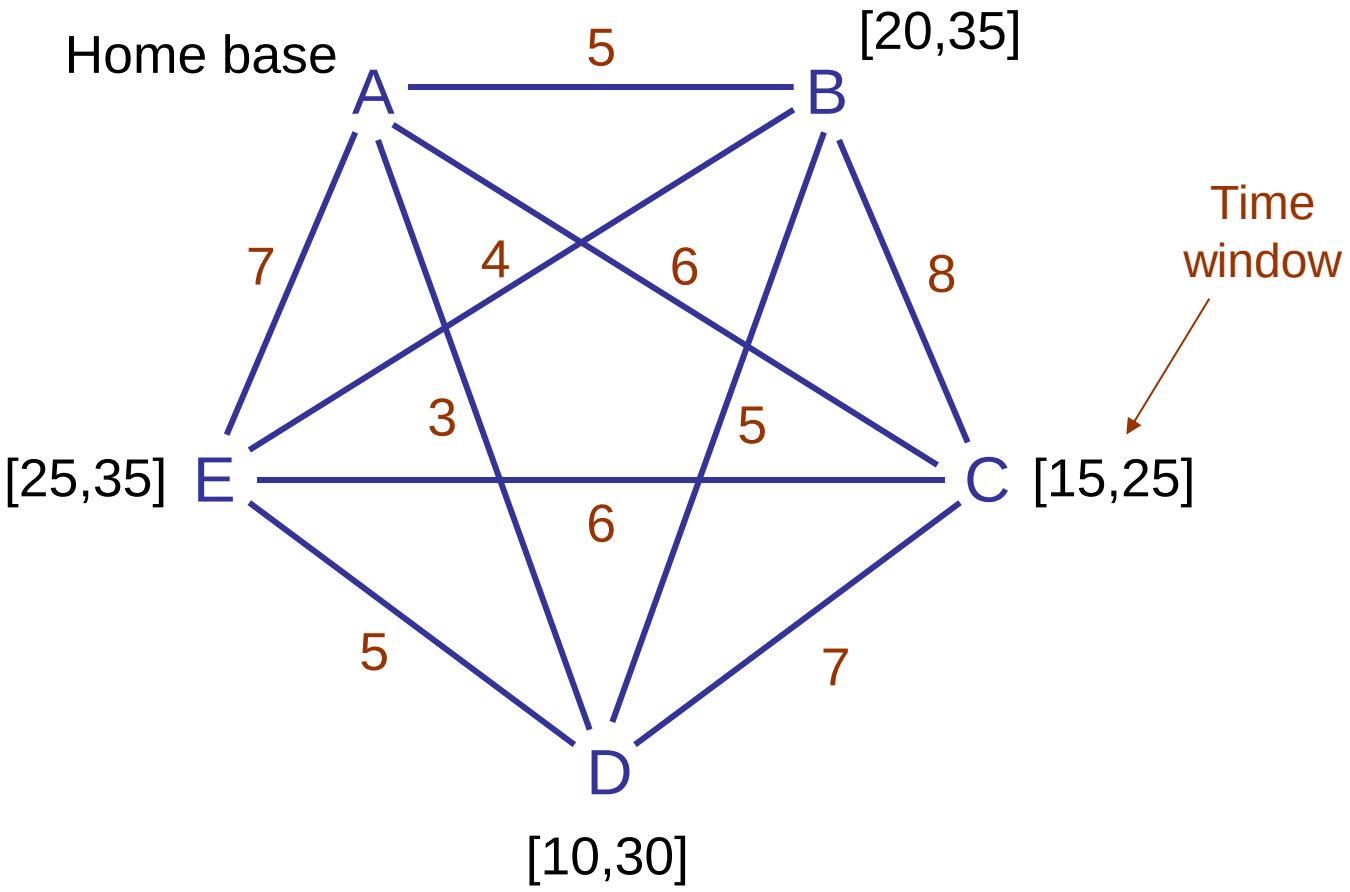
Constructive phase: select randomized greedy values until all variables fixed.

Local search phase: search over neighboring complete solutions

Stop local search. Process now starts over with new constructive phase.

An Example

TSP with Time Windows



Relaxation of TSP

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 .

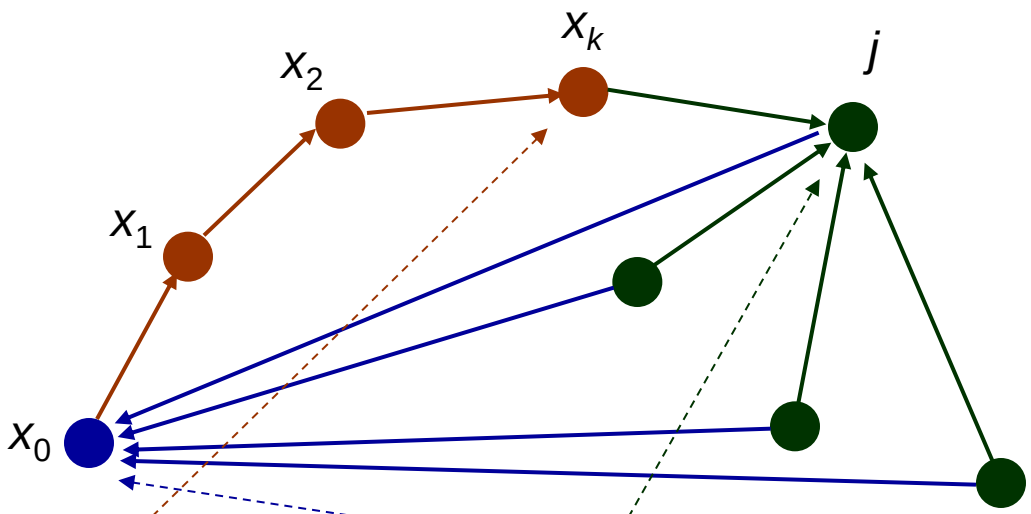
Then total travel time of completed route is bounded below by

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

Earliest time vehicle can leave customer k

Min time from customer j 's predecessor to j

Min time from last customer back to home



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

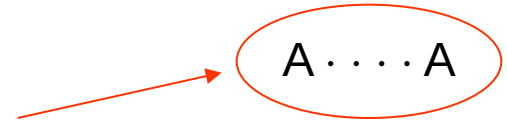
Earliest time vehicle can leave customer *k*

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

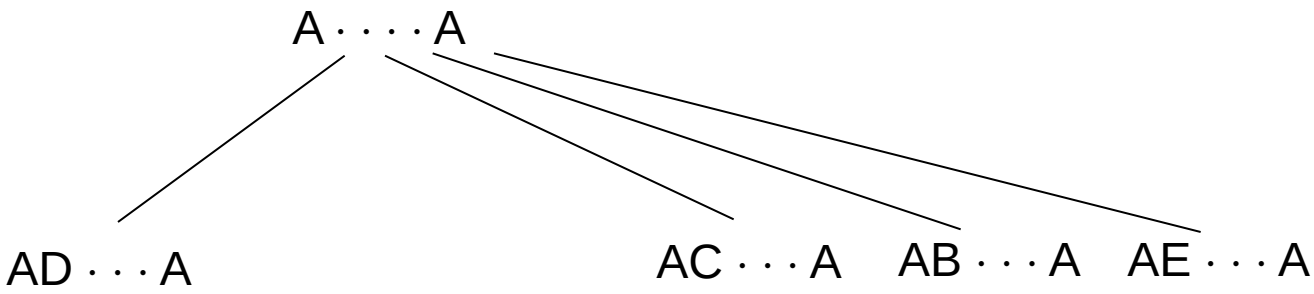
Min time from last customer back to home

Exhaustive Branch-and-Bound

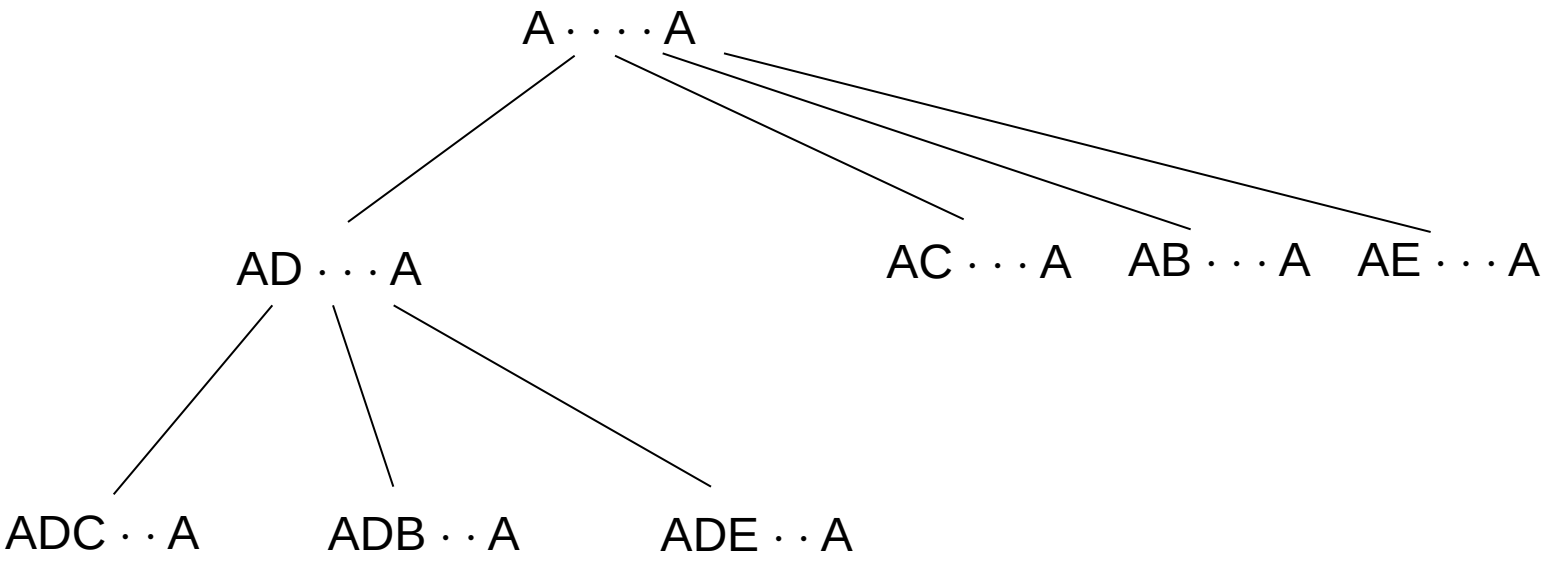
Sequence of customers visited



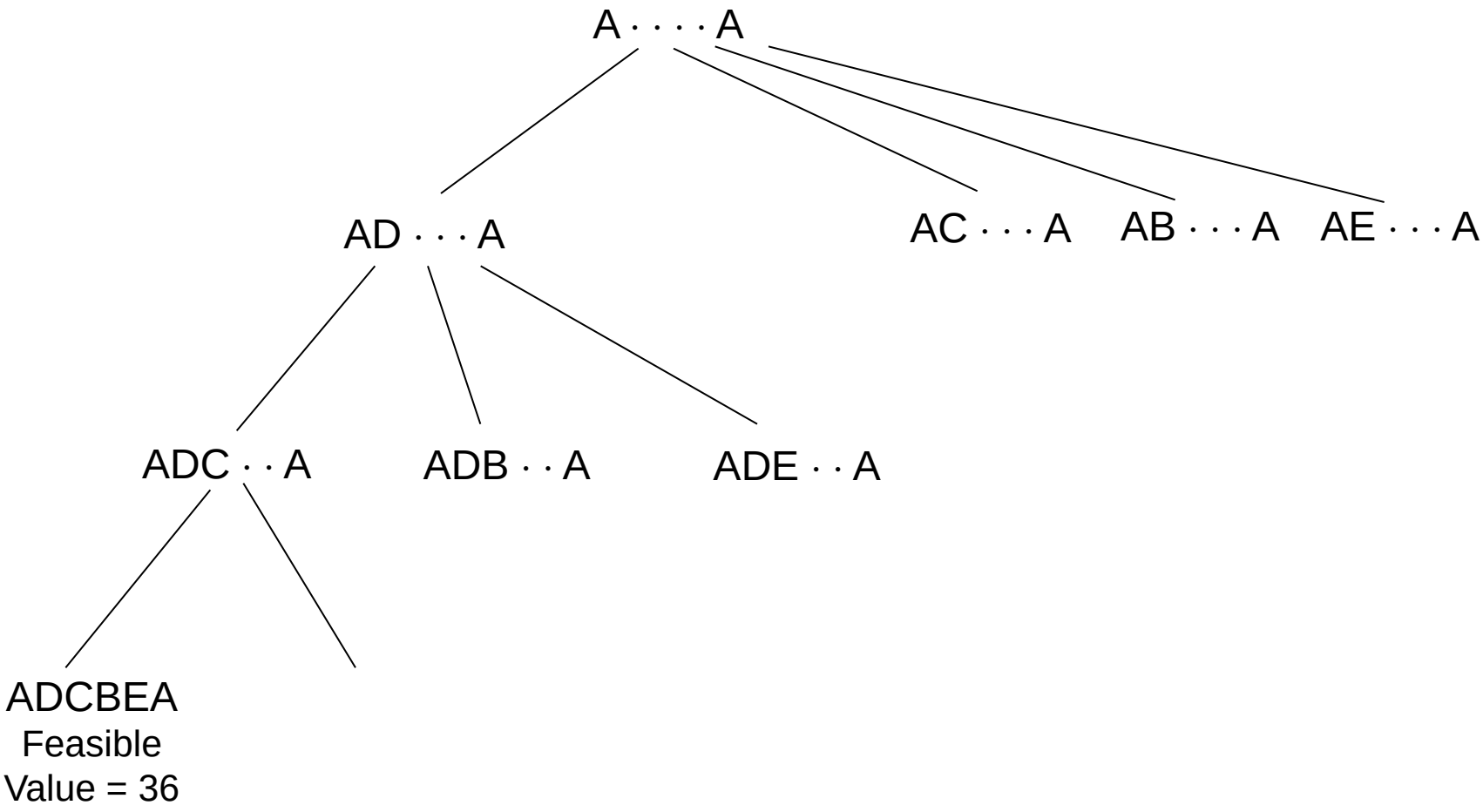
Exhaustive Branch-and-Bound



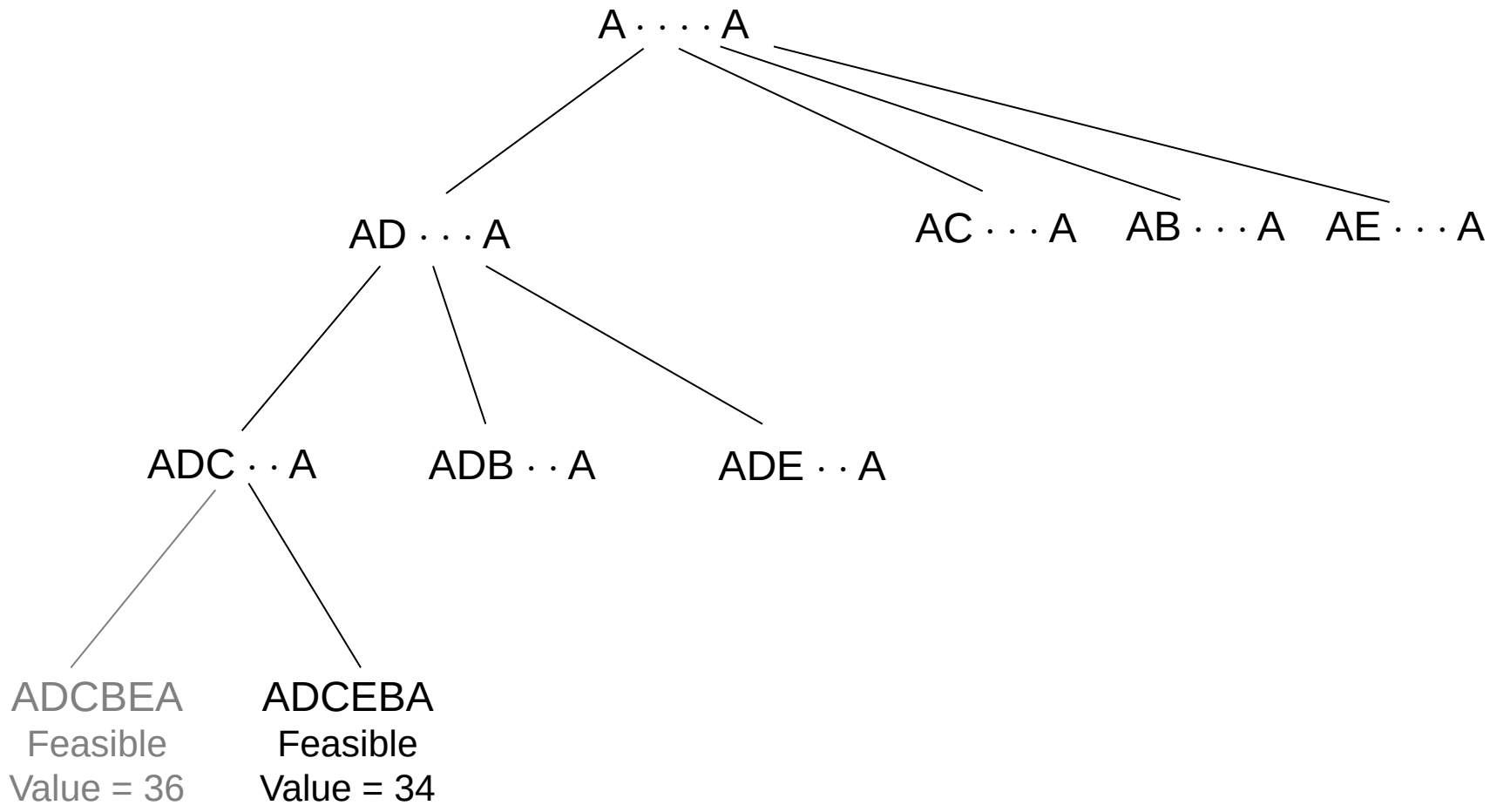
Exhaustive Branch-and-Bound



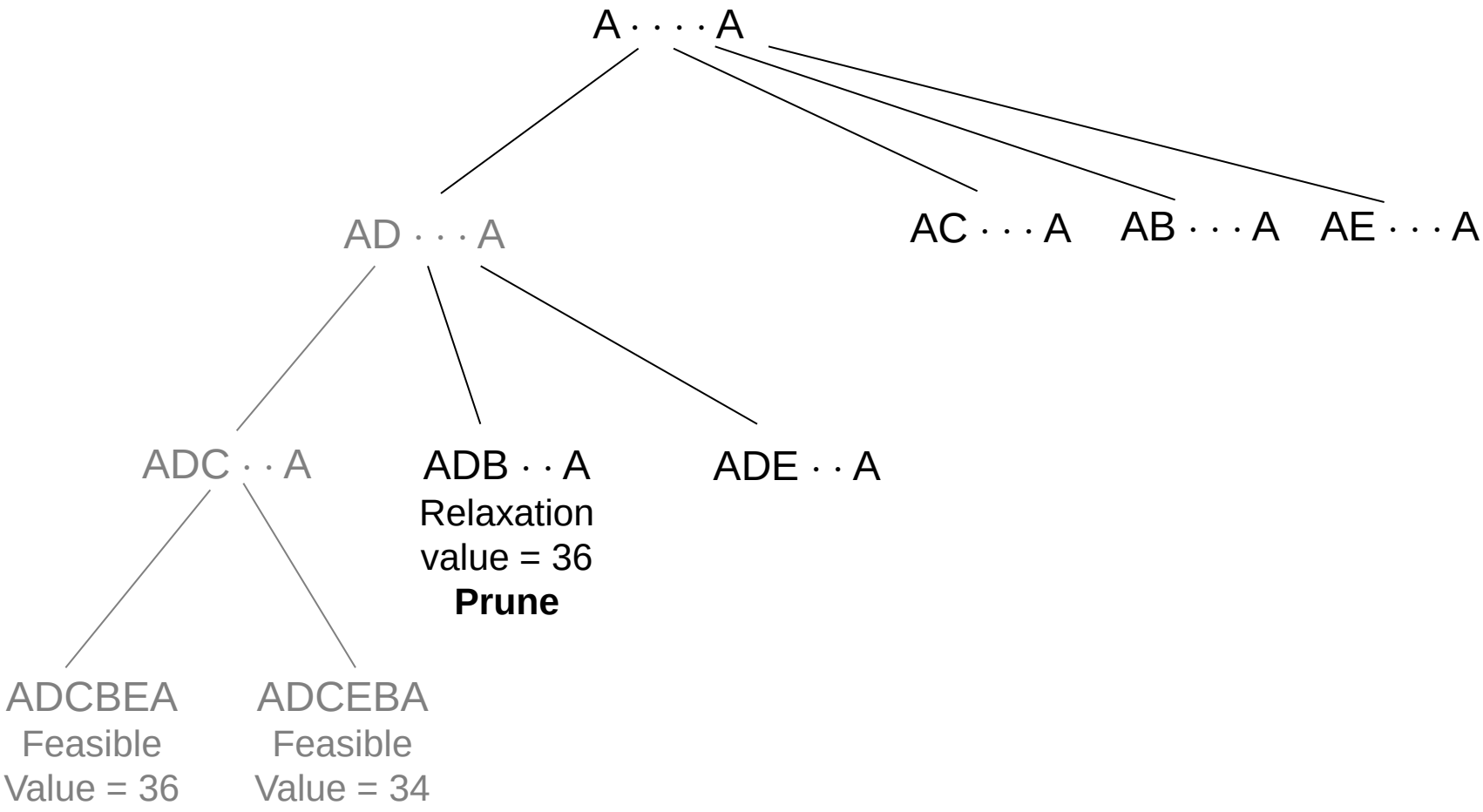
Exhaustive Branch-and-Bound



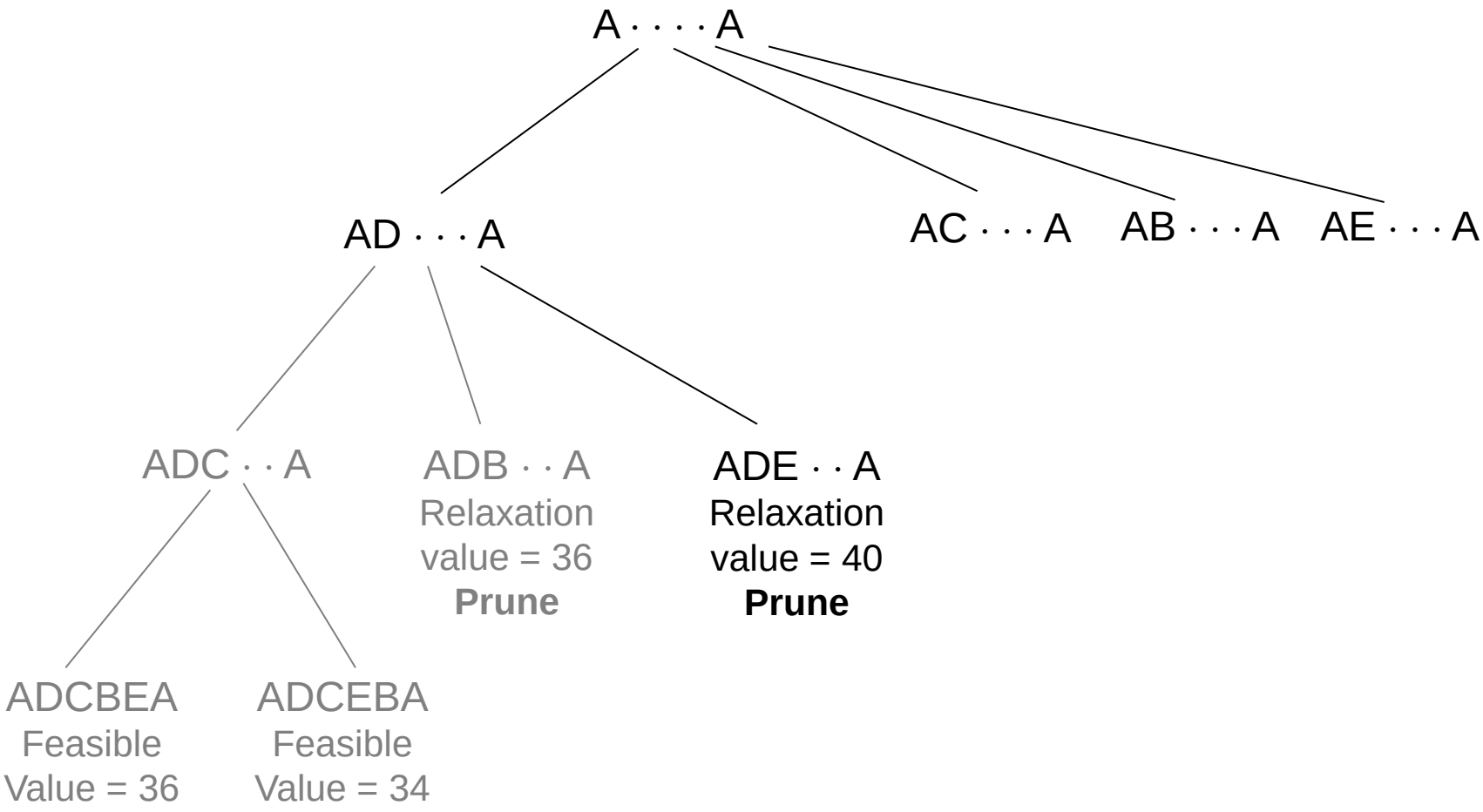
Exhaustive Branch-and-Bound



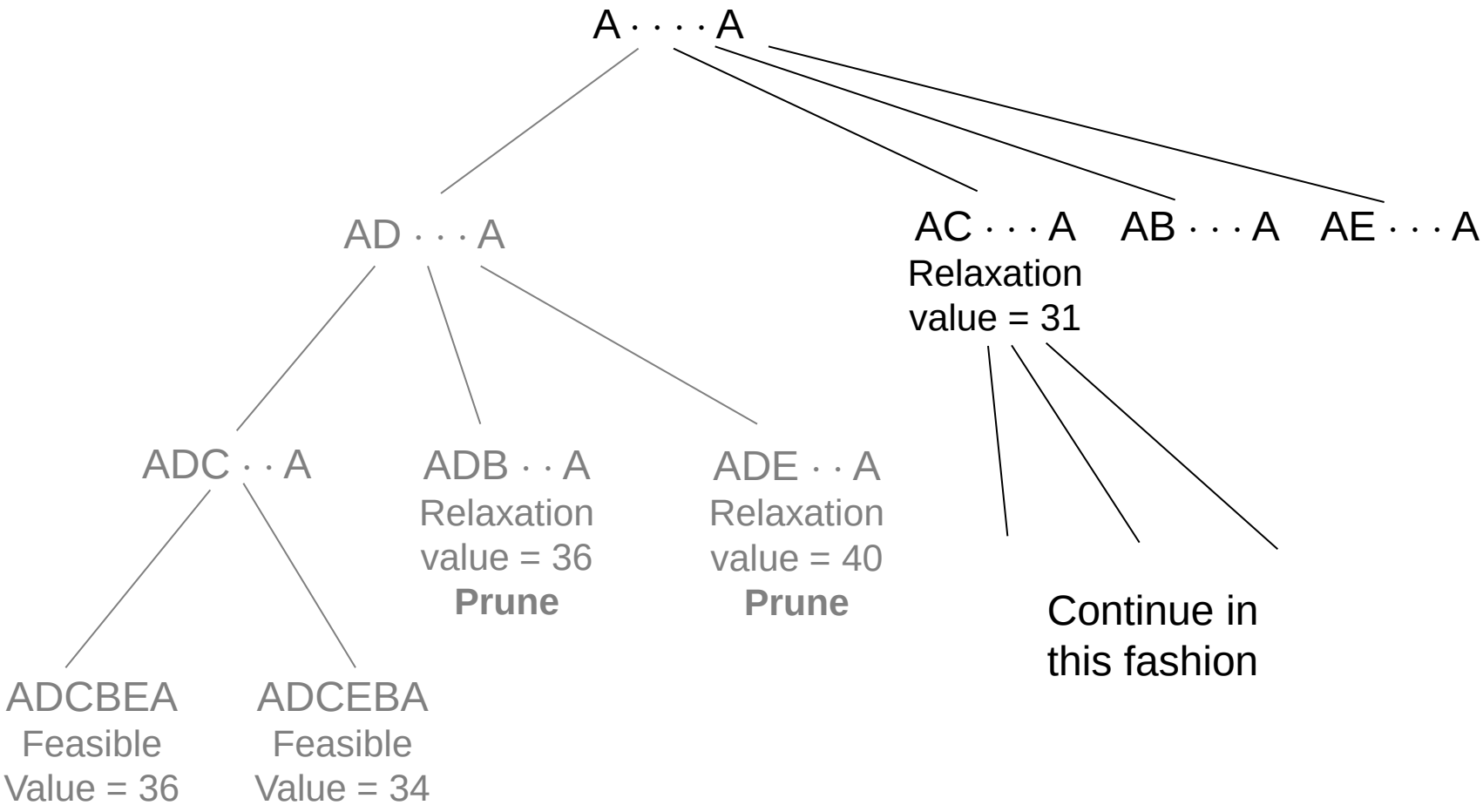
Exhaustive Branch-and-Bound



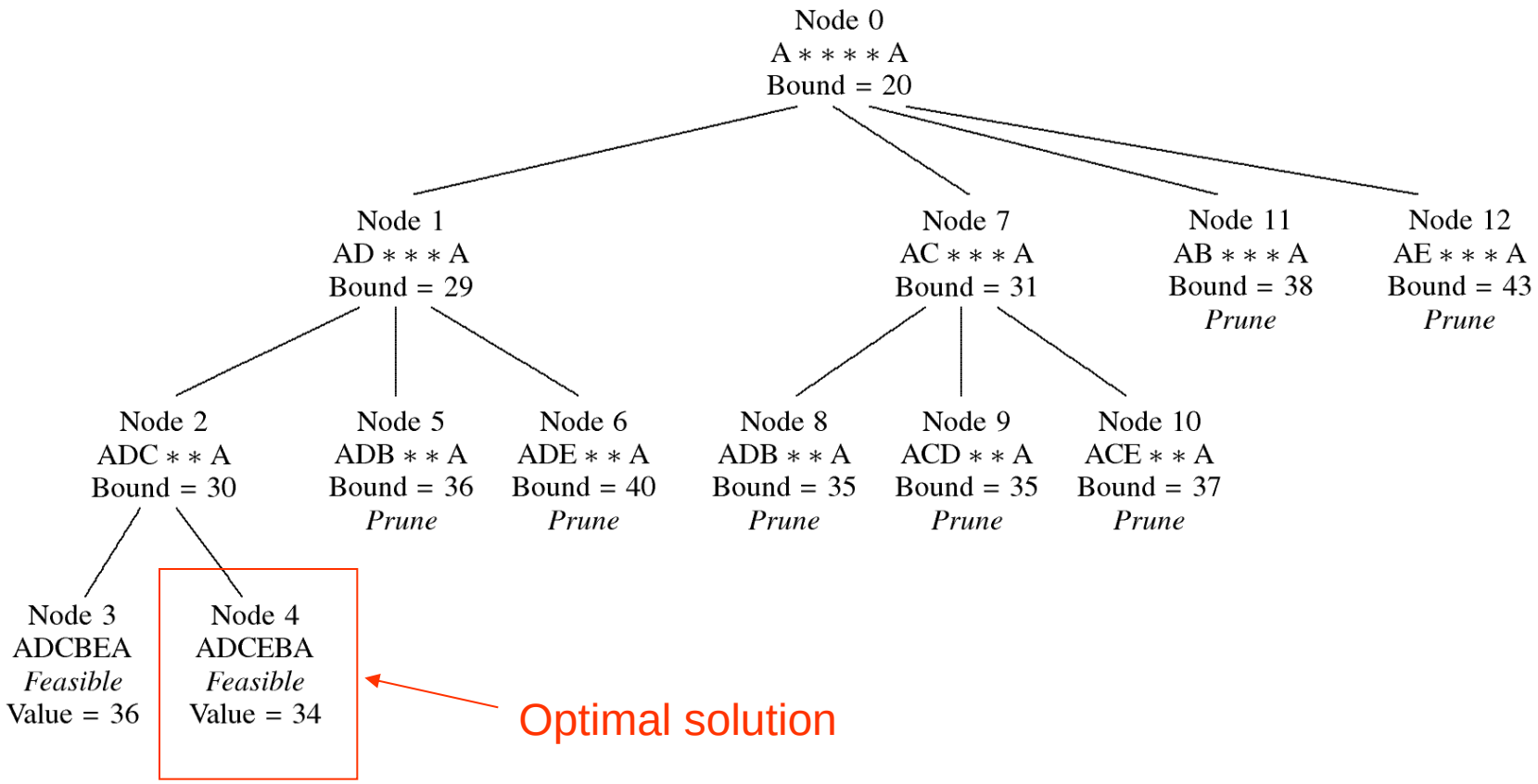
Exhaustive Branch-and-Bound



Exhaustive Branch-and-Bound

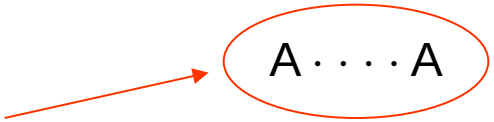


Exhaustive Branch-and-Bound



Generalized GRASP

Sequence of customers visited



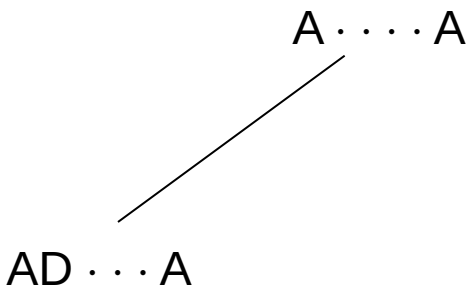
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Constructive phase



Visit customer than can be served earliest from A

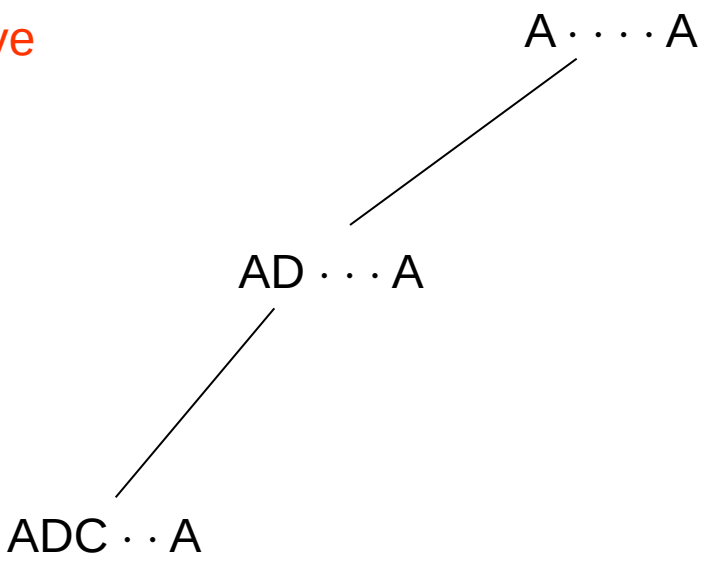
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Constructive phase



Next, visit customer than can be served earliest from D

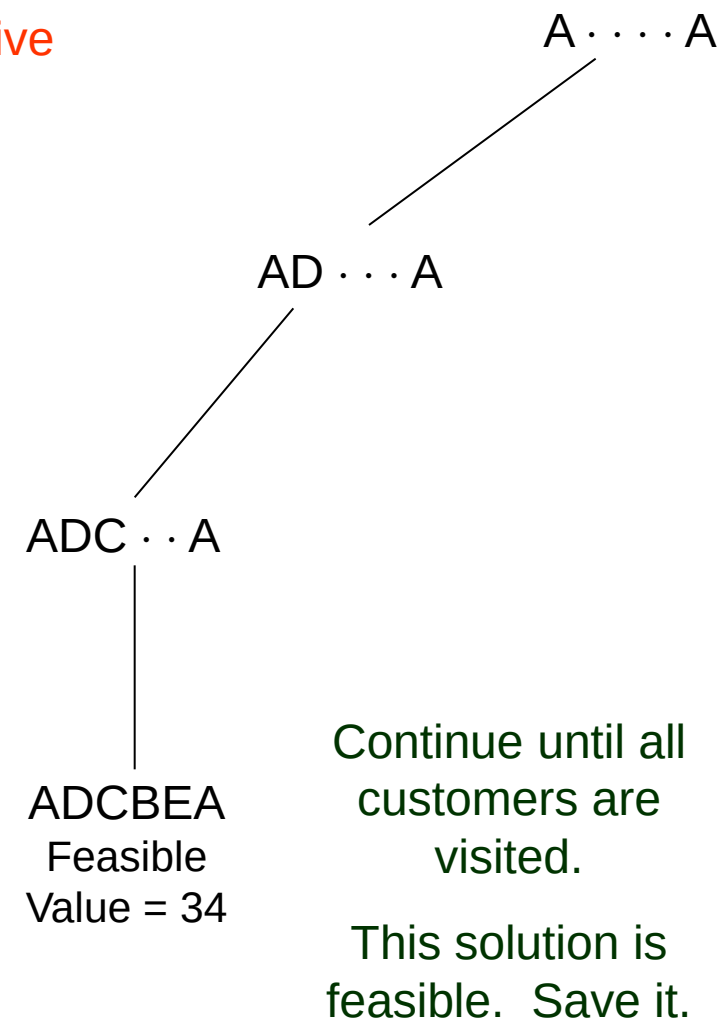
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Constructive phase



Continue until all customers are visited.

This solution is feasible. Save it.

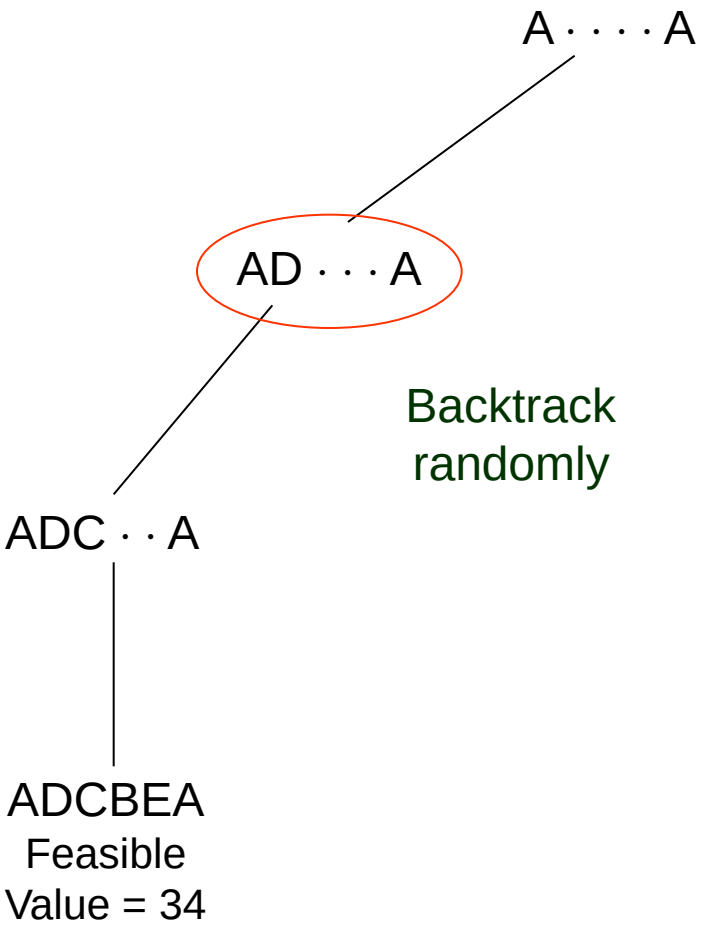
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Local search phase



Backtrack randomly

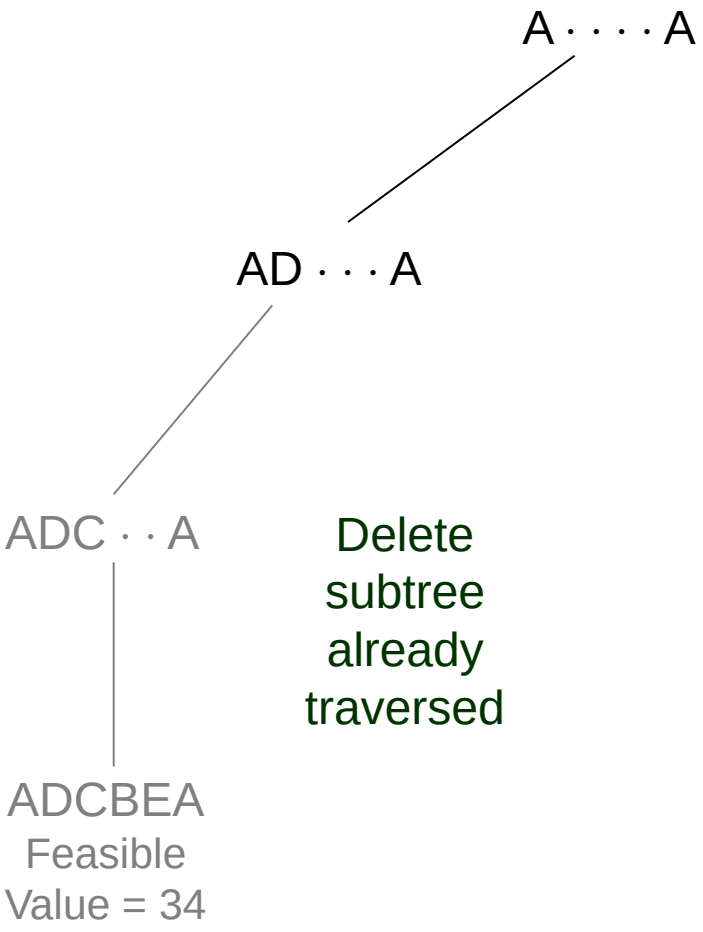
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Local search phase



Delete subtree already traversed

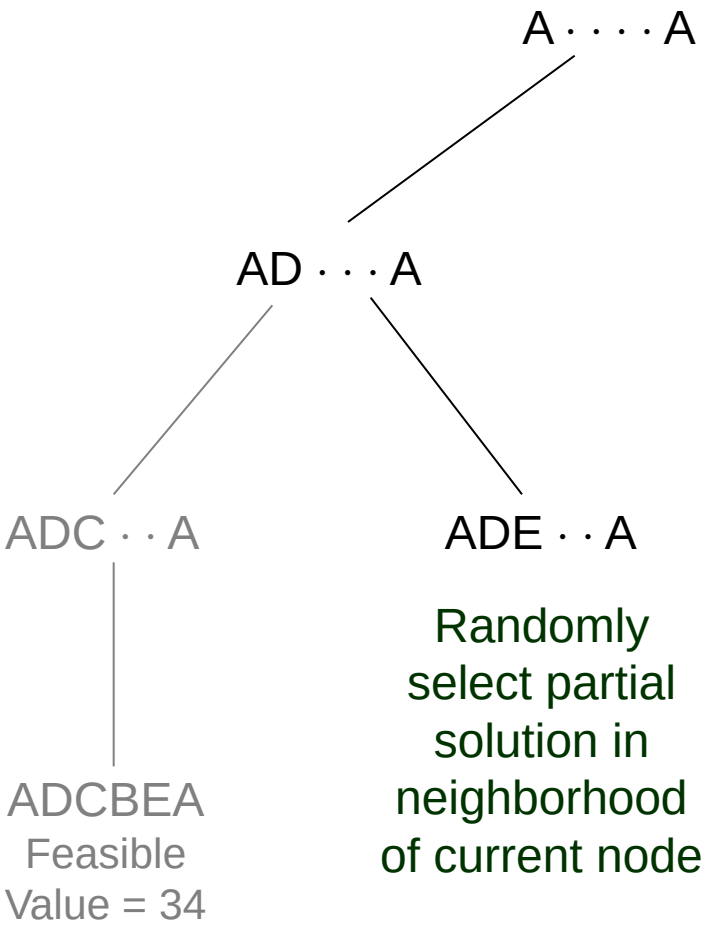
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Local search phase



Randomly select partial solution in neighborhood of current node

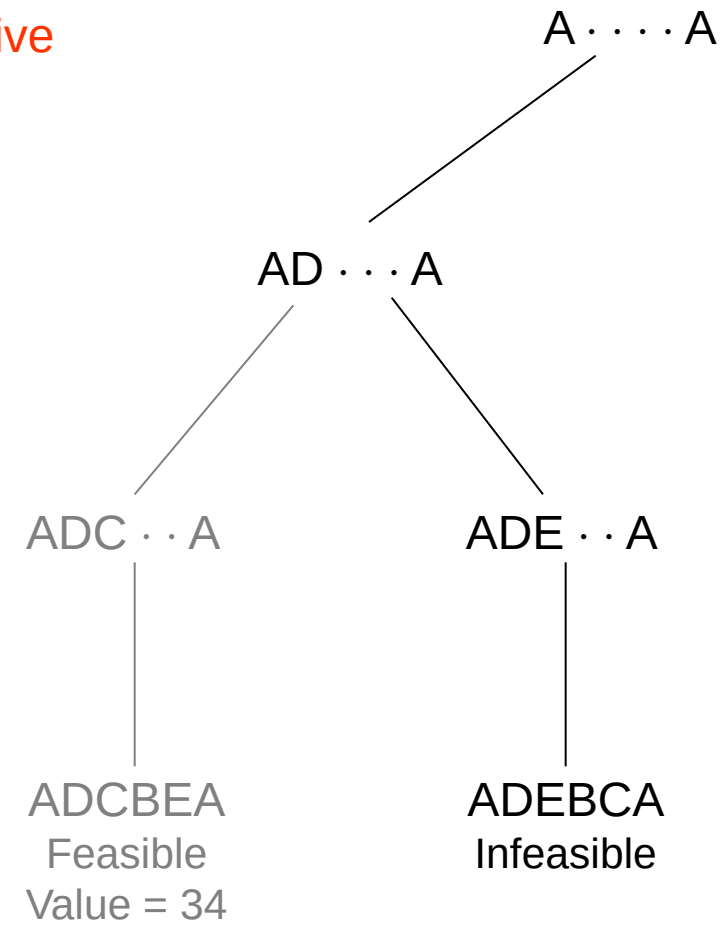
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

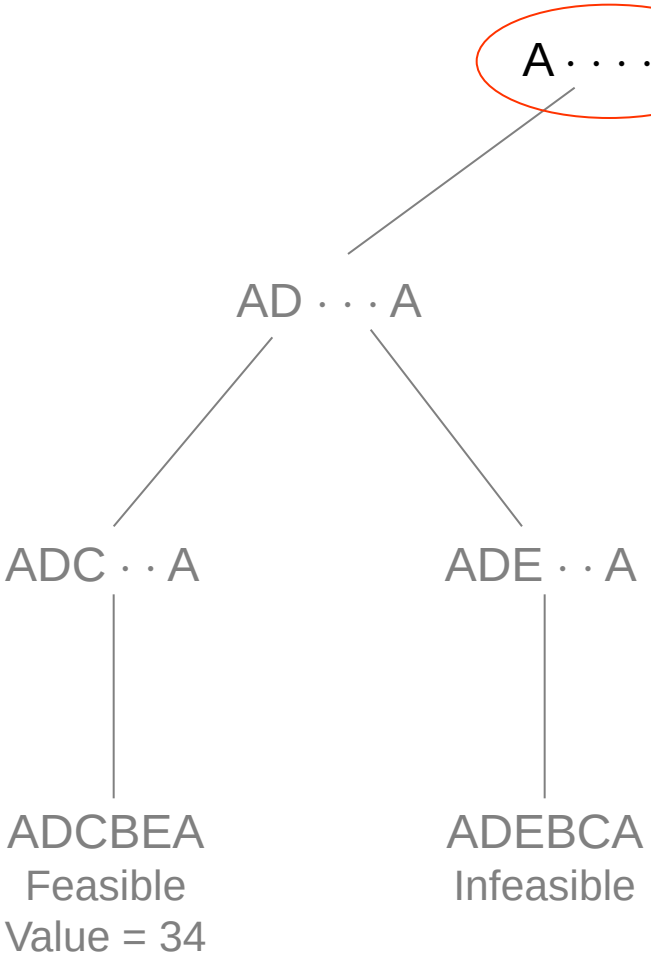
Constructive phase



Complete solution in greedy fashion

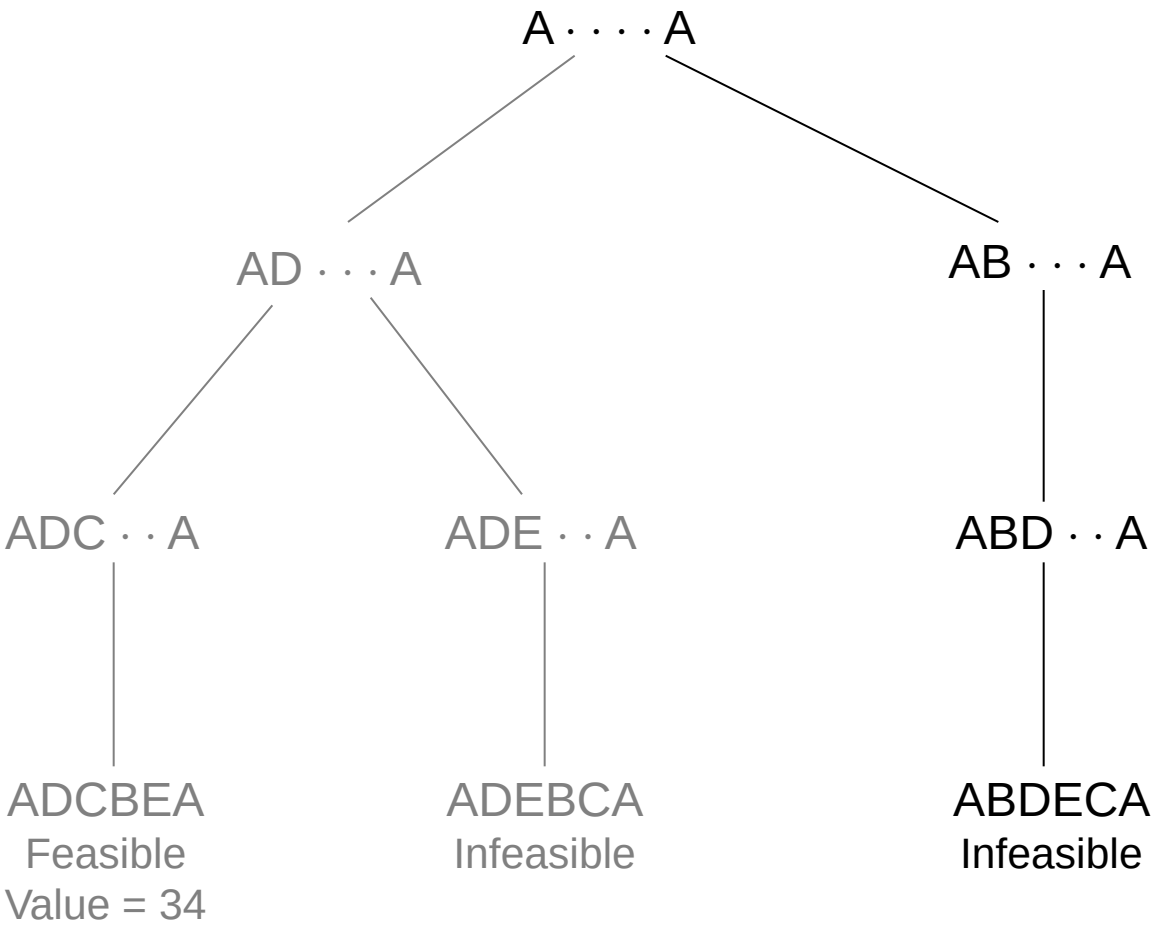
Generalized GRASP

Local search phase



Randomly backtrack

Generalized GRASP

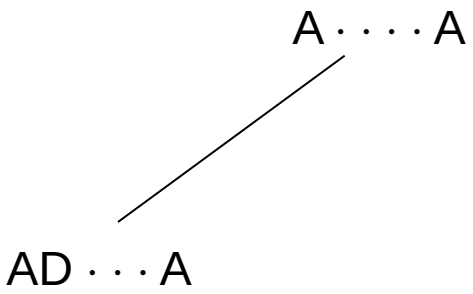


Continue in similar fashion

Exhaustive search algorithm (branching search) suggests a generalization of a heuristic algorithm (GRASP).

Generalized GRASP with relaxation

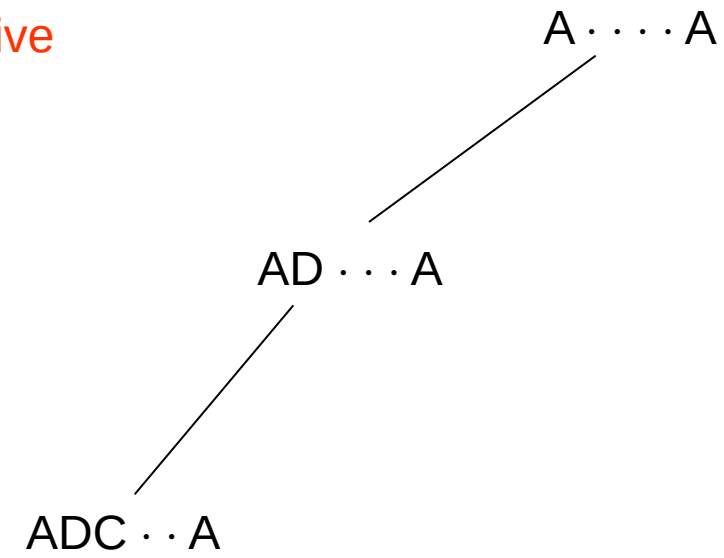
Constructive phase



Exhaustive search suggests an improvement on a heuristic algorithm: use relaxation bounds to reduce the search.

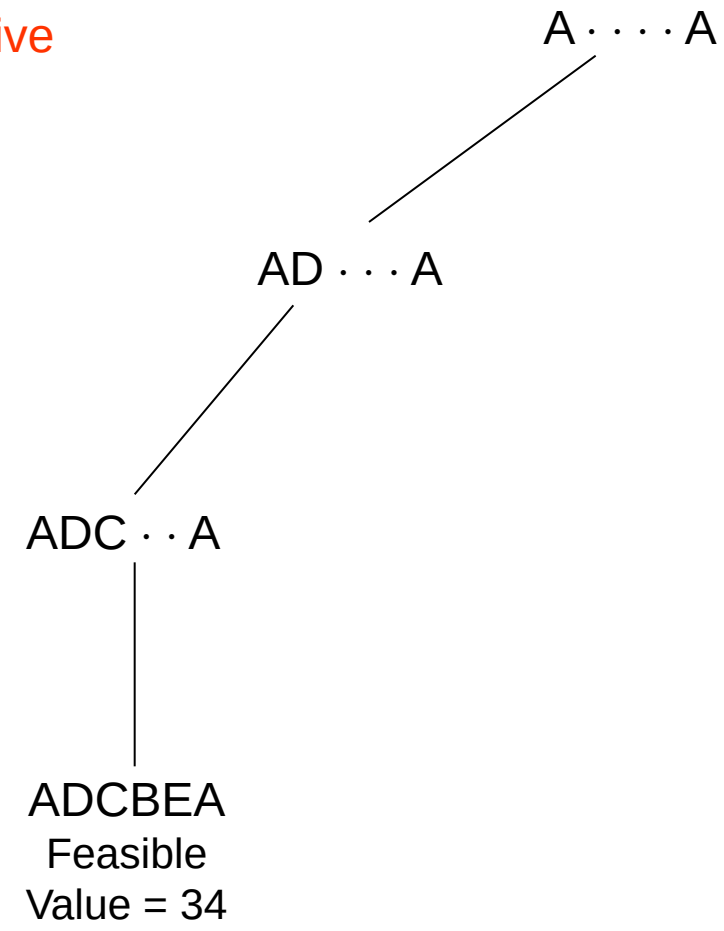
Generalized GRASP with relaxation

Constructive phase

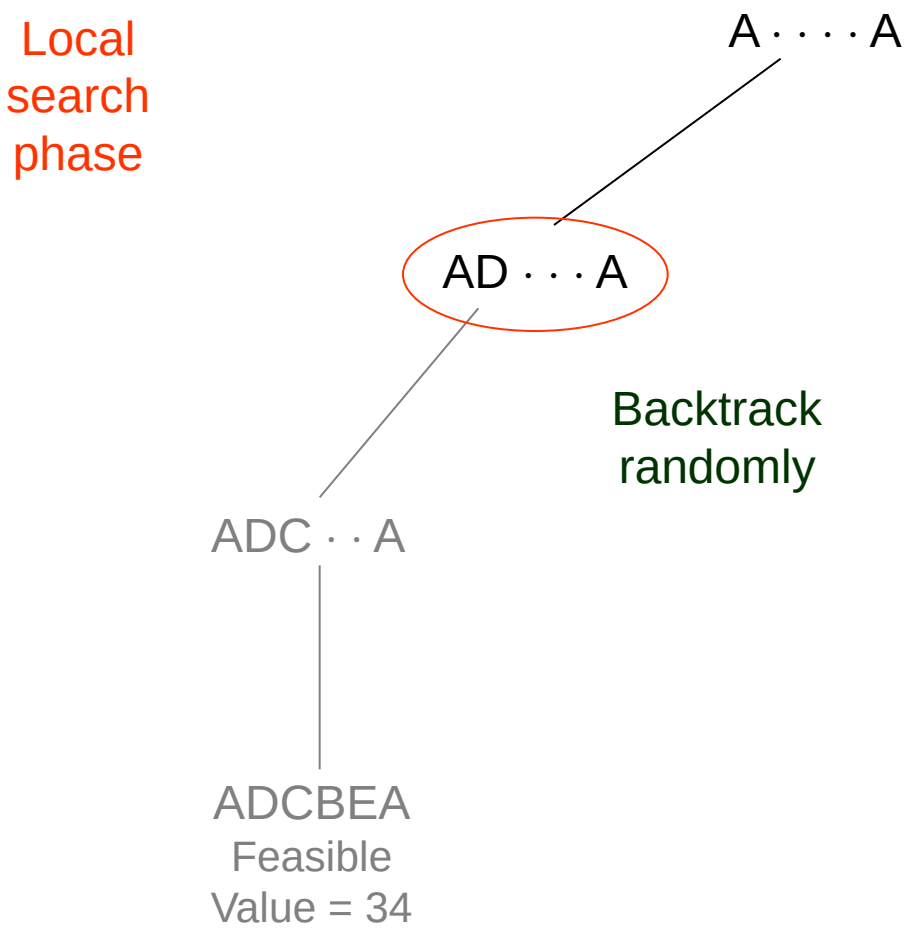


Generalized GRASP with relaxation

Constructive phase

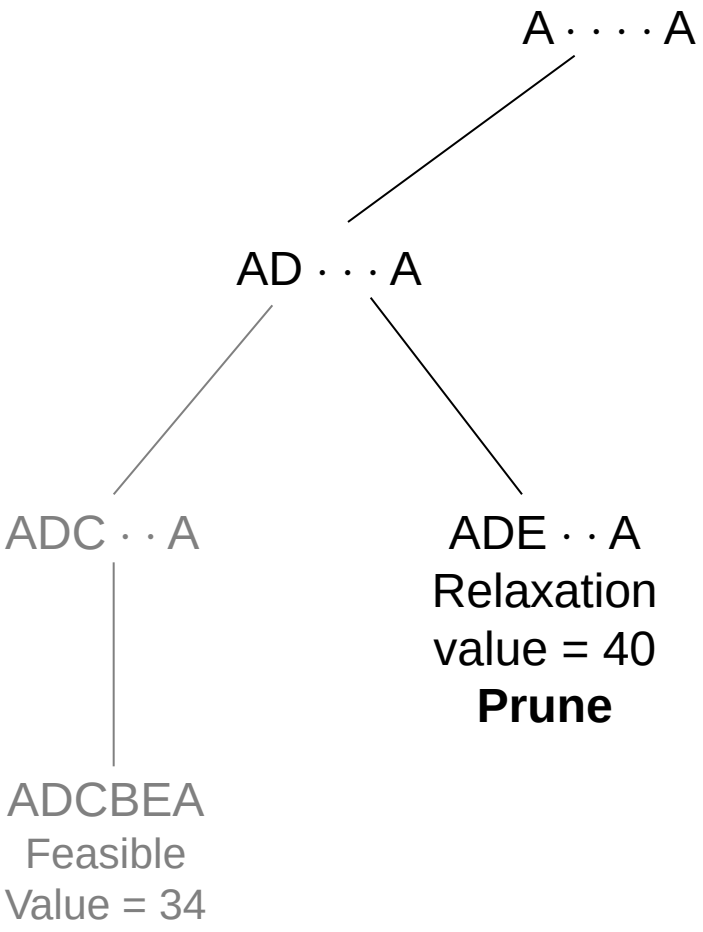


Generalized GRASP with relaxation



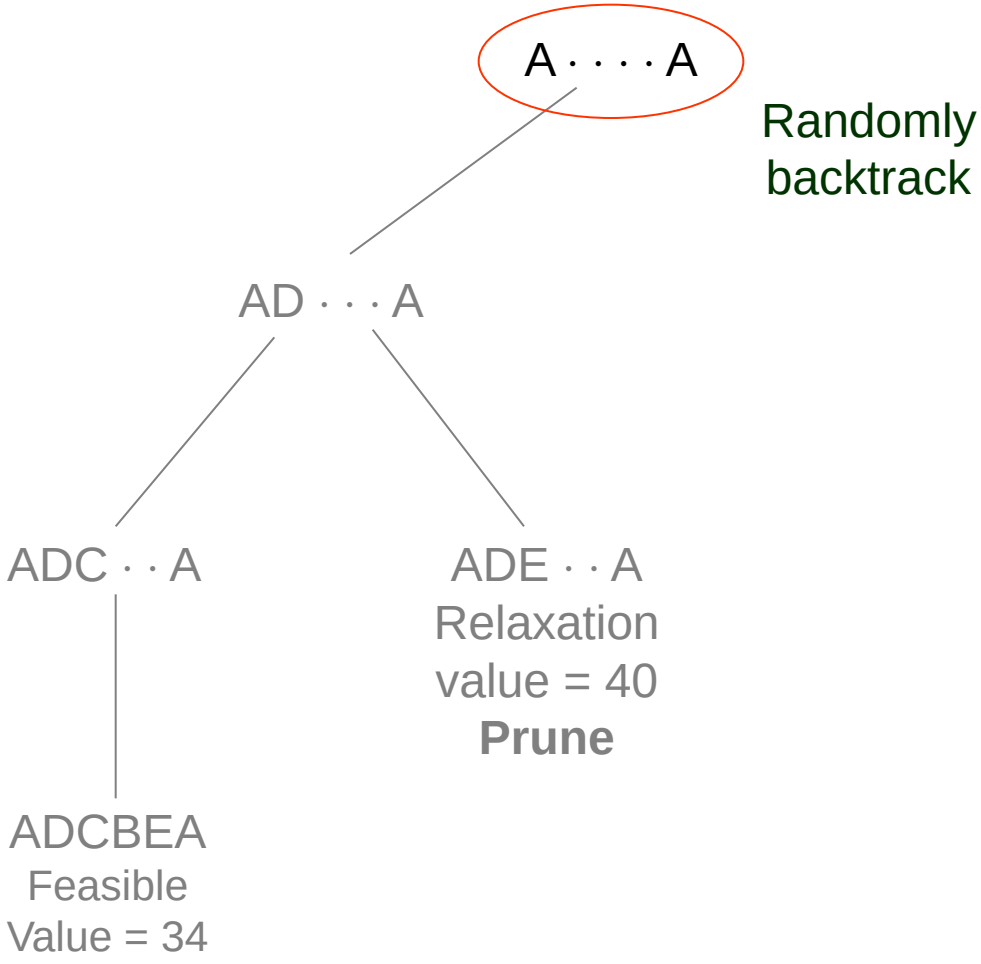
Generalized GRASP

Local search phase



Generalized GRASP

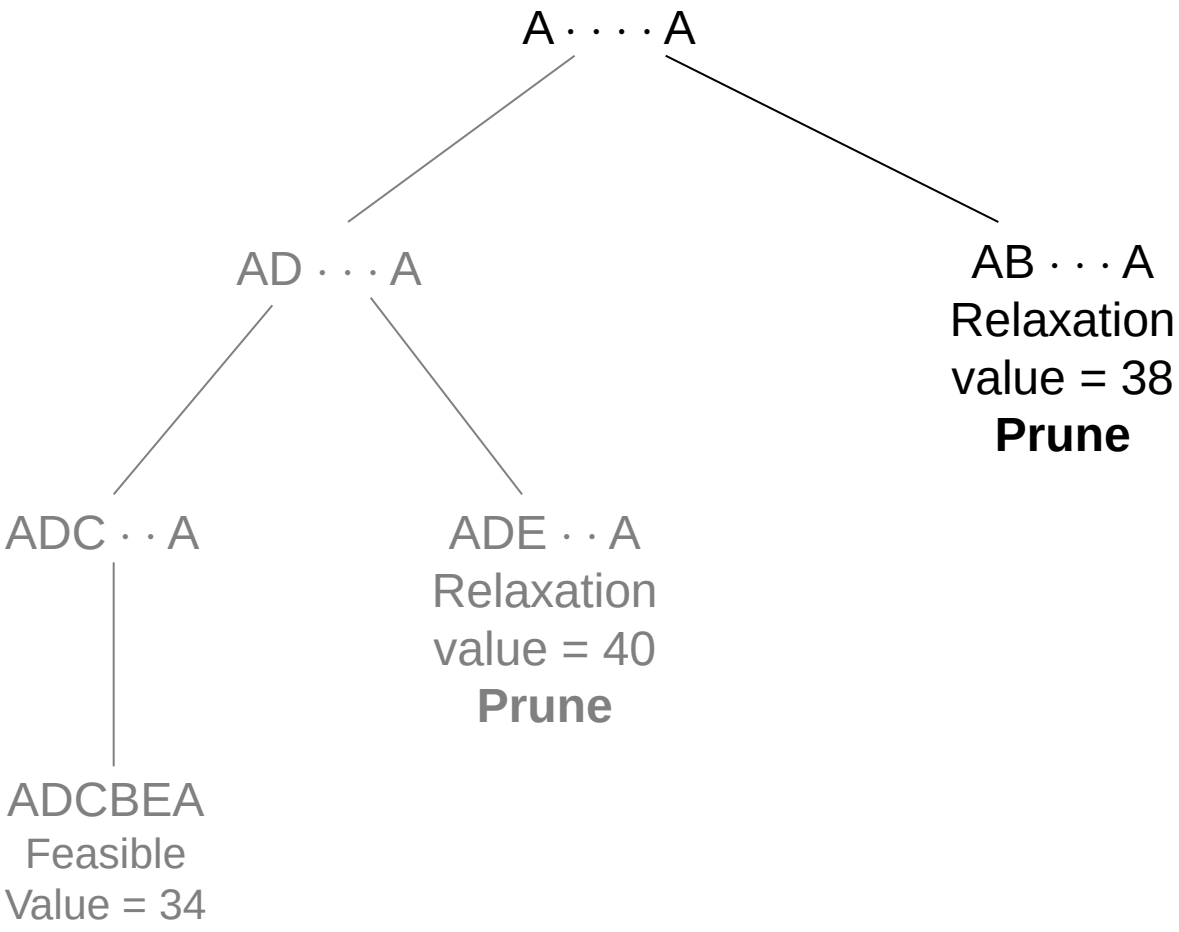
Local search phase



Randomly backtrack

Generalized GRASP

Local search phase



Heuristic Methods: Tabu Search

- Tabu search = constraint-directed local search
  - Based on **short-term** memory.
  - **Tabu list** serves as nogood set.
  - Find best solution in neighborhood of current solution that is not on tabu list.
  - Put it on a tabu list, drop oldest nogood.

Heuristic algorithm: Tabu search

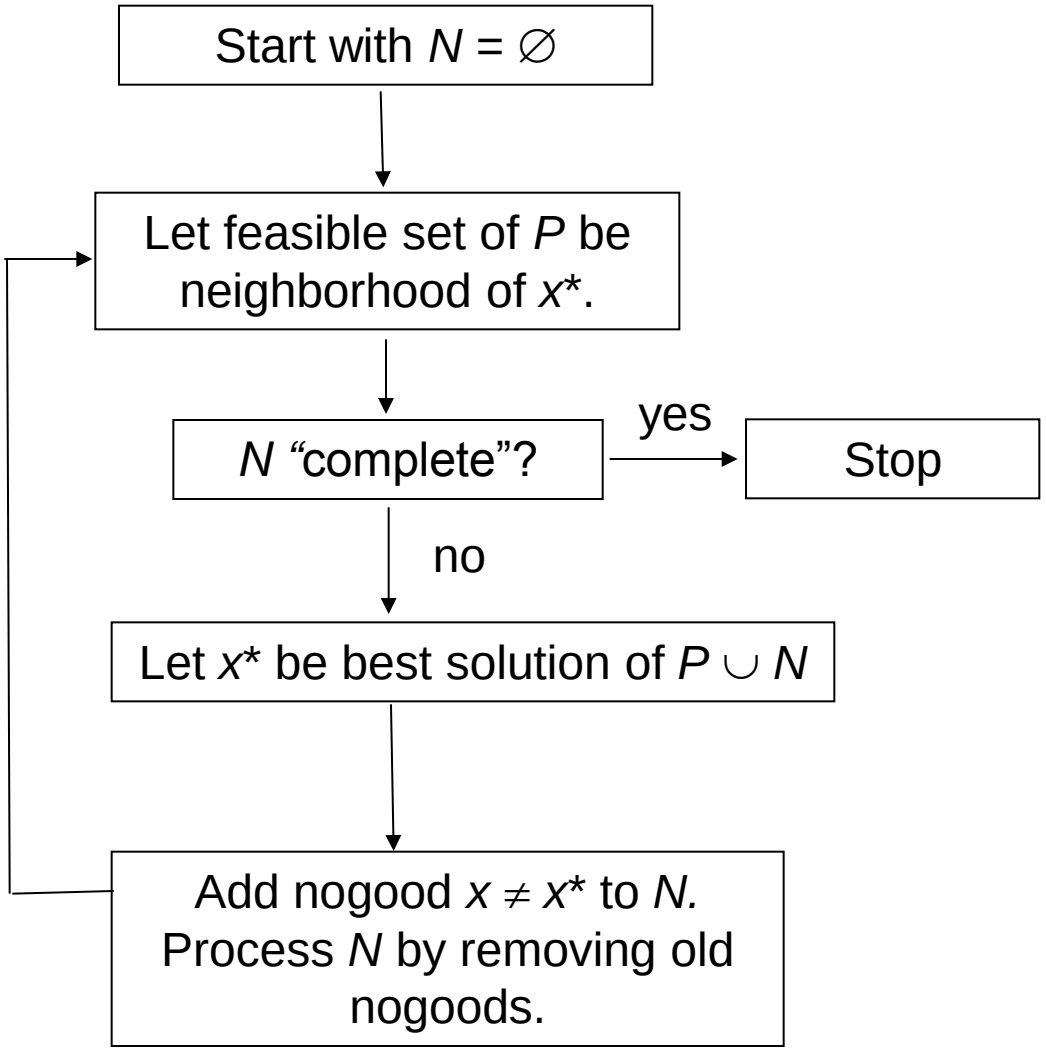
N is nogood set (tabu list)

In each iteration, search neighborhood for best solution not on tabu list.

N is "complete" when one has searched long enough.

P is problem restriction.

Its feasible set is neighborhood of current solution *x**



Heuristic algorithm: Tabu search

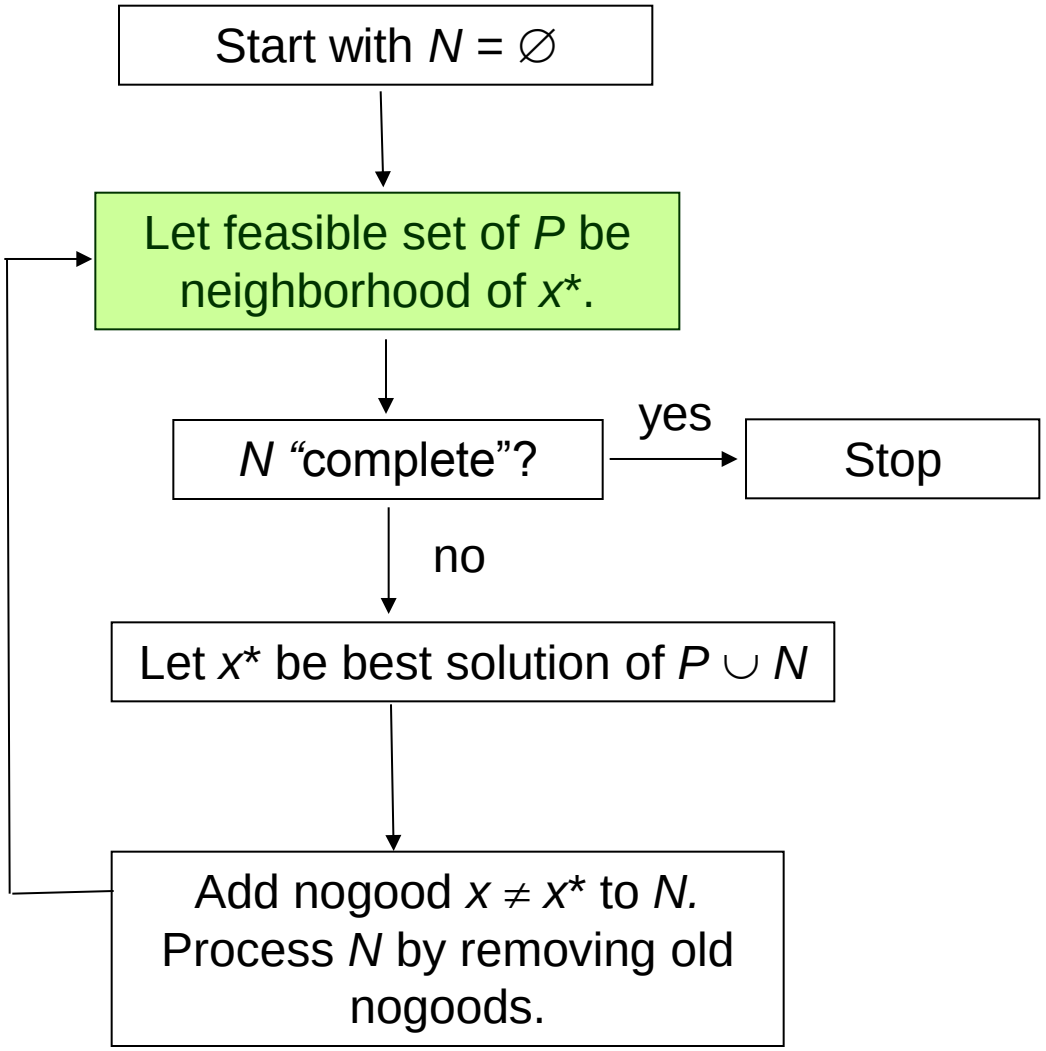
N is nogood set (tabu list)

In each iteration, search neighborhood for best solution not on tabu list.

N is "complete" when one has searched long enough.

P is problem restriction.

Its feasible set is neighborhood of current solution *x**



Heuristic algorithm: Tabu search

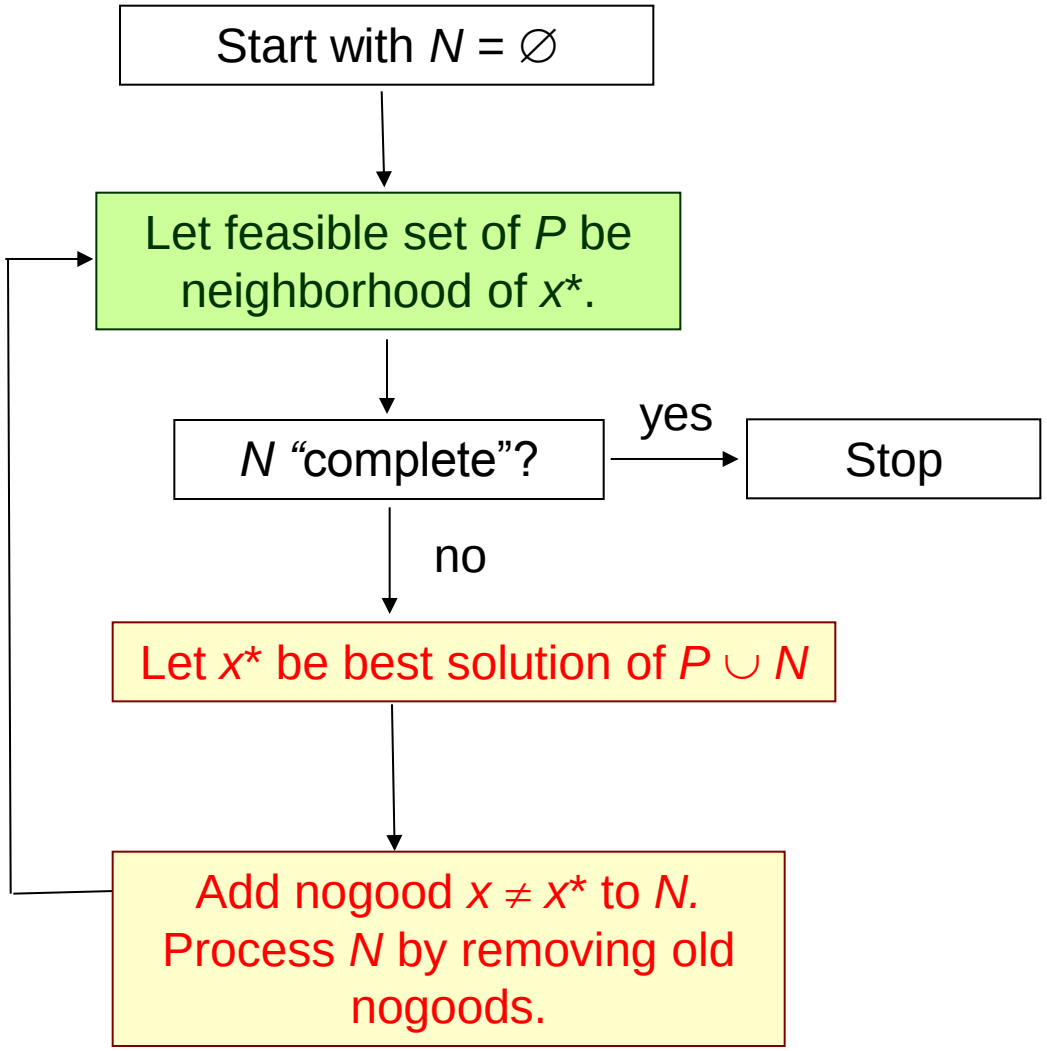
N is nogood set (tabu list)

In each iteration, search neighborhood for best solution not on tabu list.

N is "complete" when one has searched long enough.

P is problem restriction.

Its feasible set is neighborhood of current solution *x**



Heuristic Methods: Tabu Search

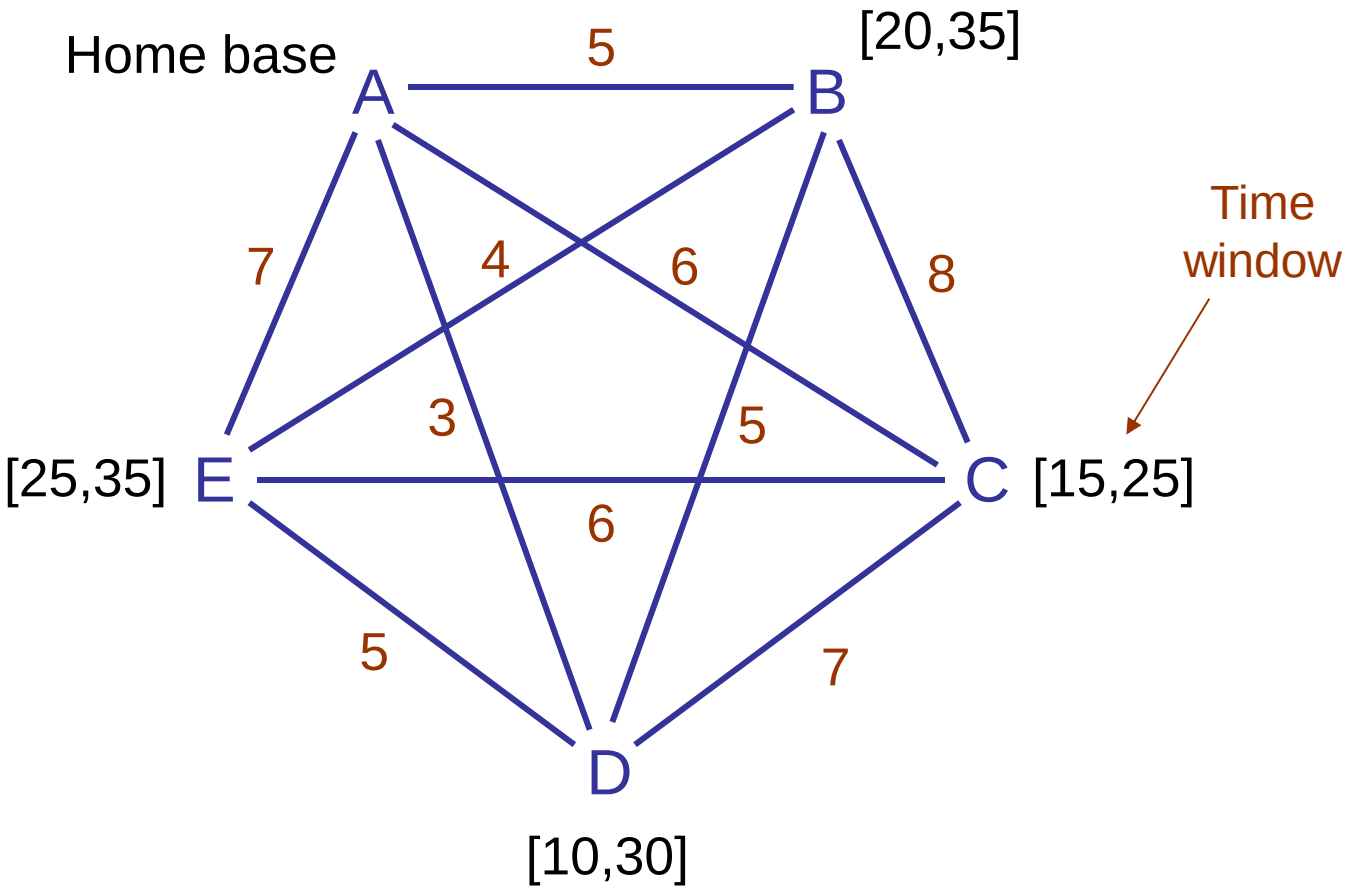
- Medium-term memory
  - Use solution elements (rather than solutions) as nogoods.
  - For example, edge removed from TSP tour.
  - Analogous to strong Benders cuts that exclude many solutions.
- Long-term memory
  - Keep track of any long-term improvement.
  - If none, use **diversification** (random perturbation).
  - Aspiration criteria, etc.

Heuristic Methods: Tabu Search

- To convert tabu search to an exact method…
  - Retain nogoods.
- To convert constraint-directed search to (sophisticated) tabu search…
  - Drop nogoods.

An Example

TSP with Time Windows



Exhaustive nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB			

Current nogoods

Excludes current solution by excluding any solution that begins ADCB

This is a special case of *partial-order dynamic backtracking*.

Exhaustive nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC			

Greedy solution of current nogood set:
Go to closest customer consistent with nogoods.

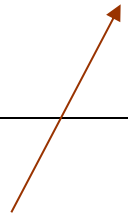
Current nogoods ADCB, ADCE rule out solutions beginning ADC.

Process nogood set by replacing ADCB, ADCE with *parallel resolvent* ADC.

Can solve the nogood set with greedy algorithm.

Exhaustive nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC	ADBEAC	infeasible	ADB
3	ADB,ADC			



Not only is ADBEAC infeasible, but no solution beginning ADB can be completed within time windows.

Exhaustive nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC	ADBEAC	infeasible	ADB
3	ADB,ADC	ADEBCA	infeasible	ADE
4	AD			

Process nogoods ADB, ADC, ADE
to obtain parallel resolvent AD

Exhaustive nogood-based search

Optimal solution.

At the end of the search, processed nogood set rules out all solutions (i.e, is infeasible).

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC	ADBEAC	infeasible	ADB
3	ADB,ADC	ADEBCA	infeasible	ADE
4	AD	ACDBEA	38	ACDB
5	ACDB,AD	ACDBEA	36	ACDE
6	ACD,AD	ACBEDA	infeasible	ACBE
7	ACD,ACBE,AD	ACBDEA	40	ACBD
8	ACB,AD	ACEBDA	infeasible	ACEB
9	ACB,ACEB,AD	ACEDBA	40	ACED
10	AC,AD	ABDECA	infeasible	ABD,ABC
11	ABD,ABC,AC,AD	ABEDCA	infeasible	ABE
12	AB,AC,AD	AEBCDA	infeasible	AEB,AEC
13	AB,AC,AD,AEB,AEC	AEDBCA	infeasible	AEB,AEC
14	A			

Heuristic nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC			

Start as before.

Remove old nogoods from nogood set.

Generate stronger nogoods by ruling out subsequences other than those starting with A.

This requires more intensive processing (full resolution), which is possible because nogood set is small.

Heuristic nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC	ADBECA	infeasible	ADB,EC
3	ABEC, ADB, ADC			

Process nogood set:
List all subsequences beginning with A
that are ruled out by current nogoods.

This requires a full resolution algorithm.

Heuristic nogood-based search

Iter.	$relax(P) \cup N$	Solution of N	Sol. value	New nogoods
0		ADCBEA	36	ADCB
1	ADCB	ADCEBA	34	ADCE
2	ADC	ADBECA	infeasible	ADB,EC
3	ABEC, ADB, ADC	ADEBCA	infeasible	ADE, EB
4	ABEC, ACEB, AD, AEB	ACDBEA	38	ACBD
⋮				

Continue in this fashion, but start dropping old nogoods.

Adjust length of nogood list to avoid cycling, as in tabu search.

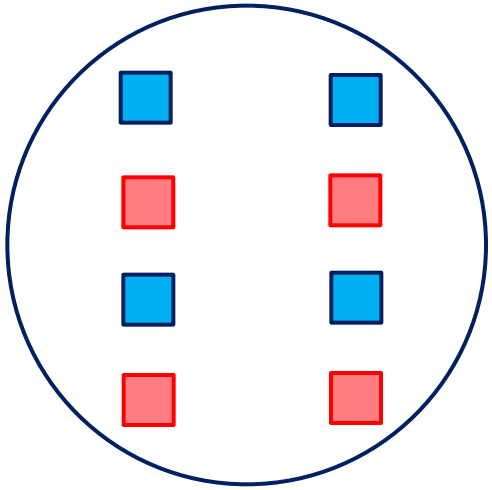
Stopping point is arbitrary.

So exhaustive nogood-based search suggests a more sophisticated variation of tabu search.

Heuristic Methods: Genetic Algorithms

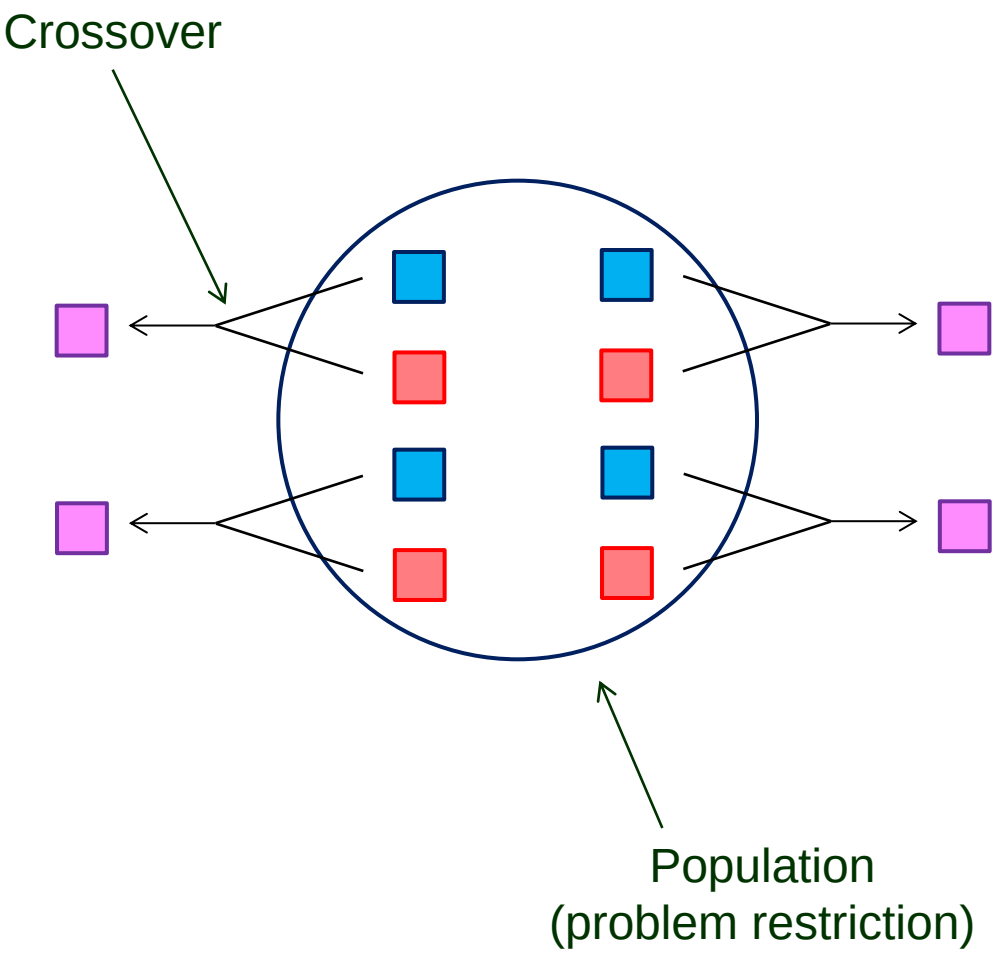
- Survival of the fittest.
  - Some solutions in current **population** are mated with **crossover** operation to produce new solutions.
  - Other solutions are **mutated**.
  - Weaker solutions deleted.
- This is a search over problem restrictions.
  - Population is feasible set of a problem restriction.

Heuristic Methods: Genetic Algorithms

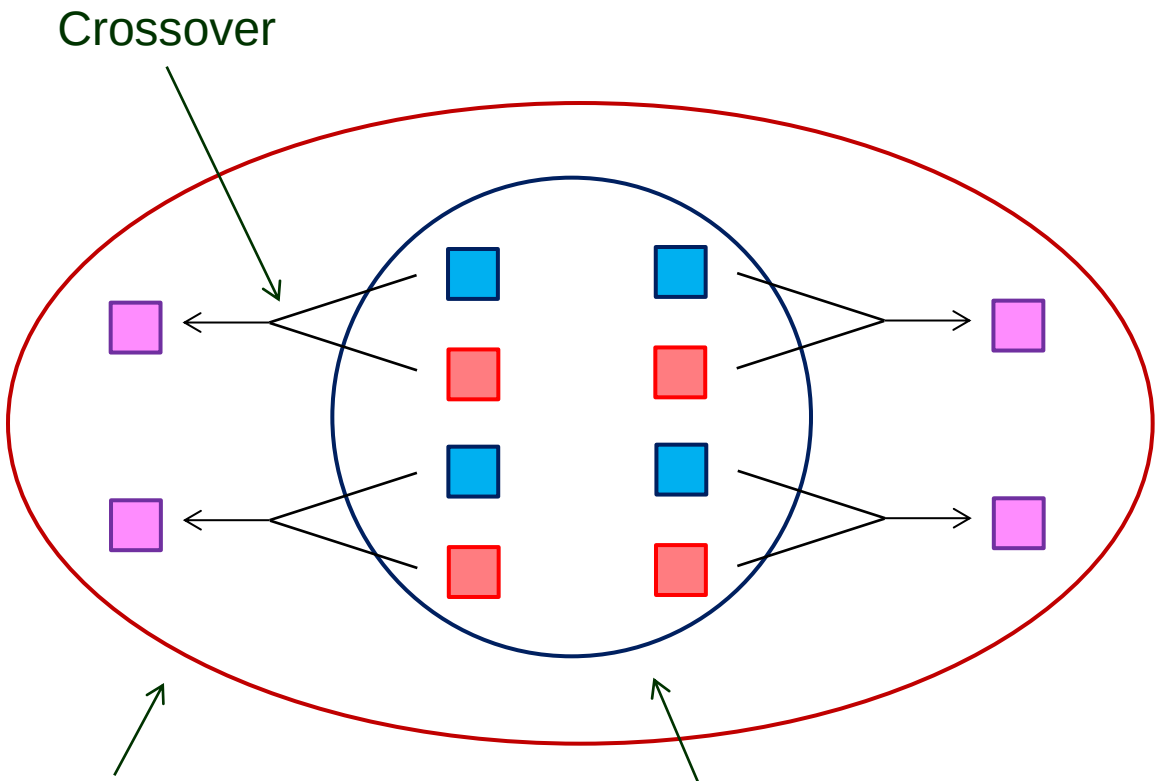


Population
(problem restriction)

Heuristic Methods: Genetic Algorithms



Heuristic Methods: Genetic Algorithms



"Solution"
of relaxation
(identifying
the fittest)
determines next
restriction
(population)

Heuristic Methods: Genetic Algorithms

- No obvious role for relaxation bounding.
  - Will “solve” relaxation of all restrictions in any case.

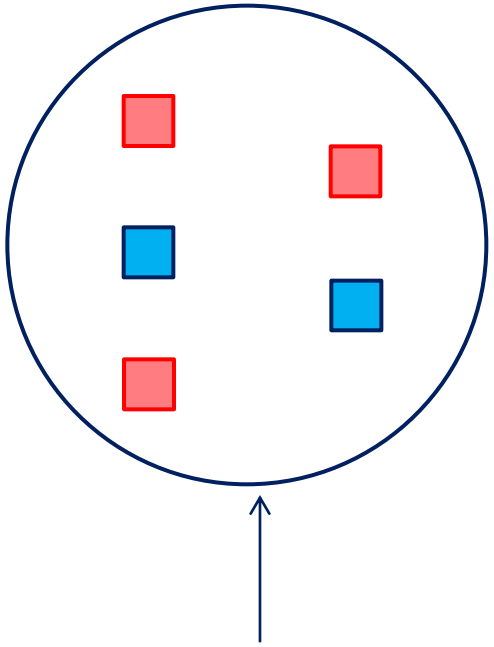
Heuristic Methods: Genetic Algorithms

- No obvious role for relaxation bounding.
  - Will “solve” relaxation of all restrictions in any case.
- **Deductive** inference dual of restriction uninteresting.
  - “Solve” restriction = examine all solutions to find best ones.
- **Inductive** inference dual may be useful.
  - Infer from current population solution characteristics that predict quality.
    - Perhaps using regression analysis.
  - Derive **probable** nogood bound $v \geq \mathrm{LB}(x)$.
  - Use nogoods to guide choice of parents.
    - Or even to design crossover.

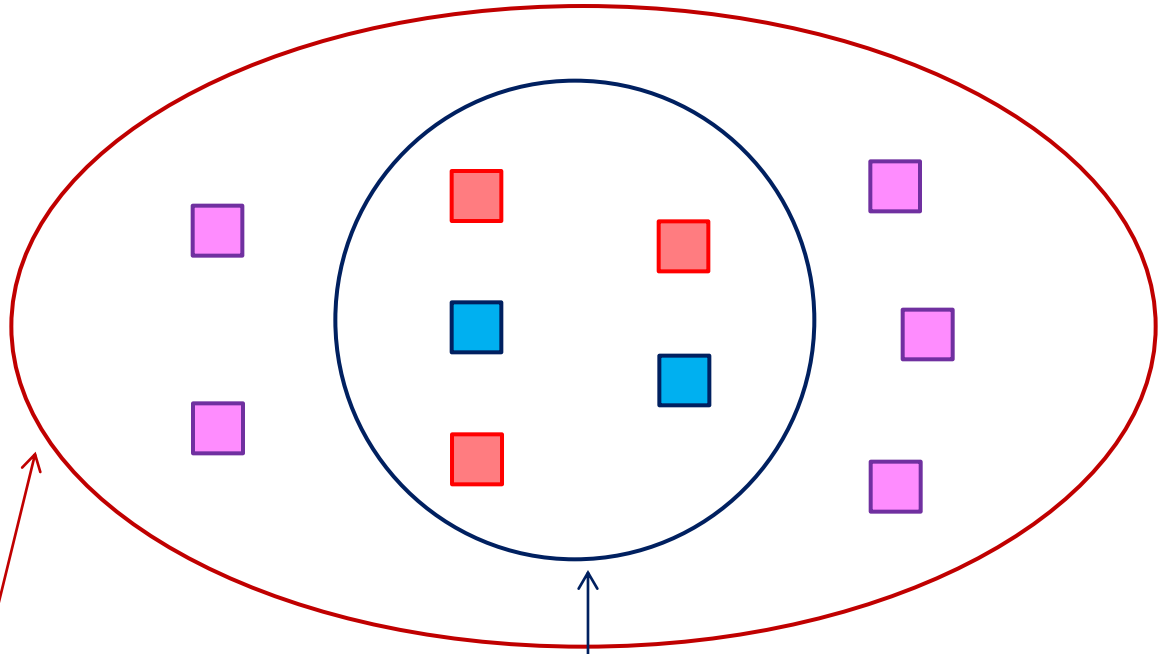
Heuristic Methods: Genetic Algorithms

- Interpretation as local search:
  - Current population is a single “solution.”
  - Its value is fitness of gene pool.
  - Neighbors are subset of population enlarged by offspring.
  - At termination, choose best solution in current population.

Heuristic Methods: Genetic Algorithms



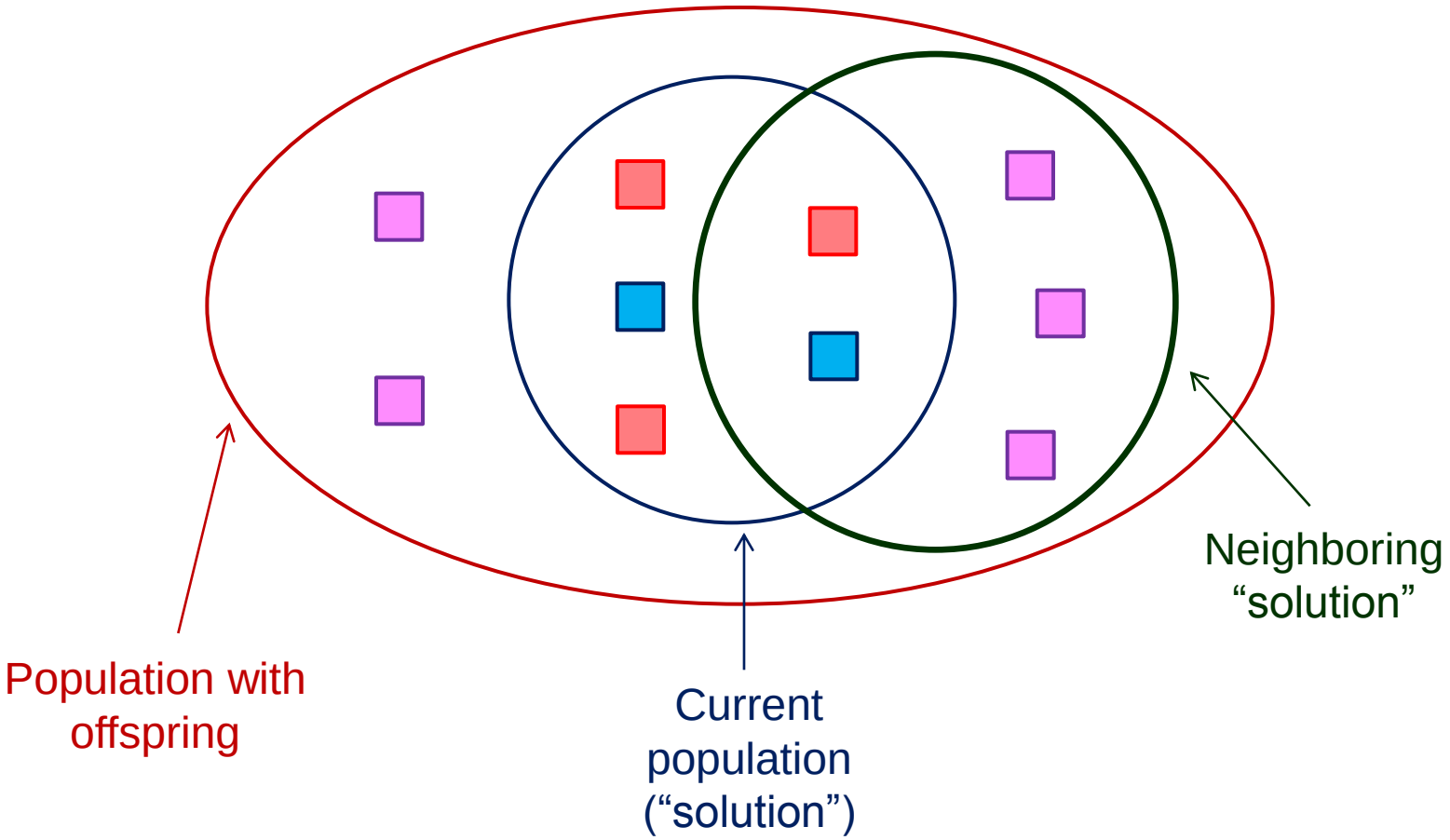
Heuristic Methods: Genetic Algorithms



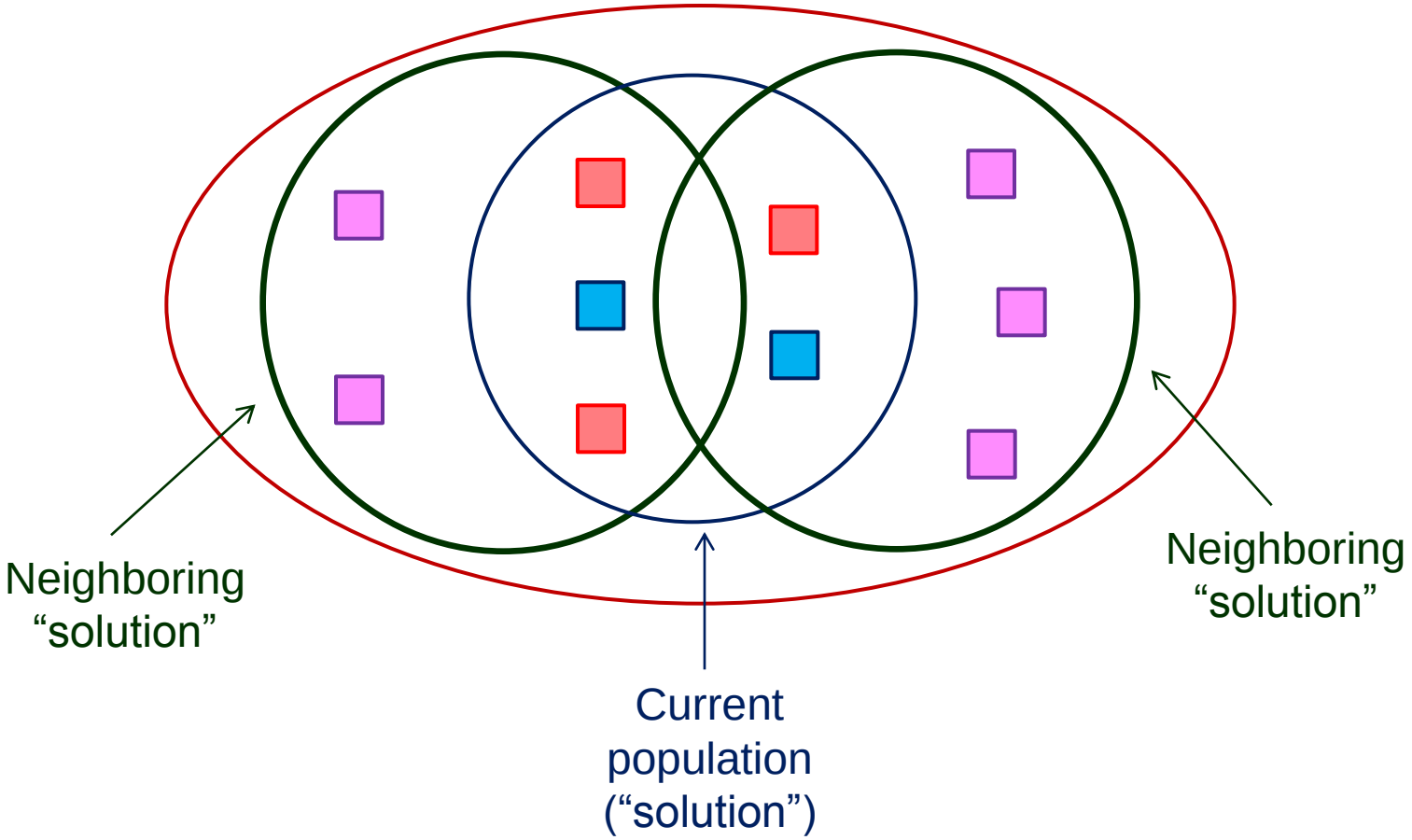
Population with offspring

Current population ("solution")

Heuristic Methods: Genetic Algorithms



Heuristic Methods: Genetic Algorithms



Heuristic Methods: Genetic Algorithms

- Can now use relaxation bounding
  - Create offspring until neighborhood contains a population with fitness better than current lower bound.
  - In effect, neighborhoods without this property are pruned.
- A device for sizing neighborhoods.
  - Enlarge feasible set of restriction until subproblem has an acceptable solution.

Heuristic Methods: Ant Colony Optimization

- A “social” algorithm
  - Agents learn from discoveries made by others.
  - Ants deposit pheromones as they search for food.
  - Trails that lead to food are reinforced, short trails more strongly (pheromones evaporate).

Heuristic Methods: Ant Colony Optimization

- A “social” algorithm
  - Agents learn from discoveries made by others.
  - Ants deposit pheromones as they search for food.
  - Trails that lead to food are reinforced, short trails more strongly (pheromones evaporate).
- Actually a GRASP algorithm.
  - …with only a constructive phase.
  - Can therefore use relaxation bounding.

Heuristic Methods: Ant Colony Optimization

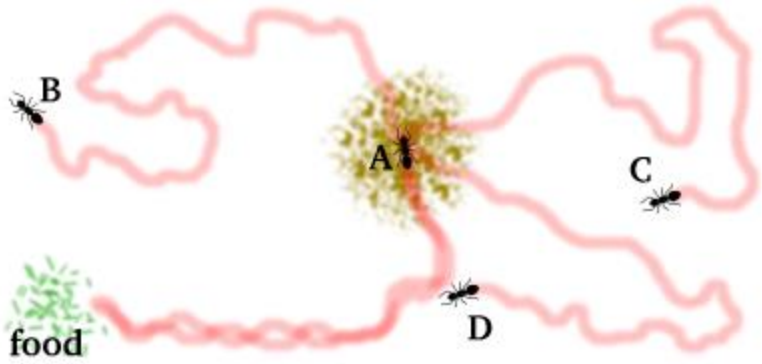
- How ants find food:

Ants search randomly, A finds food.

Heuristic Methods: Ant Colony Optimization

- How ants find food:

Ants search randomly, A finds food.

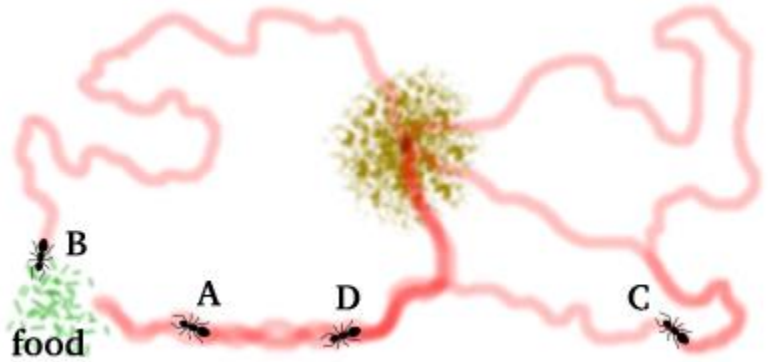


Returns to anthill, strengthens trail.

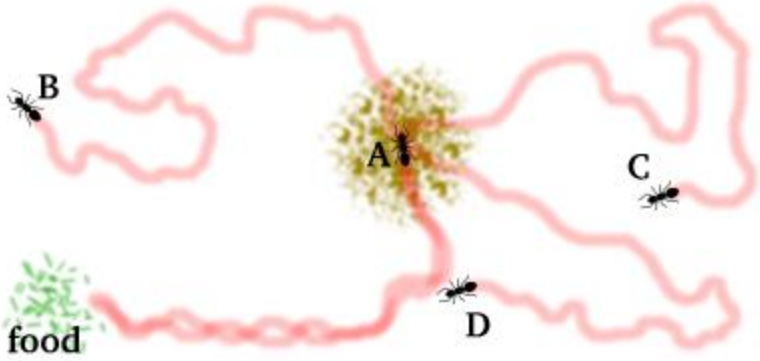
Heuristic Methods: Ant Colony Optimization

- How ants find food:

Ants search randomly, A finds food.



D follows A's trail, B finds food,
C discovers D's trail, which leads to A's

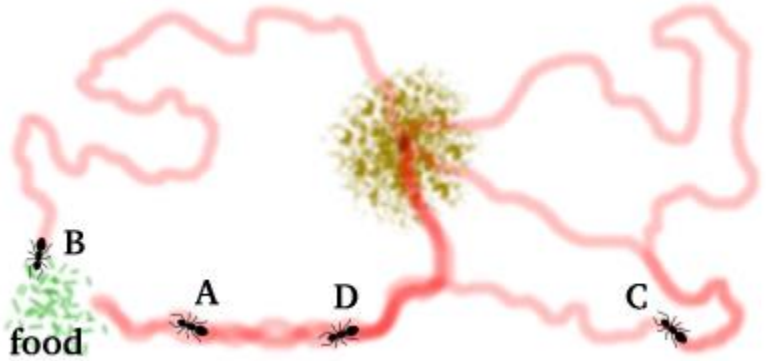


Returns to anthill, strengthens trail.

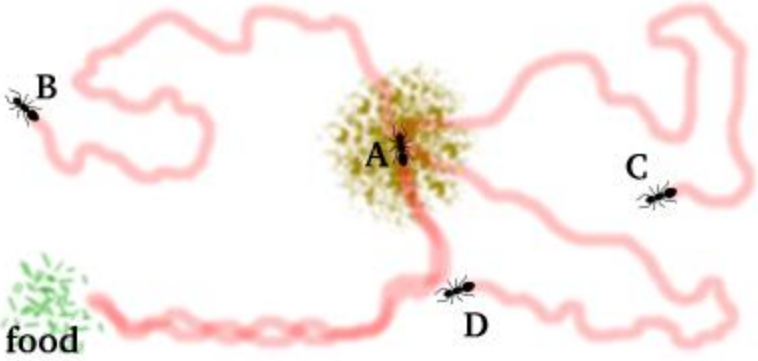
Heuristic Methods: Ant Colony Optimization

- How ants find food:

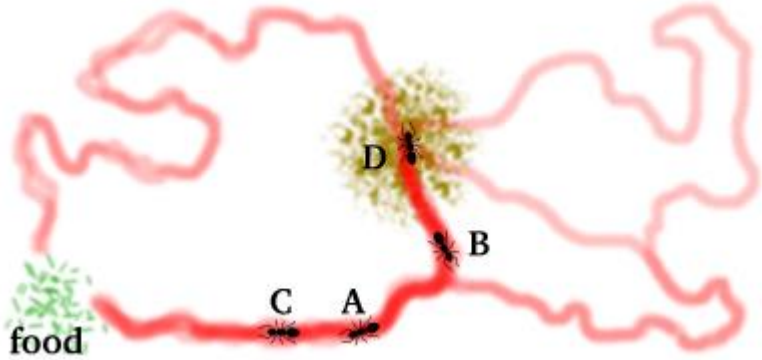
Ants search randomly, A finds food.



D follows A's trail, B finds food,
C discovers D's trail, which leads to A's



Returns to anthill, strengthens trail.



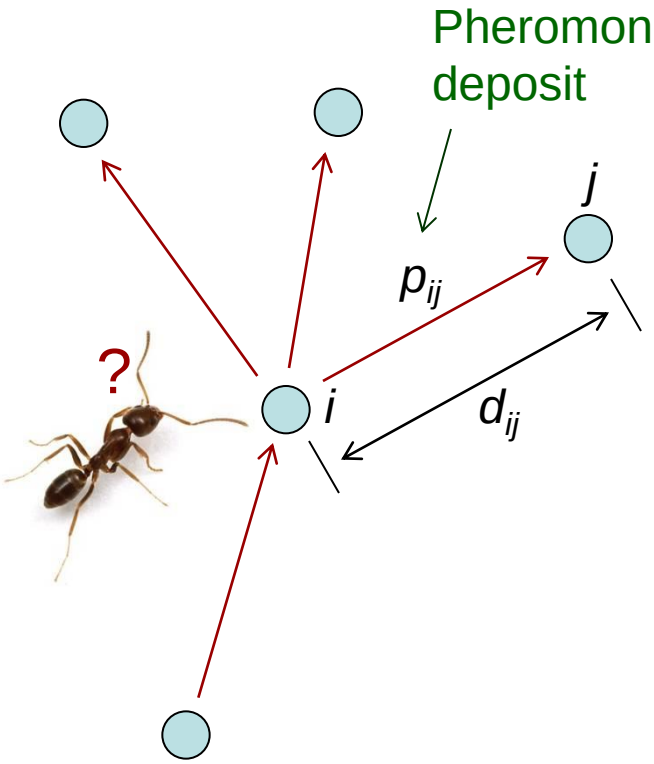
Ants favor shorter trail, more deposits.

Heuristic Methods: Ant Colony Optimization

- It's actually more complicated
  - Ants use visual memories of path, as well as pheromones.
    - Faster than following pheromone trail.
  - Ants also lead others to the food
    - Making sure that followers keep up.

Heuristic Methods: Ant Colony Optimization

- Application to TSP



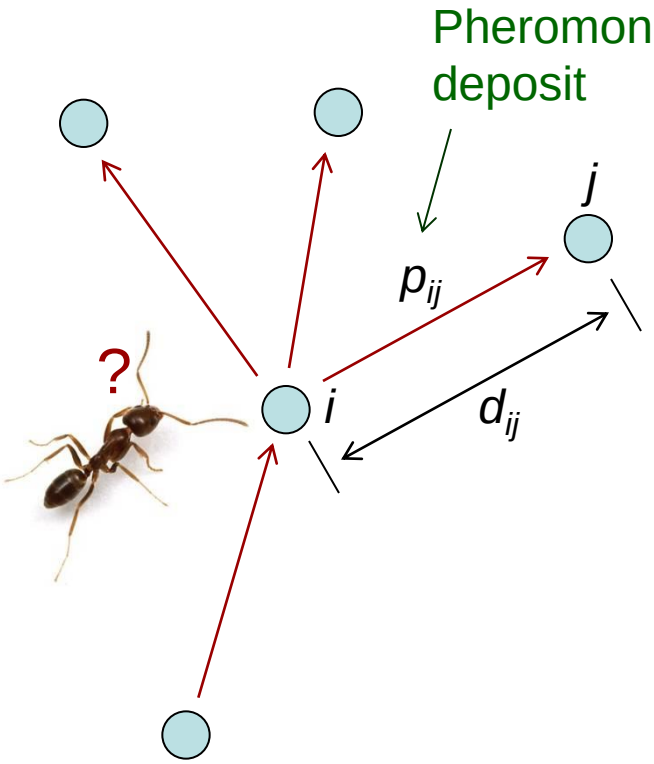
In each iteration, each ant sets out to find a tour.

Ant remembers which cities it has visited in current iteration.

Ant at city *i* chooses path to city *j* with probability proportional to p_{ij}/d_{ij}

Heuristic Methods: Ant Colony Optimization

- Application to TSP



In each iteration, each ant sets out to find a tour.

Ant remembers which cities it has visited in current iteration.

Ant at city *i* chooses path to city *j* with probability proportional to p_{ij}/d_{ij}

This is a randomized greedy heuristic -- constructive phase of a GRASP.

Use relaxation bounds to prune search as in GRASP.

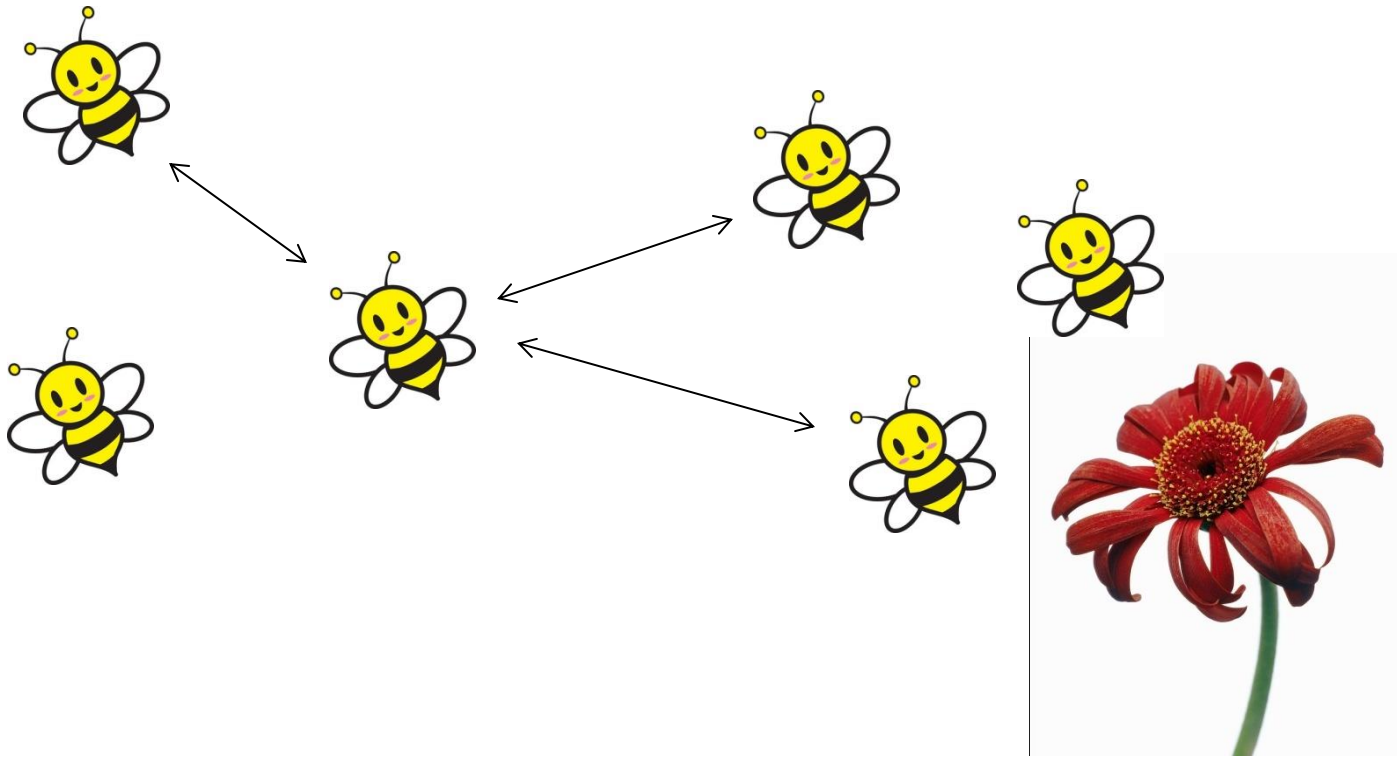
Heuristic Methods: Particle Swarms

- Another “social” algorithm
  - Can be used for continuous optimization.
  - Particle locations = collection of solutions = problem restriction.
- A search over problem restrictions.
  - As swarm locations evolve.
- Can be viewed as constraint-directed search.
  - With inductive inference, as in GA.

Heuristic Methods: Particle Swarms

Each particle location represents a solution

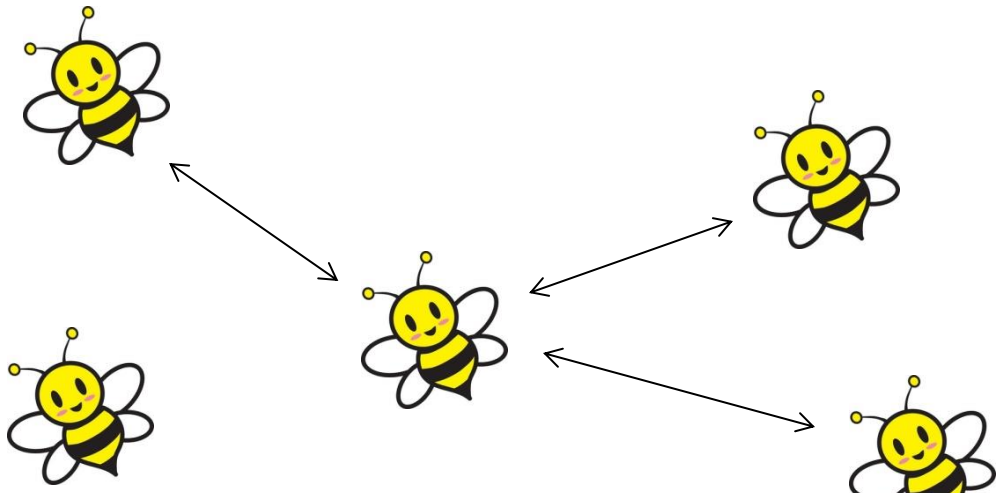
Particles communicate with certain others.



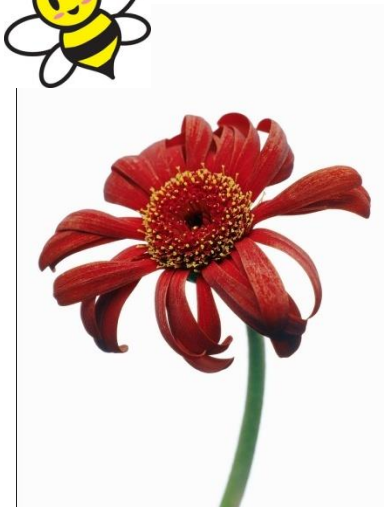
Heuristic Methods: Particle Swarms

Each particle location represents a solution

Particles communicate with certain others.



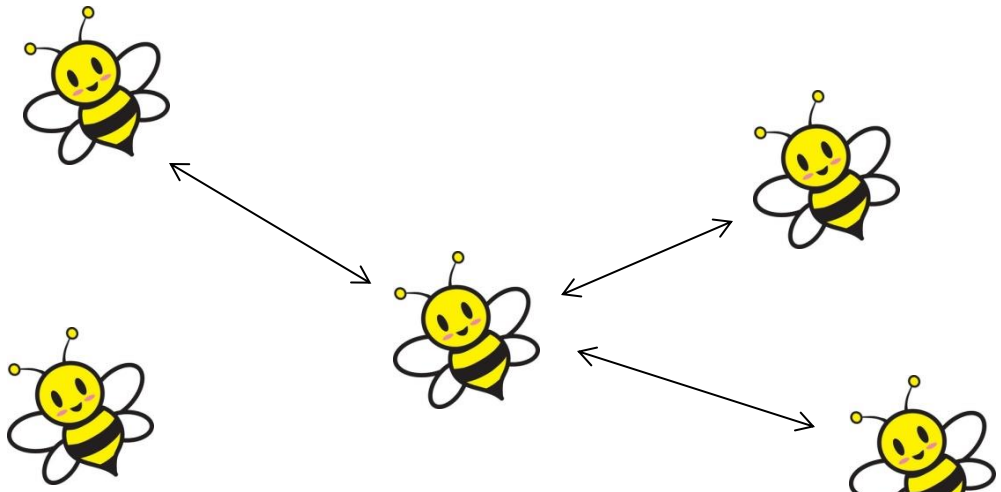
Particles gravitate toward particles in a better region of the search space.



Heuristic Methods: Particle Swarms

Each particle represents a solution

and communicates with certain other particles



Particles gravitate toward particles in a better region of the search space.

Heuristic Methods: Particle Swarms

- An instance of constraint-based search
  - "Solve" problem restriction by identifying good solutions occupied by particles.
    - Note regions with better solutions.
  - Inductive inference dual infers solution quality from region.
  - This knowledge used to select next restriction
    - That is, relocate the particles.

Outline

- Primal-dual framework
  - Inference dual
  - Relaxation dual
  - Constraint-directed search
  - DPLL
- Exact Methods
  - Simplex
  - Branch and bound
  - Benders decomposition
- Heuristic Methods
  - Local search
  - GRASP
  - Tabu search
  - Genetic algorithms
  - Ant colony optimization
  - Particle swarm optimization
- **Summing up**

Summing Up: Exact Methods

Method	Search over Restrictions	Bounds from Relaxation Dual	Nogoods from Inference Dual	To Convert to a Heuristic Method
DPLL (for SAT)	Leaf nodes of search tree		Conflict clauses	Drop some conflict clauses
Simplex (for LP)	Edges	Surrogate dual	Reduced costs	Terminate prematurely
B&B (for IP)	Leaf nodes of search tree	LP bounds	Conflict analysis	Forget tree & use nogood constraints
B&B as local search	Incomplete search trees	Uphill search over trees		Terminate prematurely
Benders decomposition	Subproblems	Master problem	Logic-based Benders cuts	Drop some Benders cuts

Summing Up: Heuristic Methods

Method	Search over Restrictions	Bounds from Relaxation Dual	Nogoods from Inference Dual	To covert to an Exact Method
Local search (e.g. for TSP)	Neighbor-hoods (e.g. 2-opt)	Control neighborhood size		Search over partial assignments
GRASP (e.g. for TSPTW)	Nodes of incomplete search tree	As in B&B		Don't forget nodes, complete the tree
Tabu search (e.g. for TSPTW)	Neighbor-hoods		Tabu list	Search over partial solutions, dynamic backtracking

Summing Up: Heuristic Methods

Method	Search over Restrictions	Bounds from Relaxation Dual	Nogoods from Inference Dual	To covert to an Exact Method
Genetic algorithm	Populations		Crossover guidance from inductive inference dual	
Genetic algorithm as local search	Subsets of population + offspring	Control neighborhood size		
Ant colonies	Same as GRASP	As in GRASP		As in GRASP
Particle swarm	Sets of swarm locations		Relocation guidance from inductive inferences	