Unifying Local and Exhaustive Search

John Hooker
Carnegie Mellon University

Encuentro Mexicano de Computación
September 2005

Exhaustive vs. Local Search

- They are generally regarded as very different.
- Exhaustive methods examine every possible solution, at least implicitly.
  - Branch and bound, Benders decomposition.
- Local search methods typically examine only a portion of the solution space.
  - Simulated annealing, tabu search, genetic algorithms, GRASP (greedy randomized adaptive search procedure).

Exhaustive vs. Local Search

- However, exhaustive and local search are often closely related.
- “Heuristic algorithm” = “search algorithm”
  - “Heuristic” is from the Greek ‘ευριςτέιν (to search, to find).
- Two classes of exhaustive search methods are very similar to corresponding local search methods:
  - Branching methods.
  - Nogood-based search.

Type of search	*Exhaustive search examples*	*Local search examples*
Branching	▪ Branch and bound ▪ DPL for SAT	▪ Simulated annealing ▪ GRASP
Nogood-based	▪ Benders decomposition ▪ DPL with clause learning ▪ Partial order dynamic backtracking	▪ Tabu search

Why Unify Exhaustive & Local Search?

- Encourages design of algorithms that have several exhaustive and inexhaustive options.
  - Can move from exhaustive to inexhaustive options as problem size increases.
- Suggests how techniques used in exhaustive search can carry over to local search.
  - And vice-versa.

Why Unify Exhaustive & Local Search?

- We will use an example (*traveling salesman problem with time windows*) to show:
  - Exhaustive branching can suggest a generalization of a local search method (*GRASP*).
    - The bounding mechanism in branch and bound also carries over to generalized GRASP.
  - Exhaustive nogood-based search can suggest a generalization of a local search method (*tabu search*).

Outline

- Branching search.
  - Generic algorithm (exhaustive & inexhaustive)
  - Exhaustive example: Branch and bound
  - Inexhaustive examples: Simulated annealing, GRASP.
  - Solving TSP with time windows
    - Using exhaustive branching & generalized GRASP.
- Nogood-based search.
  - Generic algorithm (exhaustive & inexhaustive)
  - Exhaustive example: Benders decomposition
  - Inexhaustive example: Tabu search
  - Solving TSP with time windows
    - Using exhaustive nogood-based search and generalized tabu search.

Branching Search

- Each node of the branching tree corresponds to a *restriction* P of the original problem.
  - Restriction = constraints are added.
- Branch by generating restrictions of P.
  - Add a new leaf node for each restriction.
- Keep branching until problem is “easy” to solve.
- Notation:
  - *feas*(P) = feasible set of P
  - *relax*(P) = a relaxation of P

Branching Search Algorithm

- Repeat while leaf nodes remain:
  - Select a problem *P* at a leaf node.
  - If *P* is "easy" to solve then
    - If solution of *P* is better than previous best solution, save it.
    - Remove *P* from tree.
  - Else
    - If optimal value of *relax*(*P*) is better than previous best solution, then
      - If solution of *relax*(*P*) is feasible for *P* then *P* is "easy"; save the solution and remove *P* from tree
      - Else **branch**.
    - Else
      - Remove *P* from tree

Branching Search Algorithm

- To branch:
  - If set restrictions $P_1, \ldots, P_k$ of P so far generated is *complete*, then
    - Remove P from tree.
  - Else
    - Generate new restrictions $P_{k+1}, \ldots, P_m$ and leaf nodes for them.

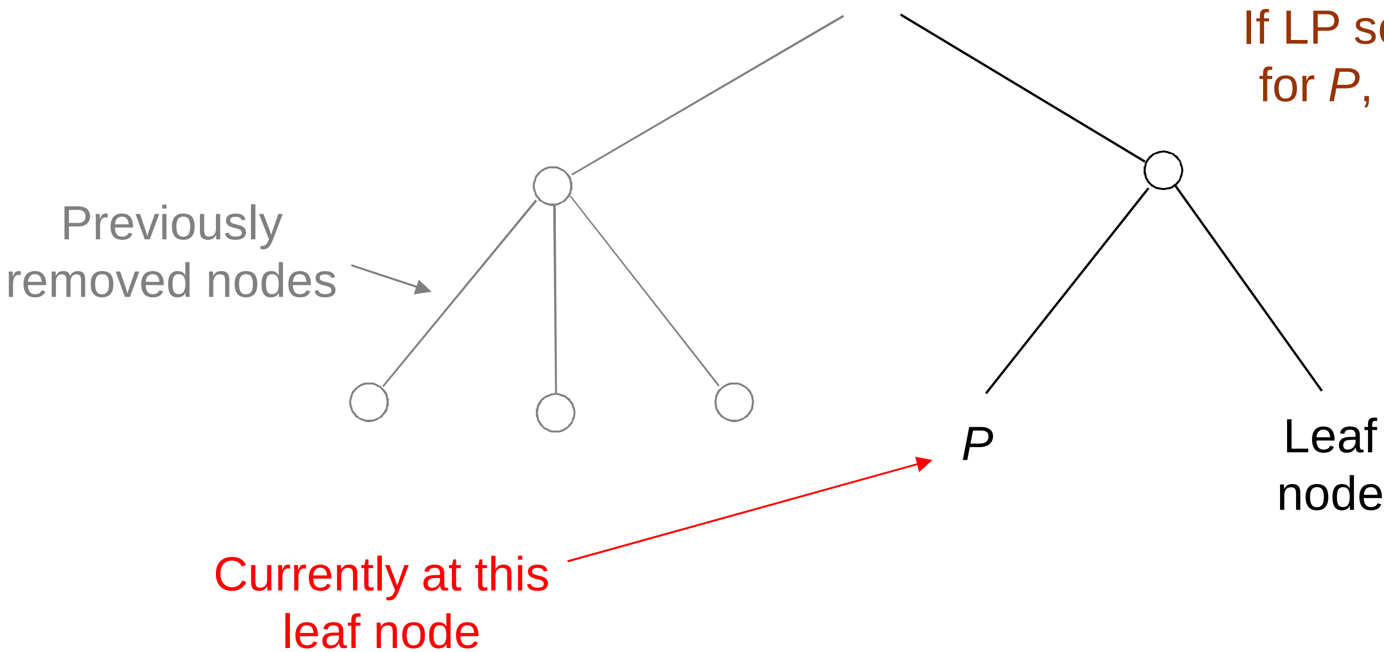
Branching Search Algorithm

- To branch:
  - If set restrictions $P_1, \ldots, P_k$ of P so far generated is *complete*, then
    - Remove P from tree.
  - Else
    - Generate new restrictions $P_{k+1}, \ldots, P_m$ and leaf nodes for them.
- Exhaustive vs. heuristic algorithm
  - In exhaustive search, “complete” = exhaustive

$$\bigcup_i feas(P_i) = feas(P)$$

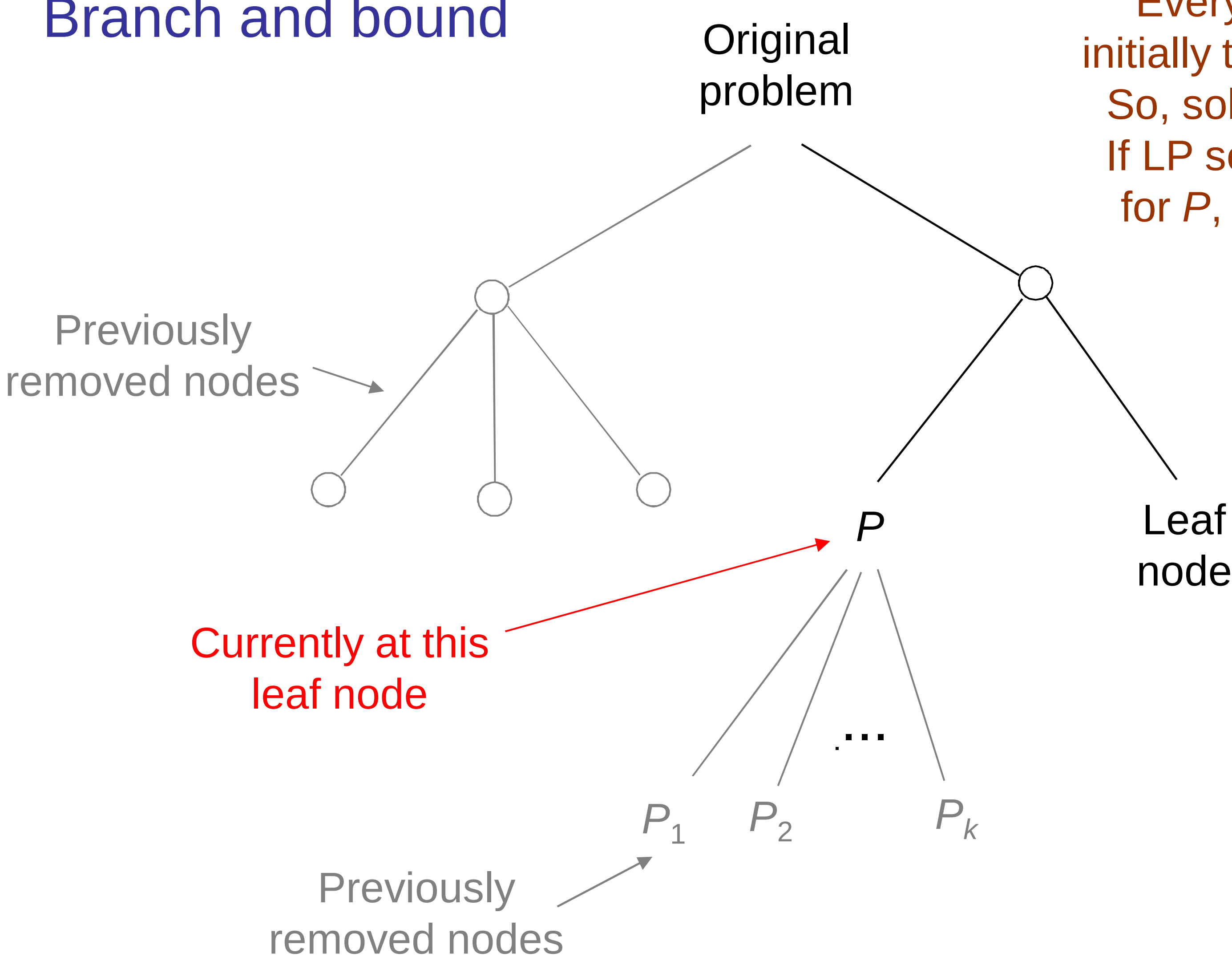
  - In a heuristic algorithm, “complete” $\neq$ exhaustive

Exhaustive search: Branch and bound



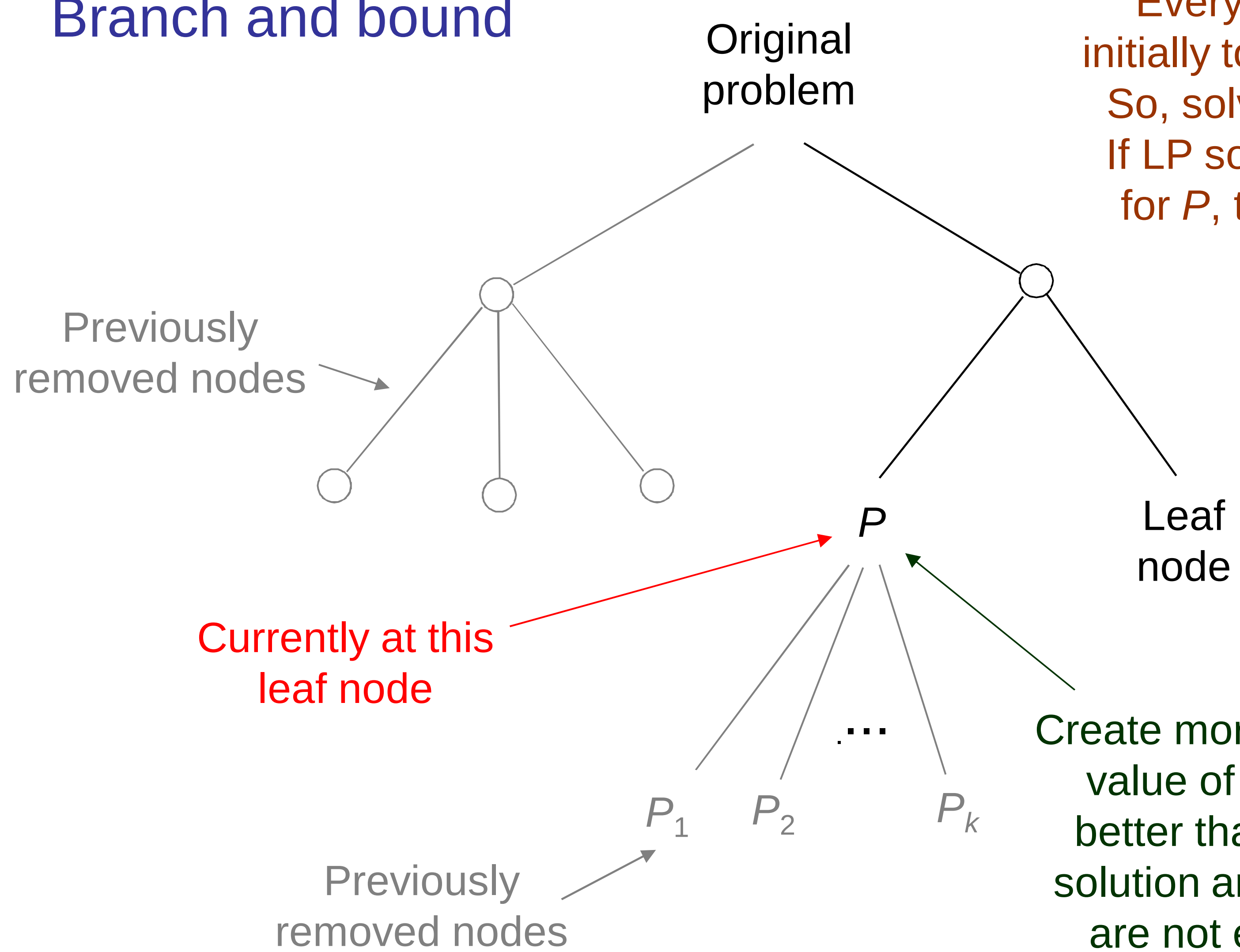
Every restriction *P* is initially too "hard" to solve. So, solve LP relaxation. If LP solution is feasible for *P*, then *P* is "easy."

Exhaustive search: Branch and bound



Every restriction P is initially too "hard" to solve. So, solve LP relaxation. If LP solution is feasible for P, then P is "easy."

Exhaustive search: Branch and bound

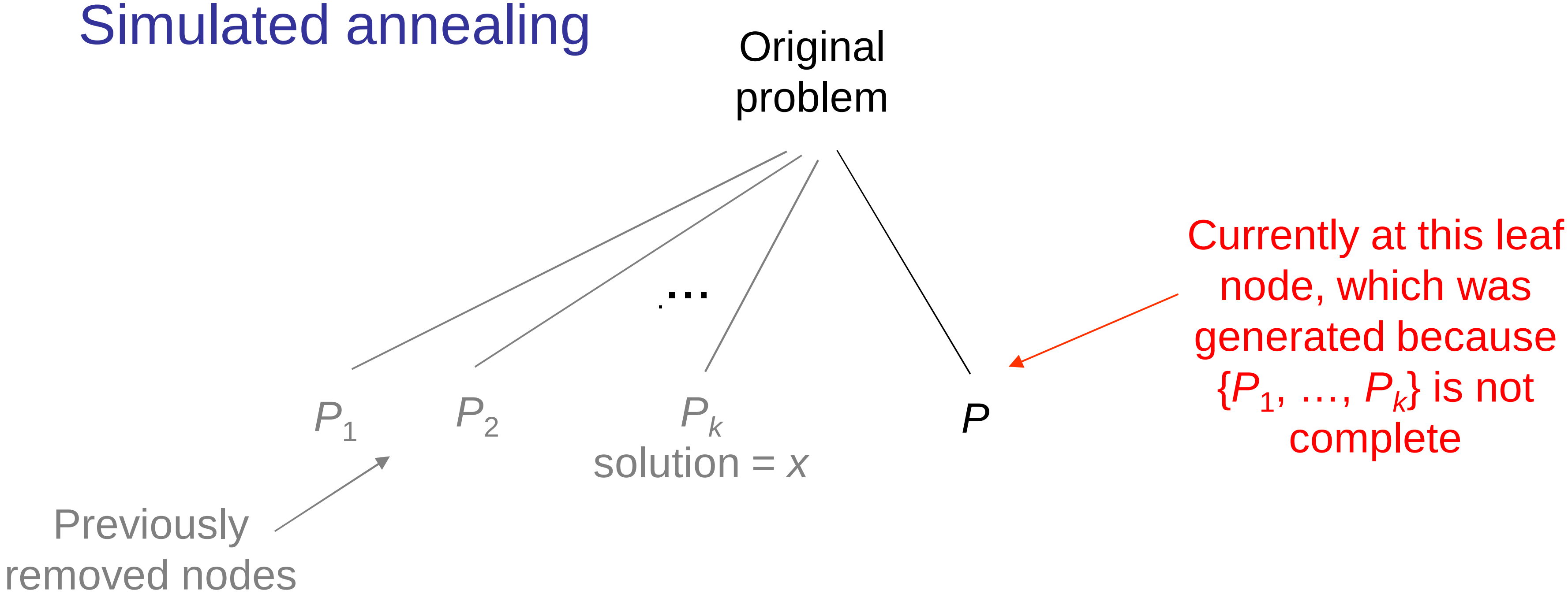


Every restriction P is initially too "hard" to solve. So, solve LP relaxation. If LP solution is feasible for P, then P is "easy."

Create more branches if value of *relax*(P) is better than previous solution and P_1, …, P_2 are not exhaustive

Algorithm	*Restriction P*	*P easy enough to solve?*	*relax(P)*
Branch and bound (exhaustive)	Created by splitting variable domain	Never. Always solve *relax*(*P*)	LP relaxation
Simulated annealing (inexhaustive)			
GRASP (inexhaustive) *Greedy phase* *Local search phase*			

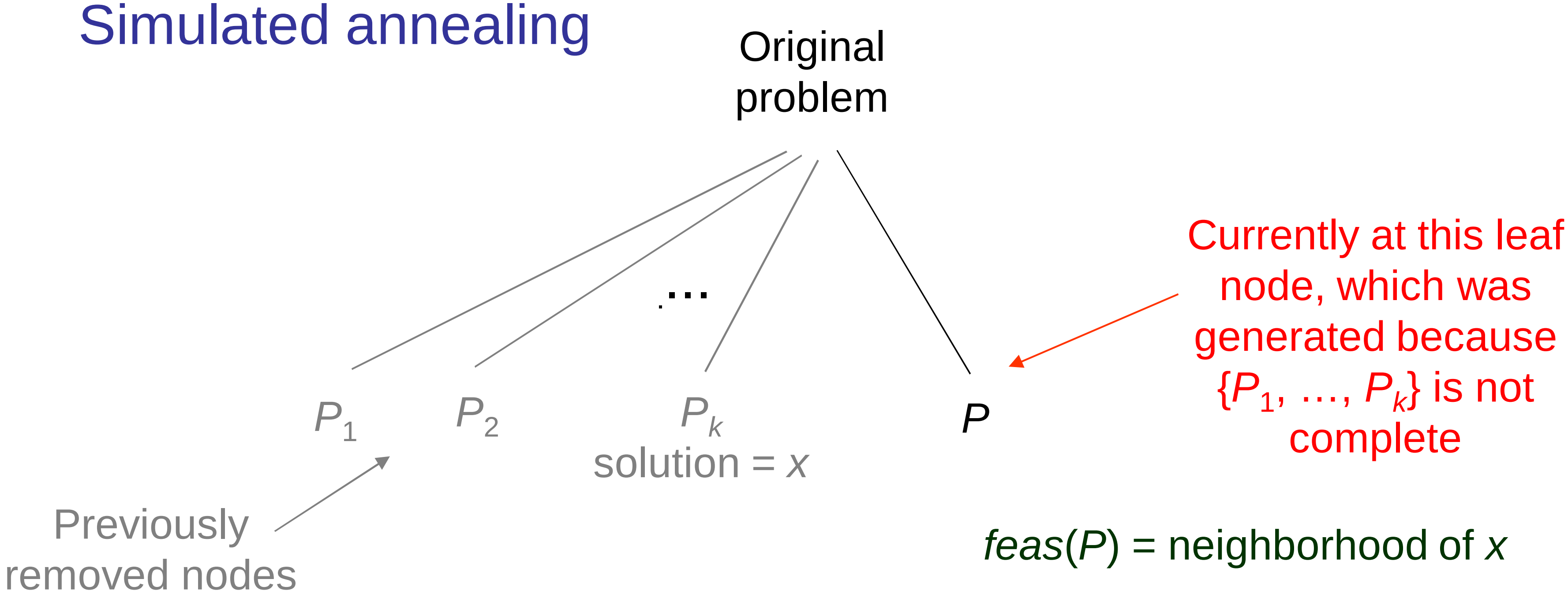
Heuristic algorithm: Simulated annealing



Search tree has 2 levels.

Second level problems are always "easy" to solve by searching neighborhood of previous solution.

Heuristic algorithm: Simulated annealing



feas(P) = neighborhood of x

Randomly select $y \in$ *feas*(P)

Solution of P =
y if y is better than x;
otherwise, y with probability p,
x with probability $1 - p$.

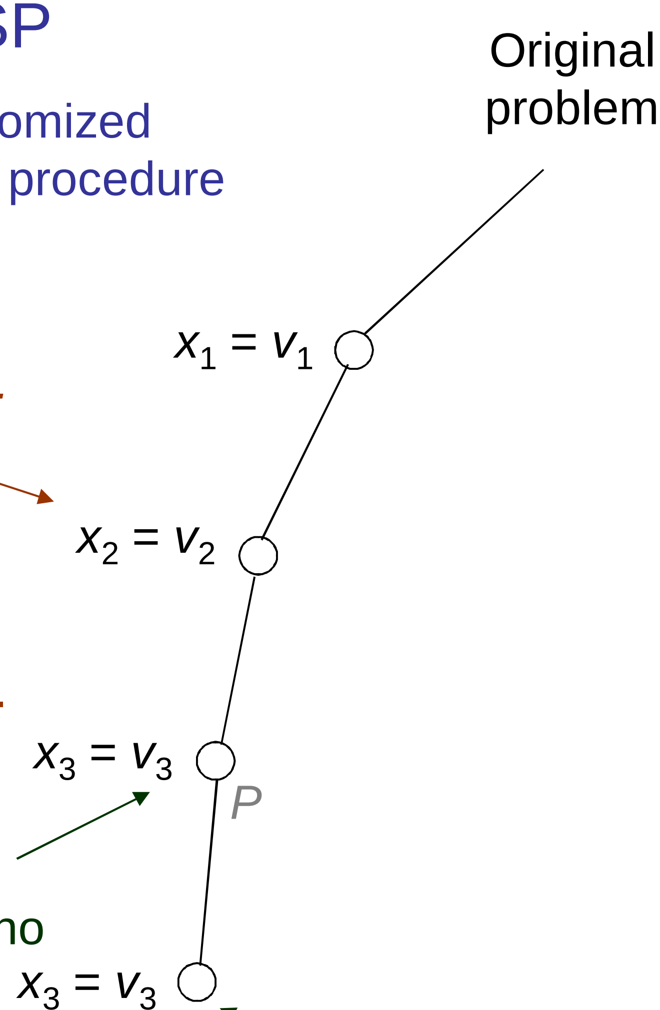
Search tree has 2 levels.

Second level problems are always "easy" to solve by searching neighborhood of previous solution.

Algorithm	*Restriction P*	*P easy enough to solve?*	*relax(P)*
Branch and bound (exhaustive)	Created by splitting variable domain	Never. Always solve *relax*(*P*)	LP relaxation
Simulated annealing (inexhaustive)	Created by defining neighborhood of previous solution	Always. Examine random element of neighborhood.	Not used.
GRASP (inexhaustive) *Greedy phase* *Local search phase*			

Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure



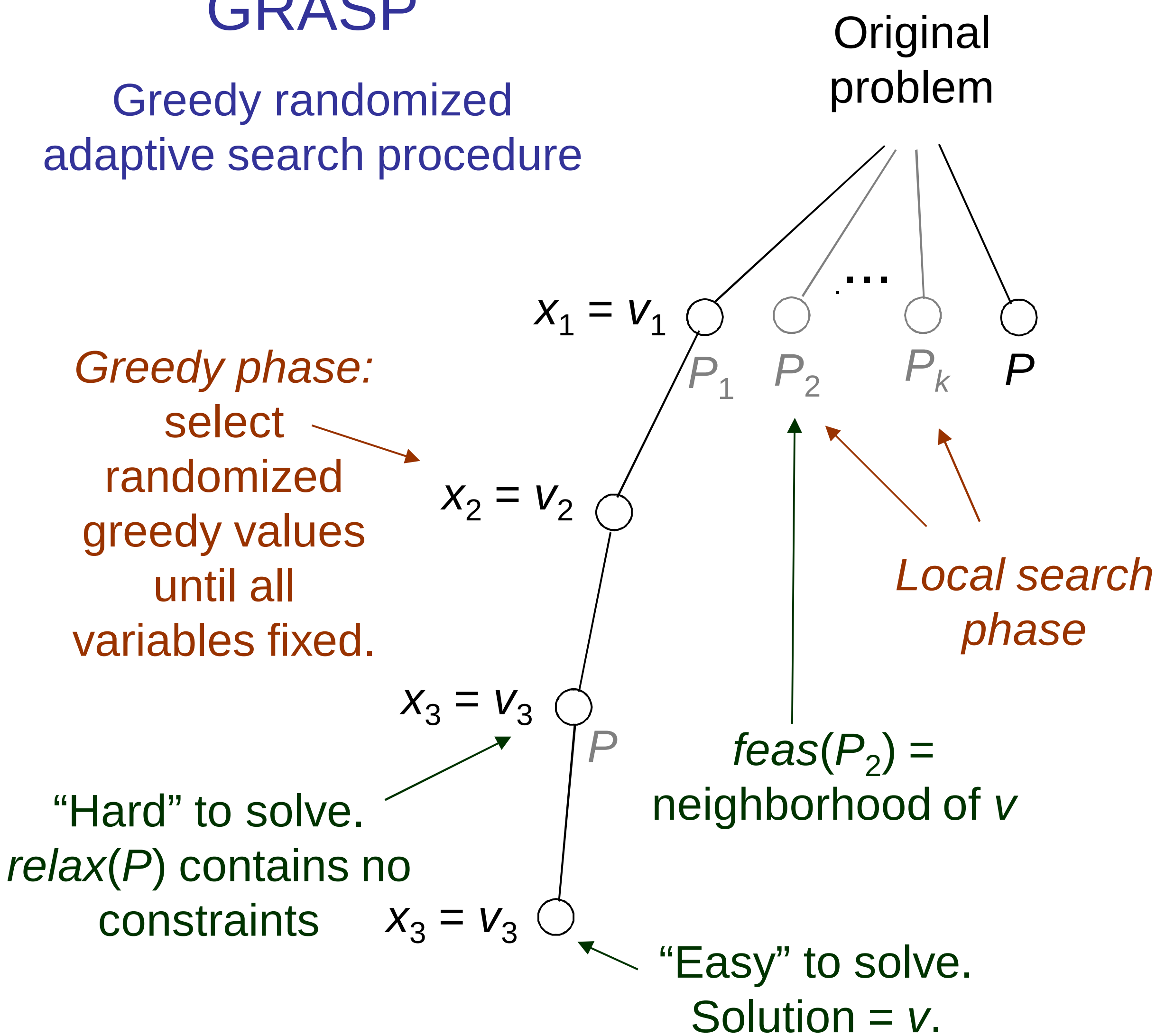
Greedy phase: select randomized greedy values until all variables fixed.

"Hard" to solve. *relax*(*P*) contains no constraints

"Easy" to solve. Solution = *v*.

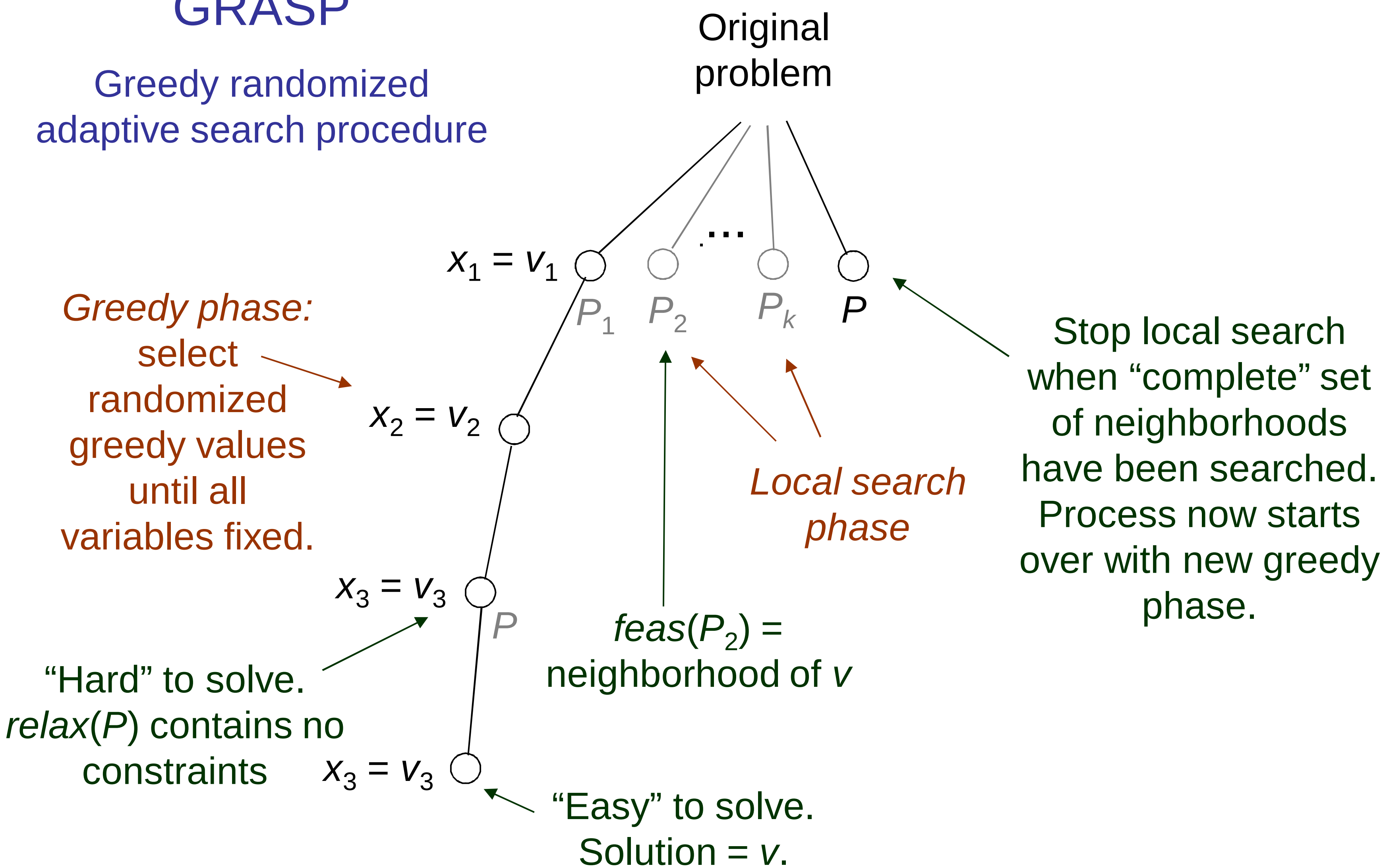
Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure



Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure



Algorithm	*Restriction P*	*P easy enough to solve?*	*relax(P)*
Branch and bound (exhaustive)	Created by splitting variable domain	Never. Always solve *relax*(*P*)	LP relaxation
Simulated annealing (inexhaustive)	Created by defining neighborhood of previous solution	Always. Examine random element of neighborhood.	Not used.
GRASP (inexhaustive) *Greedy phase*	Created by assigning next variable a randomized greedy value.	Not until all variables are assigned values.	No constraints, so solution is always infeasible and branching necessary.
Local search phase	Created by defining neighborhood of previous solution.	Always. Search neighborhood.	Not used.

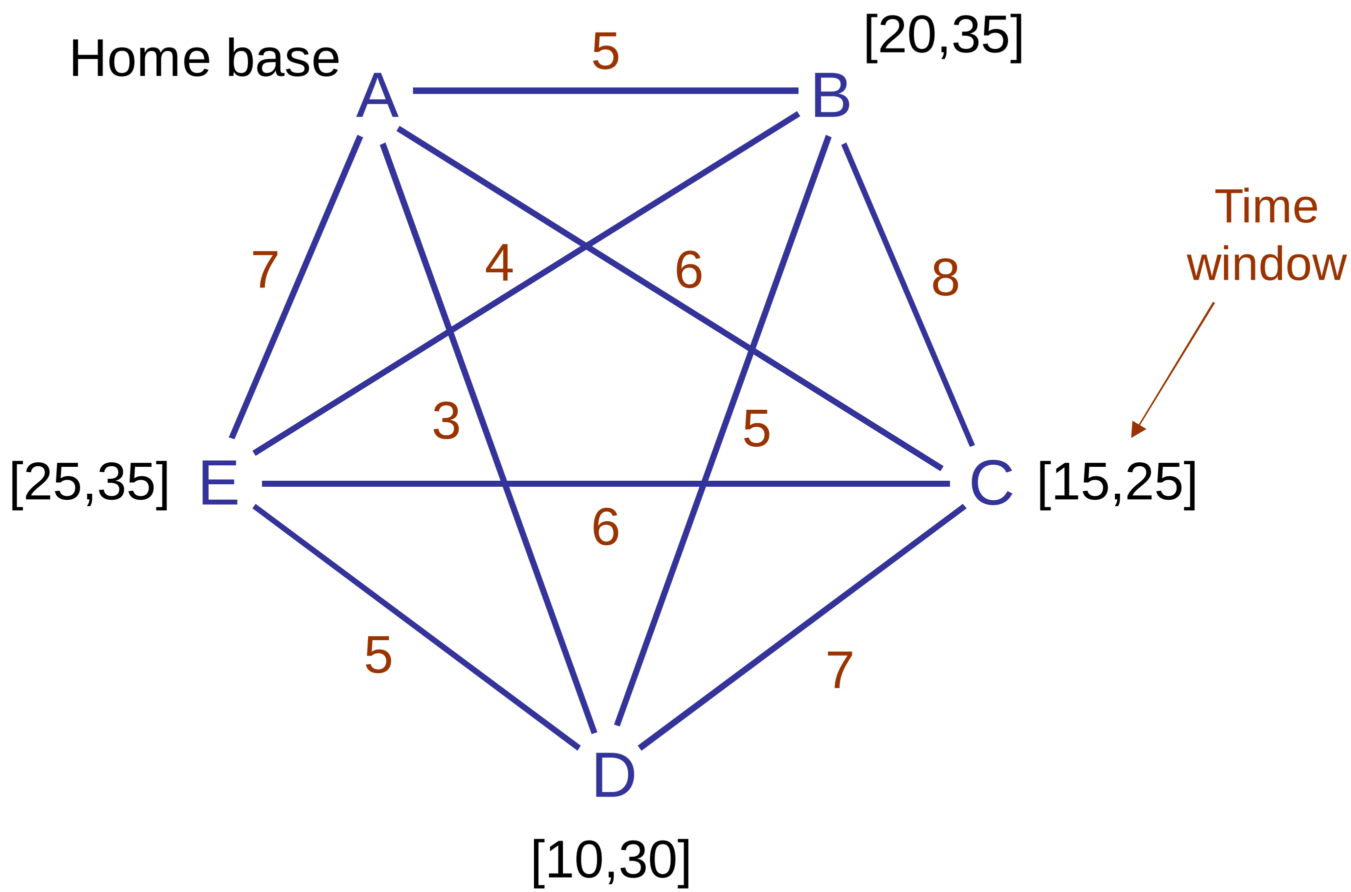
An Example

TSP with Time Windows

- A salesman must visit several cities.
- Find a minimum-length tour that visits each city exactly once and returns to home base.
- Each city must be visited within a time window.

An Example

TSP with Time Windows



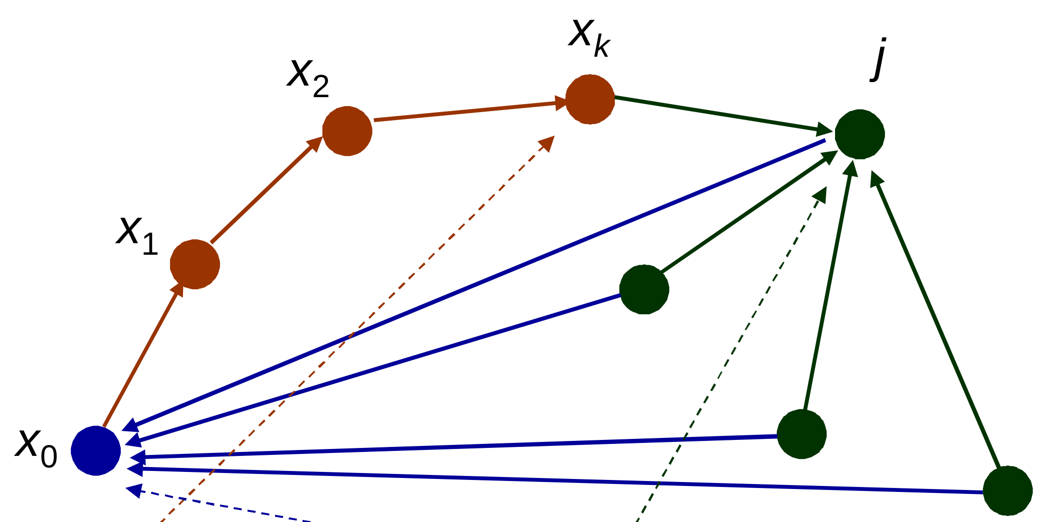
Relaxation of TSP

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

$$\underbrace{\boxed{T}}_{\text{Earliest time vehicle can leave customer } k} + \sum_{j \notin \{x_0, \ldots, x_k\}} \underbrace{\boxed{\min\left\{ t_{x_k j},\ \min_{i \notin \{j, x_0, \ldots, x_k\}} \{t_{ij}\} \right\}}}_{\text{Min time from customer } j\text{'s predecessor to } j} + \underbrace{\boxed{\min_{j \notin \{x_0, \ldots, x_k\}} \{t_{j0}\}}}_{\text{Min time from last customer back to home}}$$



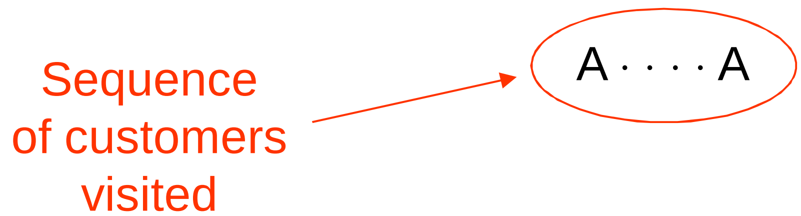
$$\boxed{T} + \sum_{j \notin \{x_0,\ldots,x_k\}} \boxed{\min\left\{ t_{x_k j}, \min_{i \notin \{j, x_0,\ldots,x_k\}} \{t_{ij}\} \right\}} + \boxed{\min_{j \notin \{x_0,\ldots,x_k\}} \{t_{j0}\}}$$

Earliest time vehicle can leave customer *k*

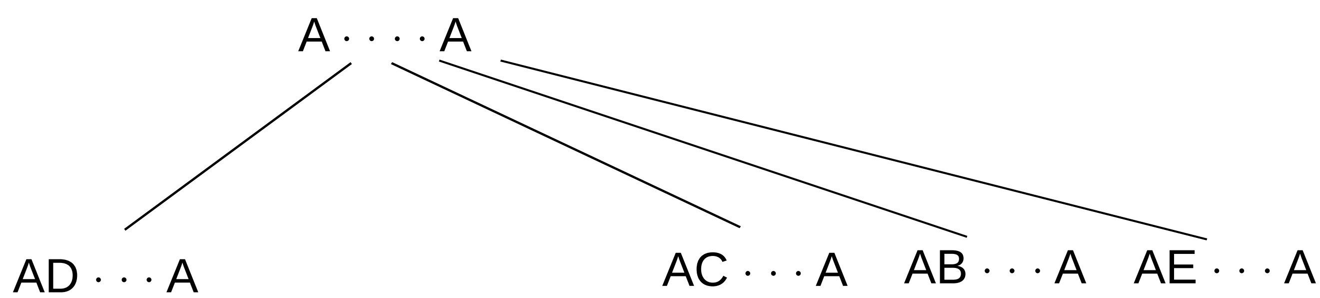
Min time from customer *j* 's predecessor to *j*

Min time from last customer back to home

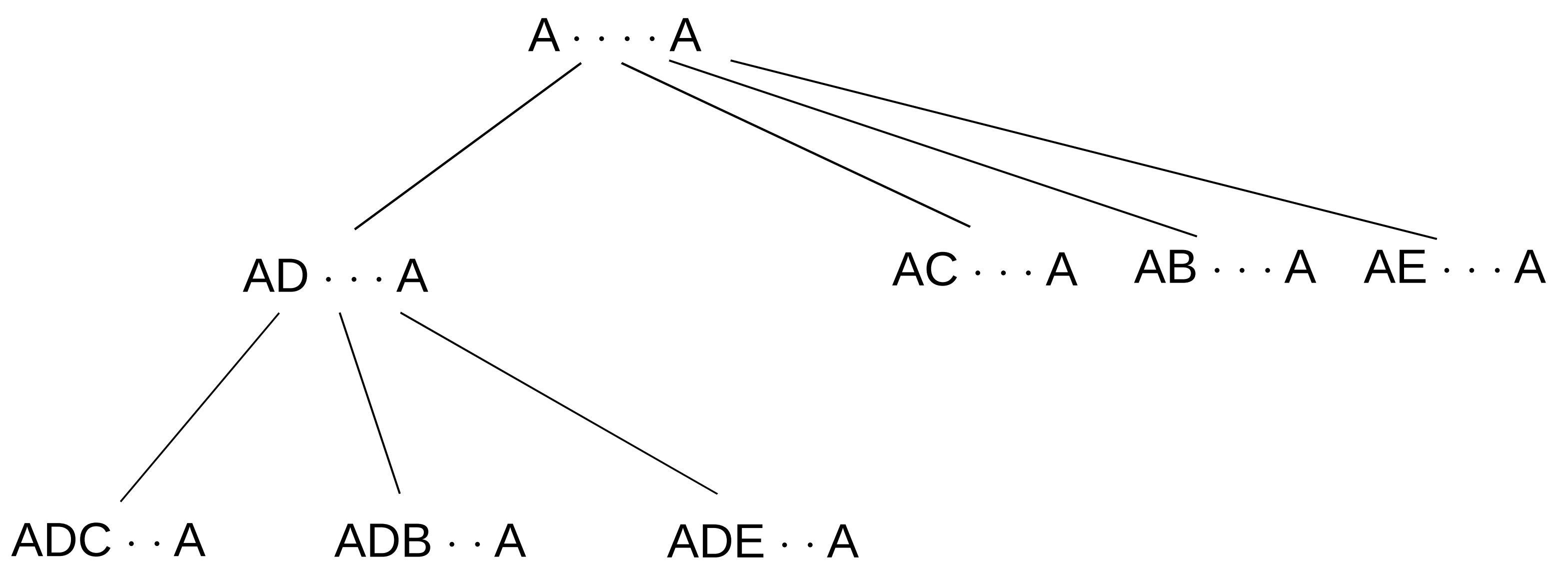
Exhaustive Branch-and-Bound



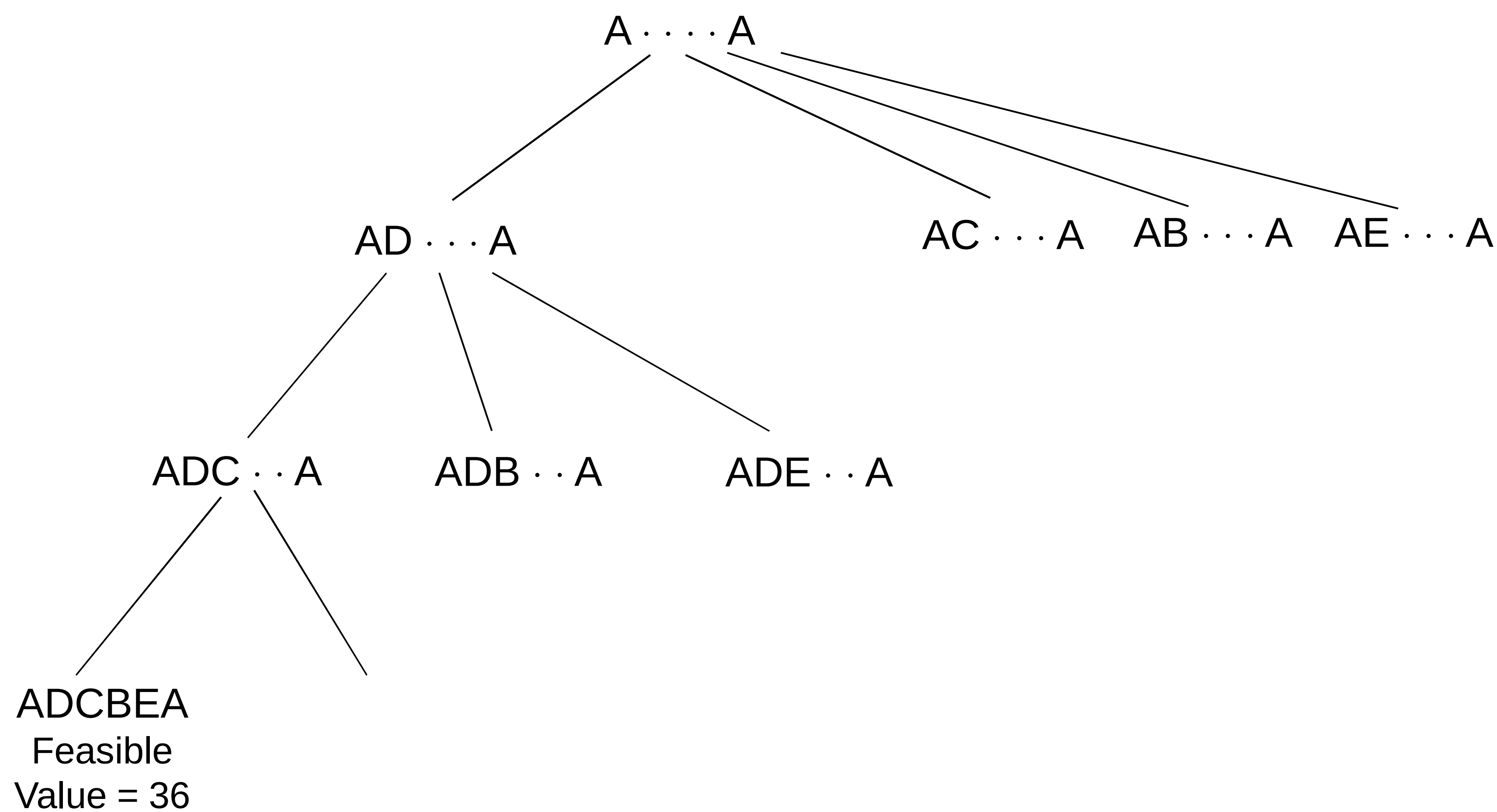
Exhaustive Branch-and-Bound



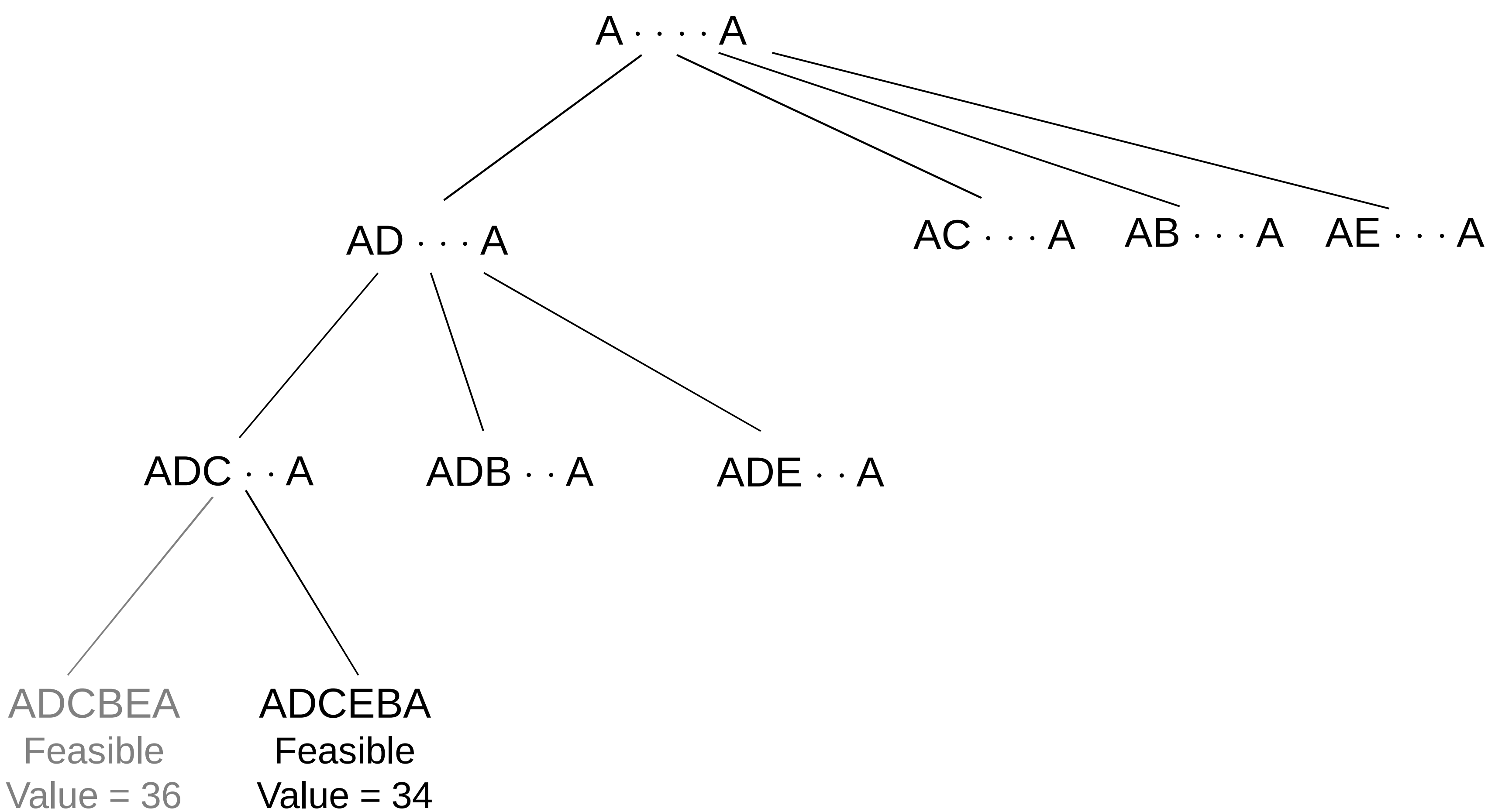
Exhaustive Branch-and-Bound



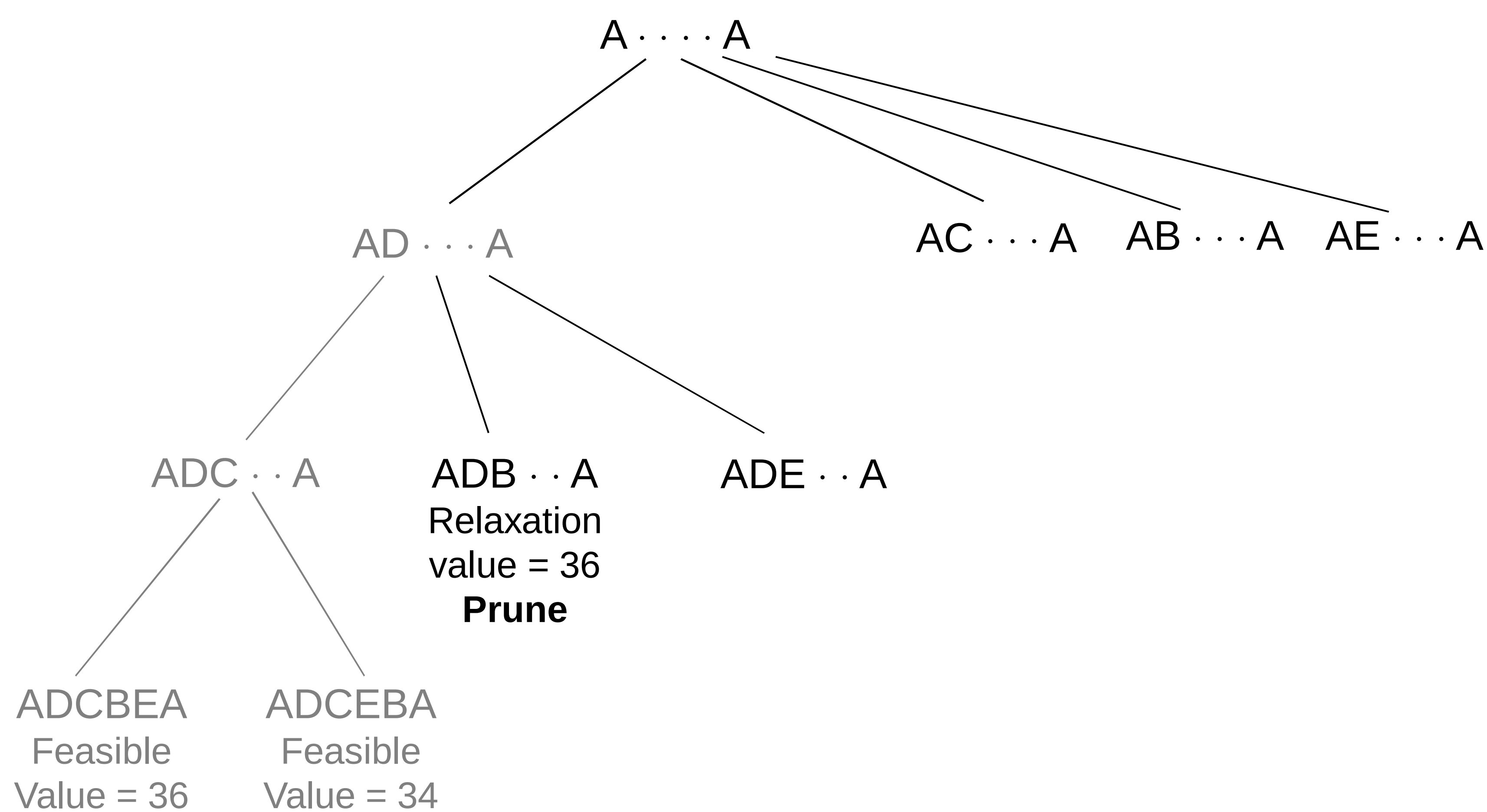
Exhaustive Branch-and-Bound



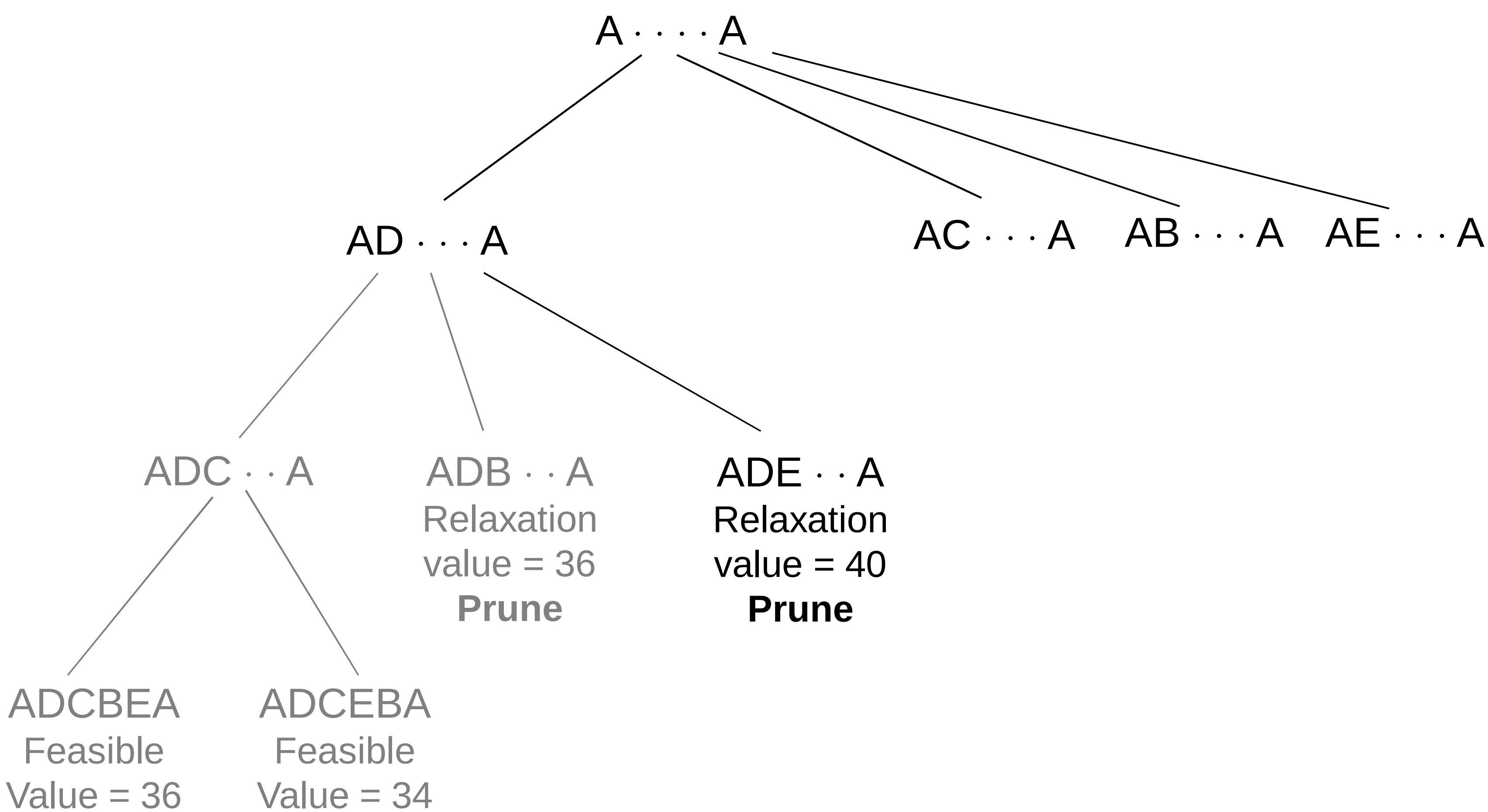
Exhaustive Branch-and-Bound



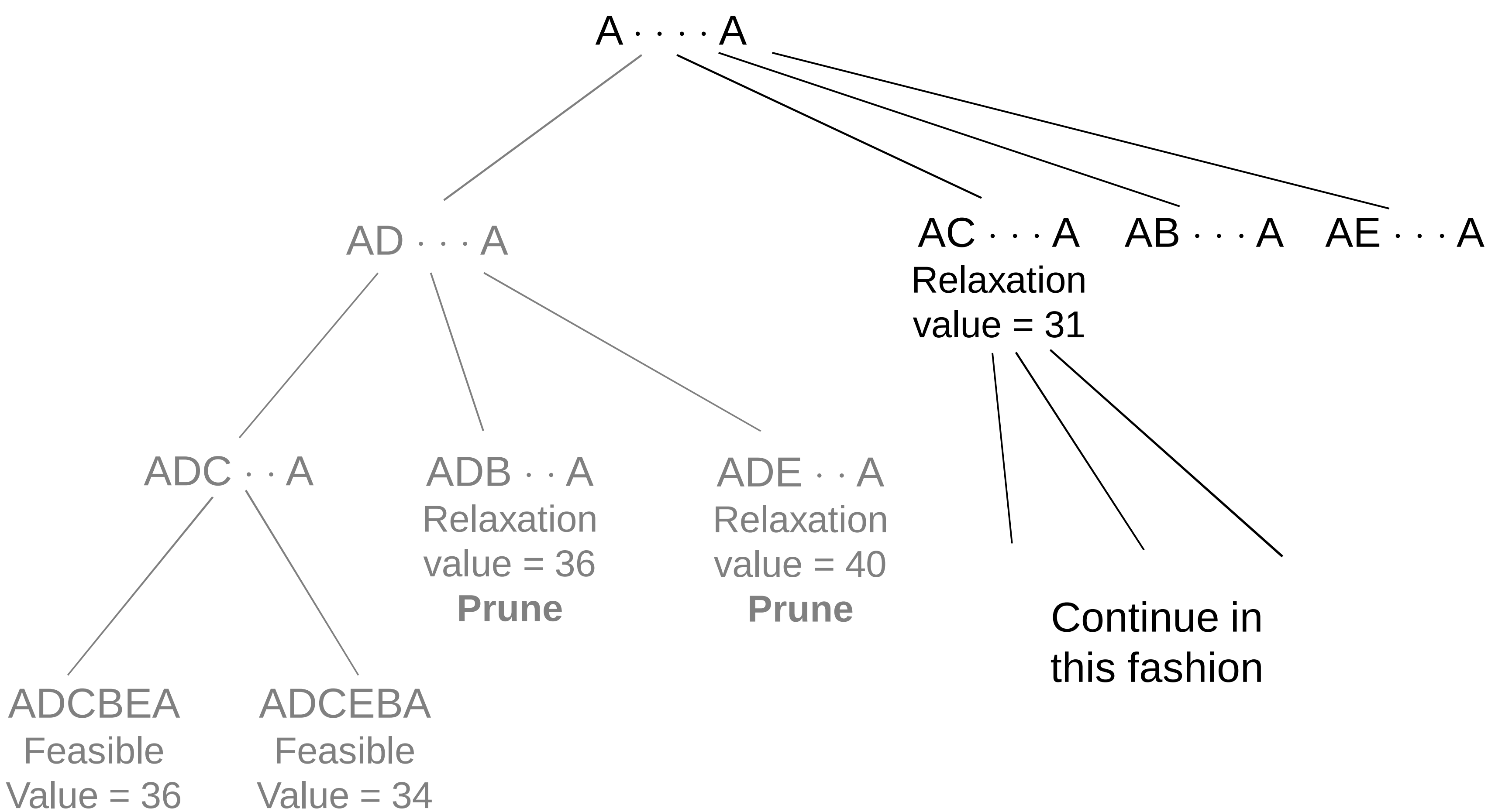
Exhaustive Branch-and-Bound



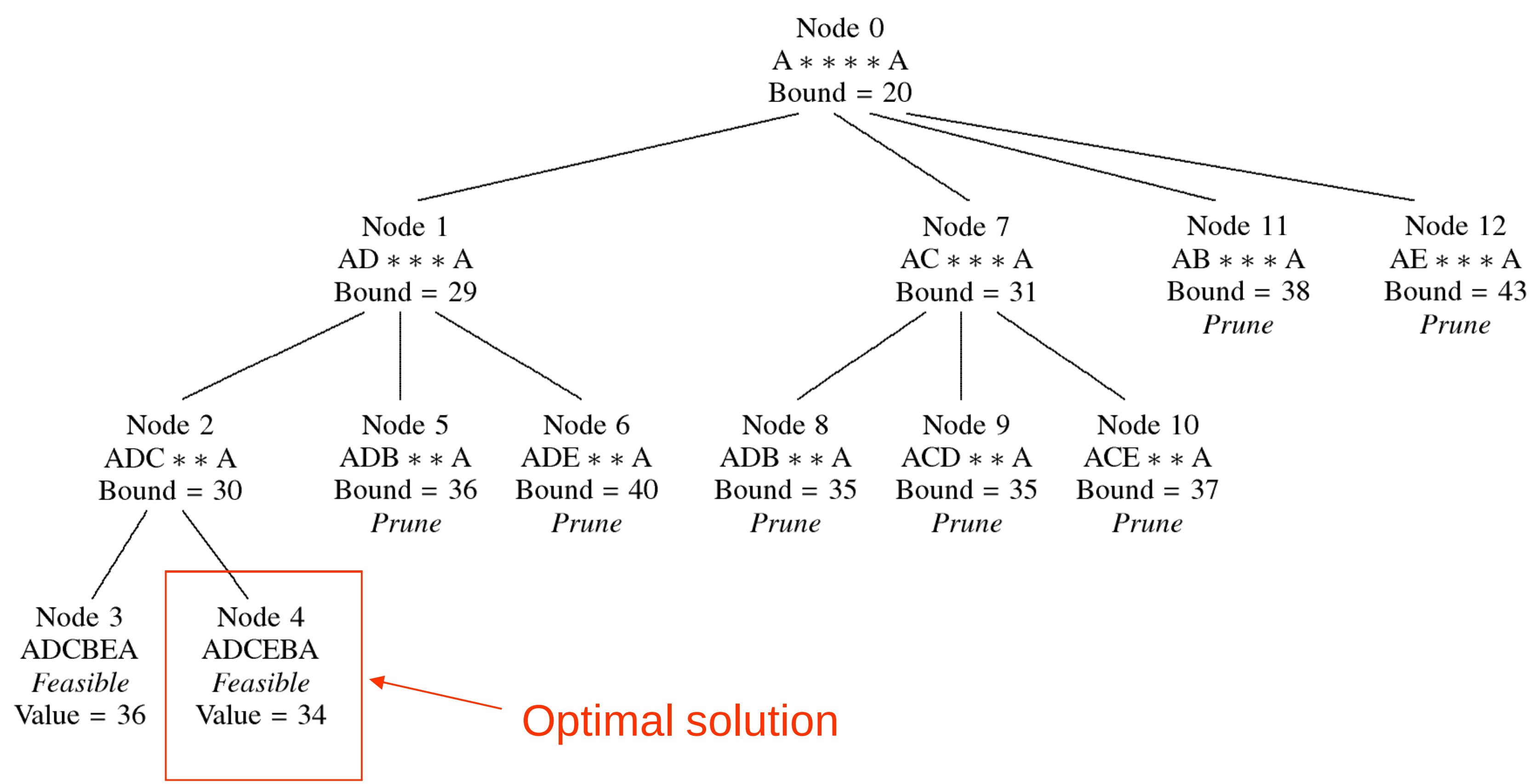
Exhaustive Branch-and-Bound



Exhaustive Branch-and-Bound



Exhaustive Branch-and-Bound



Algorithm	*Restriction* *P*	*P easy enough to solve?*	*relax(P)*
Branch and bound for TSPTW (exhaustive)	Created by fixing next variable	Never. Always solve *relax*(*P*).	TSPTW relaxation
Generalized GRASP for TSPTW (inexhaustive) *Greedy phase* *Local search phase*			

Generalized GRASP

A · · · · A

Sequence of customers visited

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

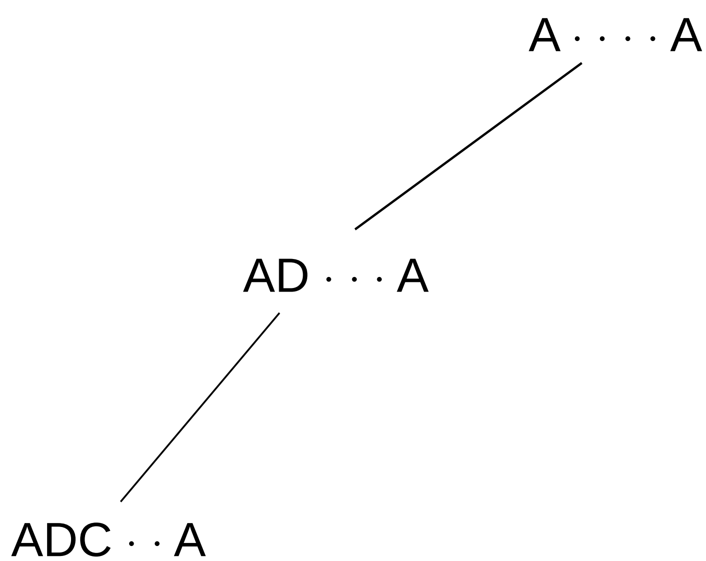
Basically,

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

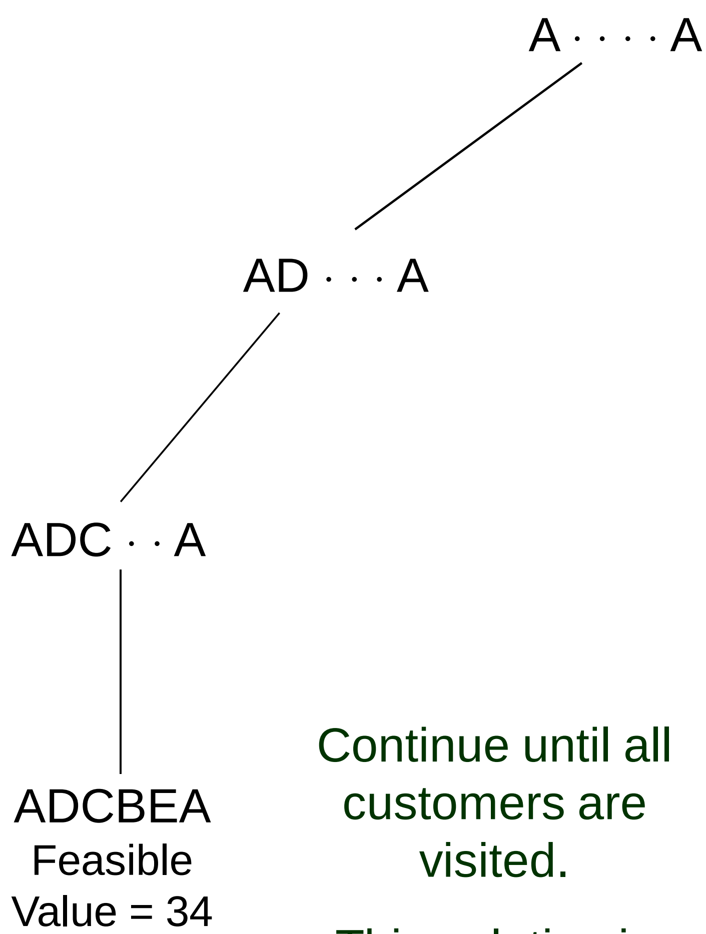
Basically,

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

Local
search
phase

A · · · · A

AD · · · A

Backtrack
randomly

ADC · · A

ADCBEA
Feasible
Value = 34

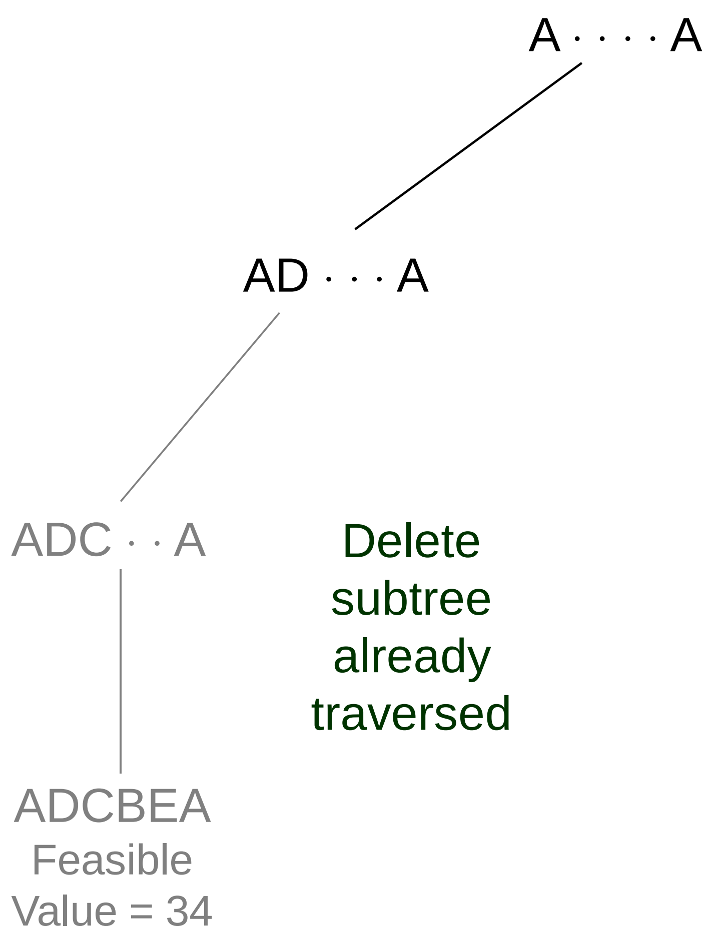
Basically,

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

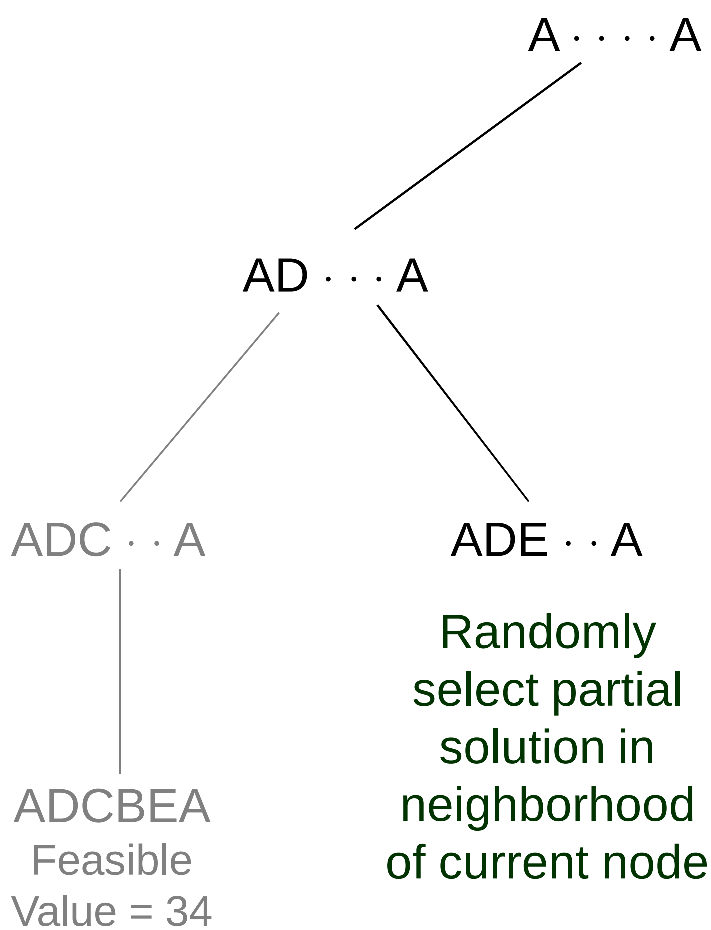
Basically,

GRASP = greedy solution + local search

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

Generalized GRASP

Local search phase

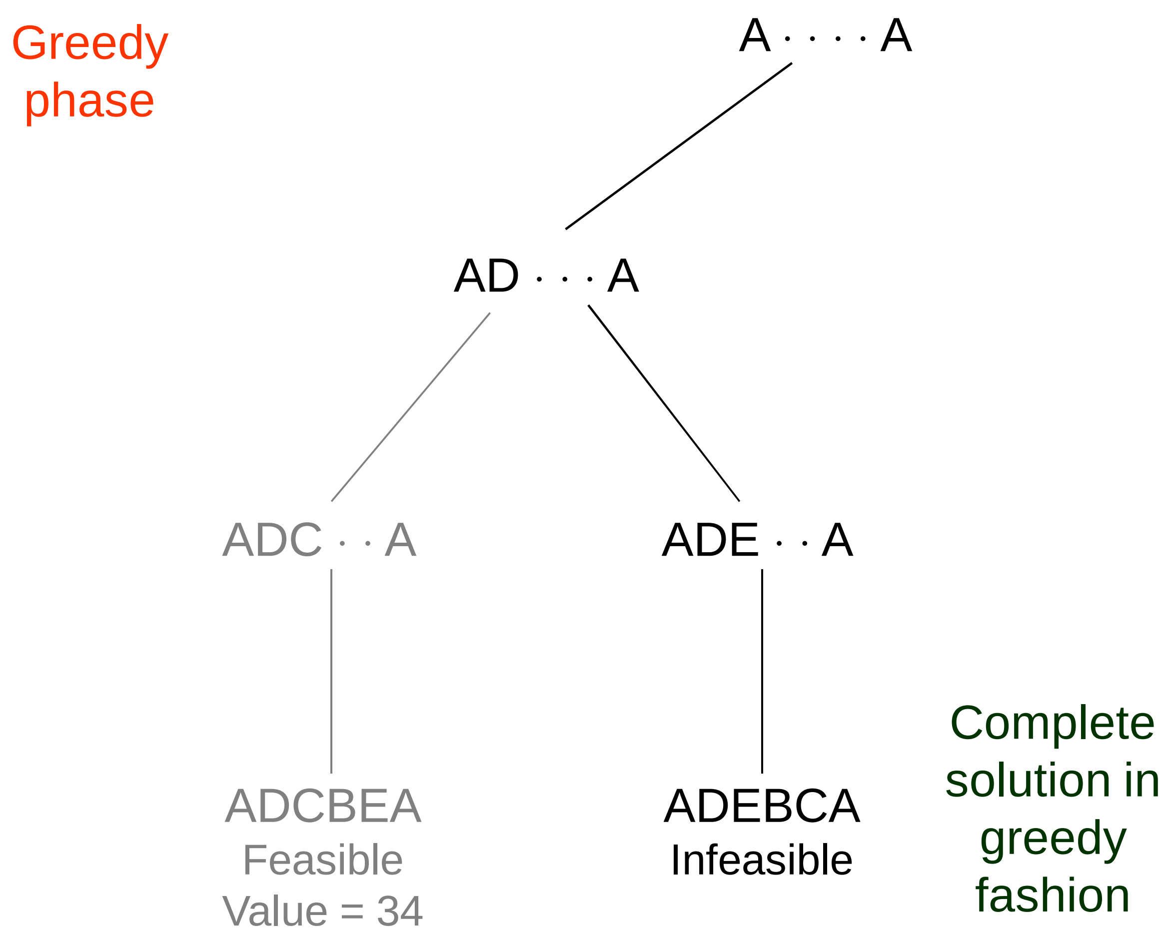


Basically,

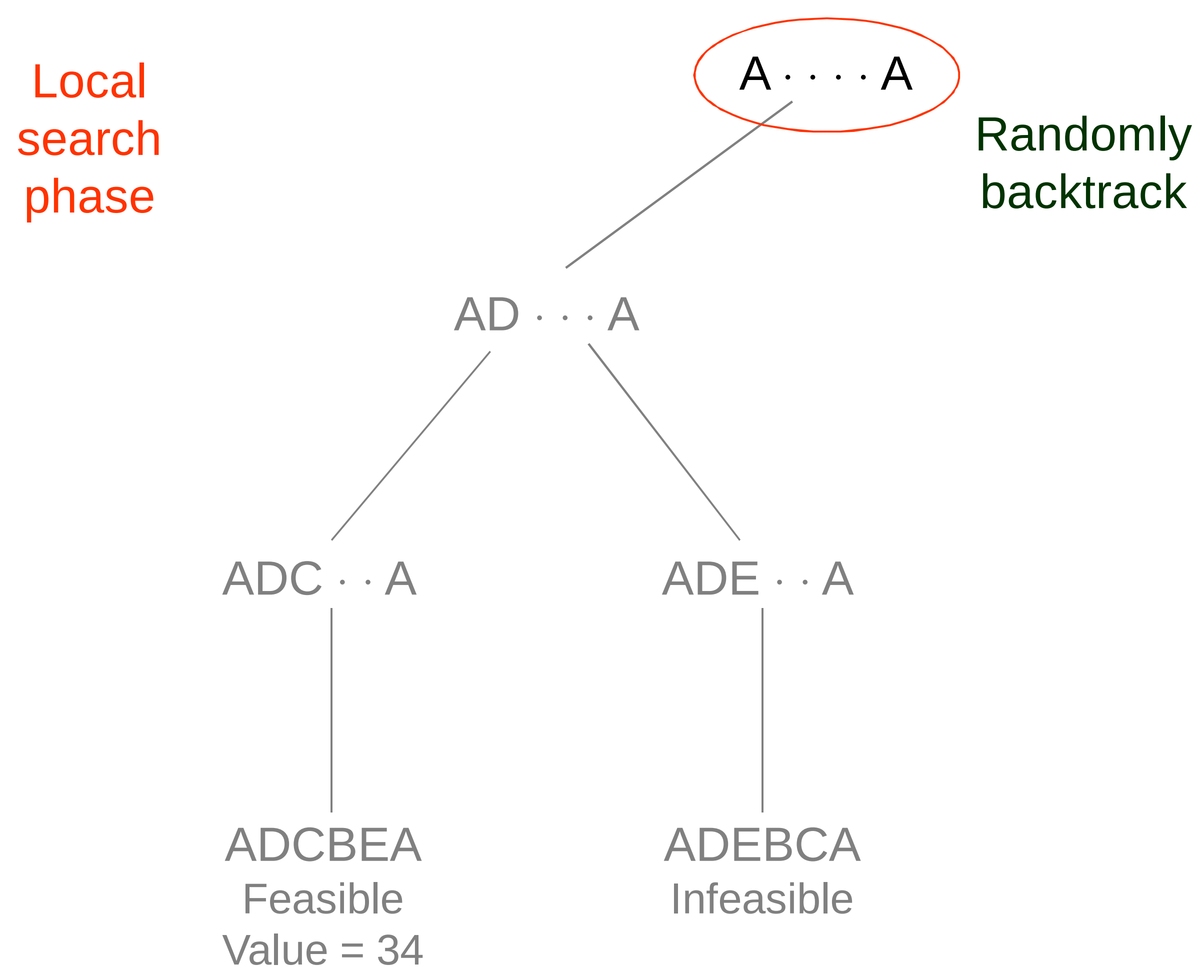
GRASP = greedy solution + local search

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

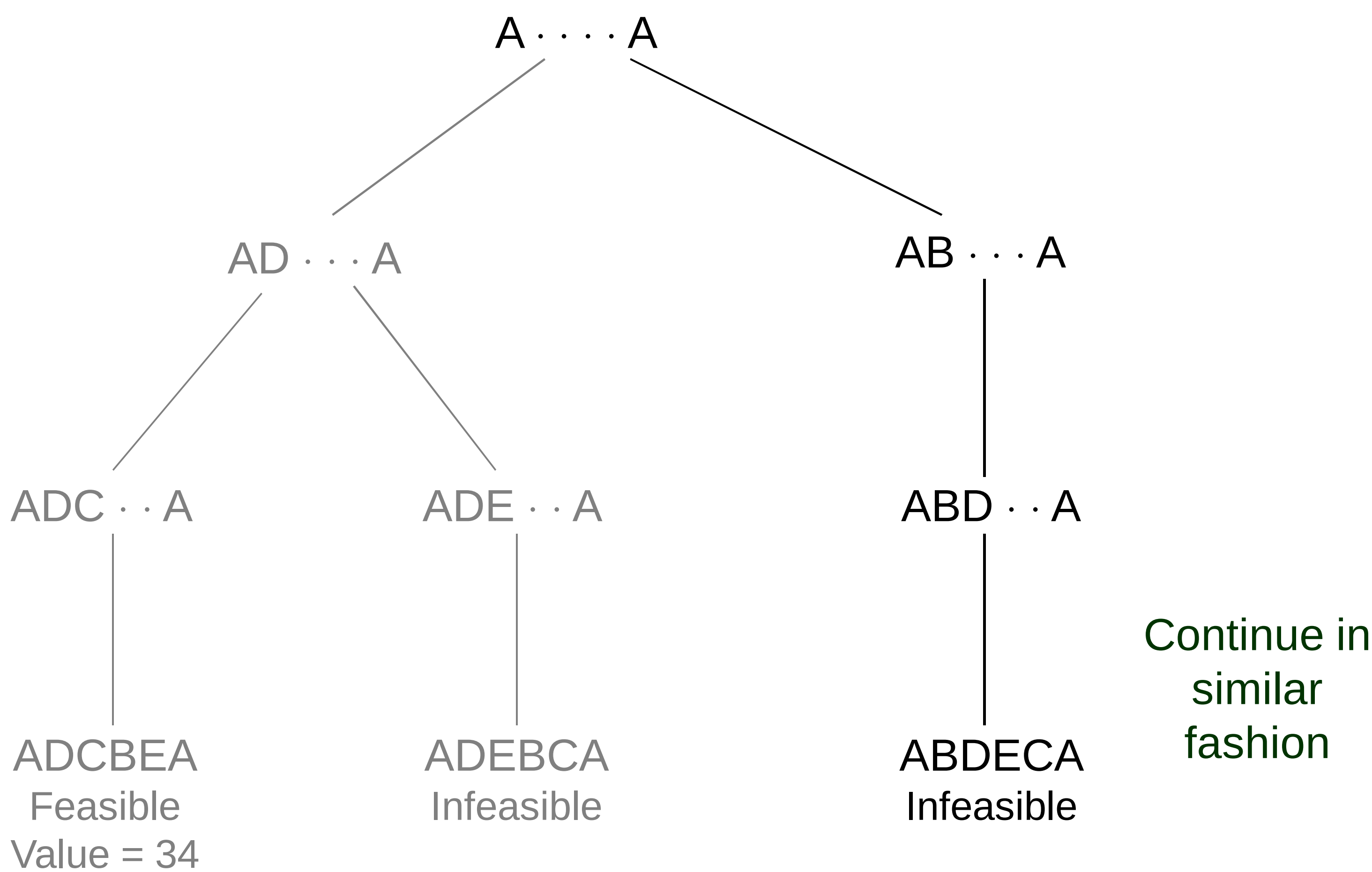
Generalized GRASP



Generalized GRASP



Generalized GRASP



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

Generalized GRASP with relaxation

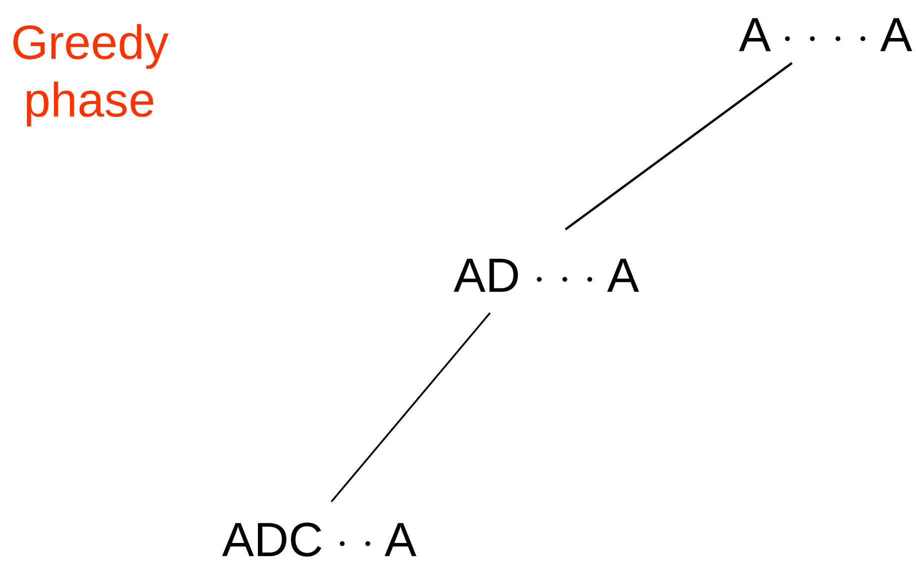
Greedy phase

A · · · · A

AD · · · A

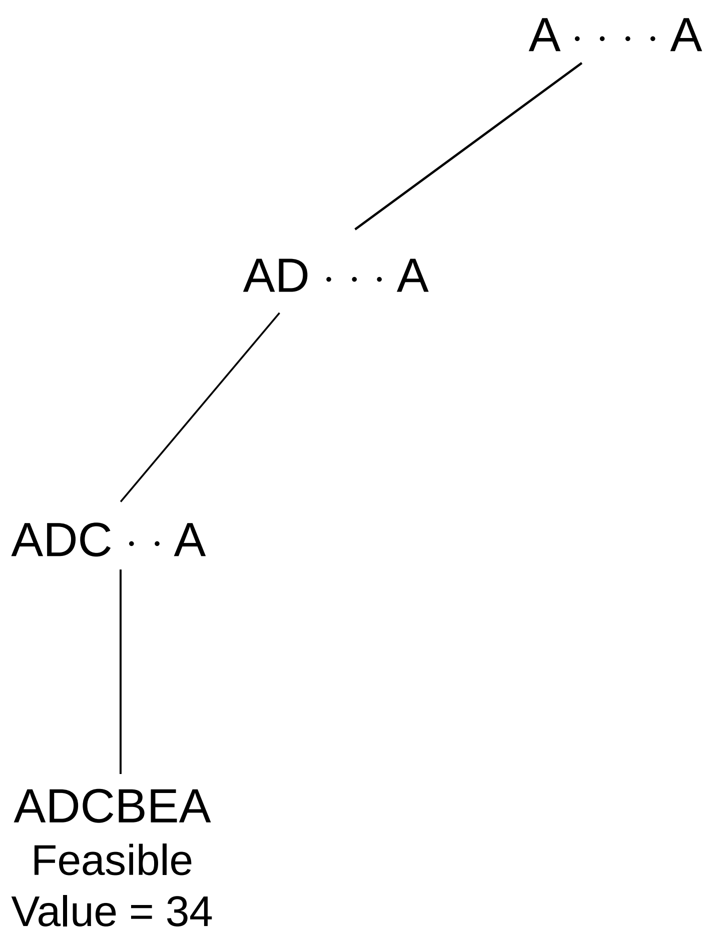
Exhaustive search suggests an improvement on a heuristic algorithm: use relaxation bounds to reduce the search.

Generalized GRASP with relaxation

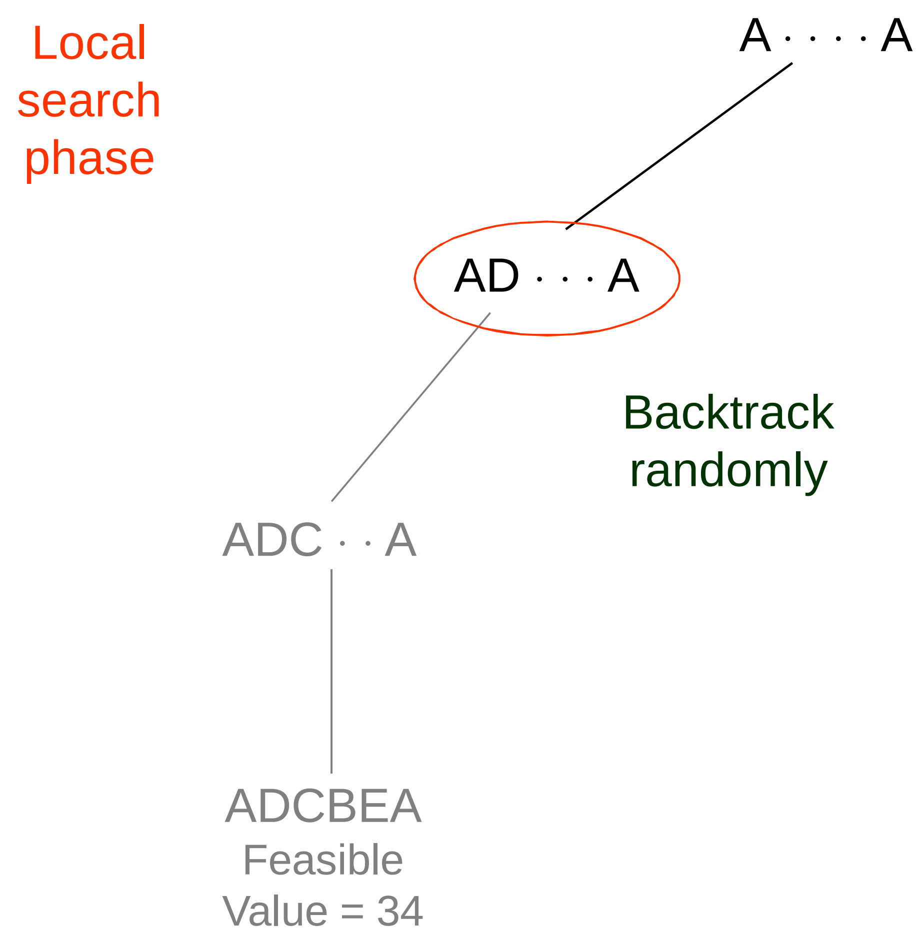


Generalized GRASP with relaxation

Greedy phase

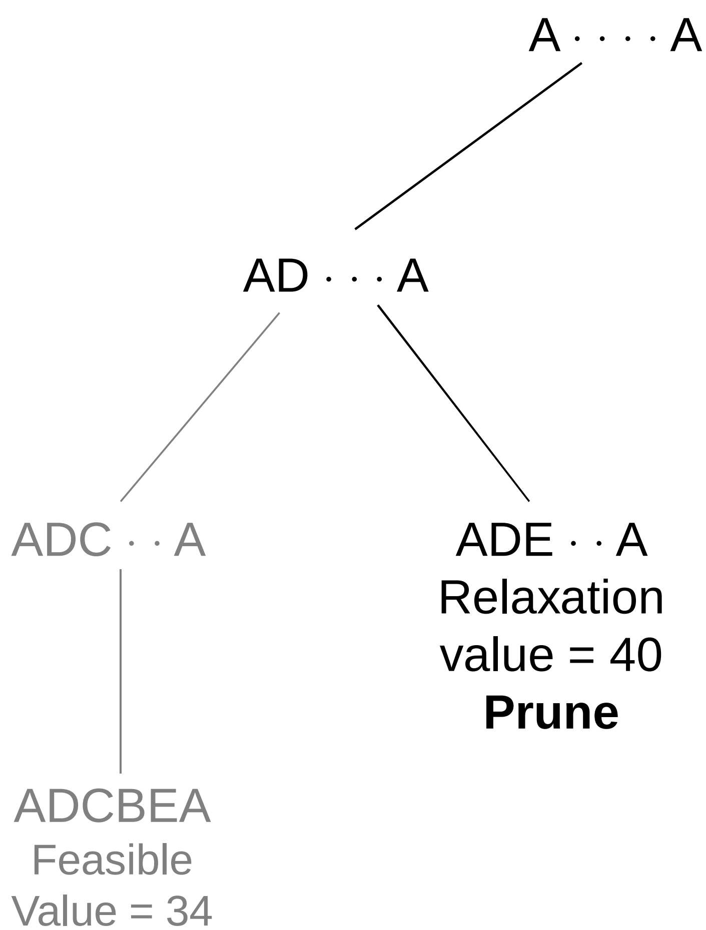


Generalized GRASP with relaxation

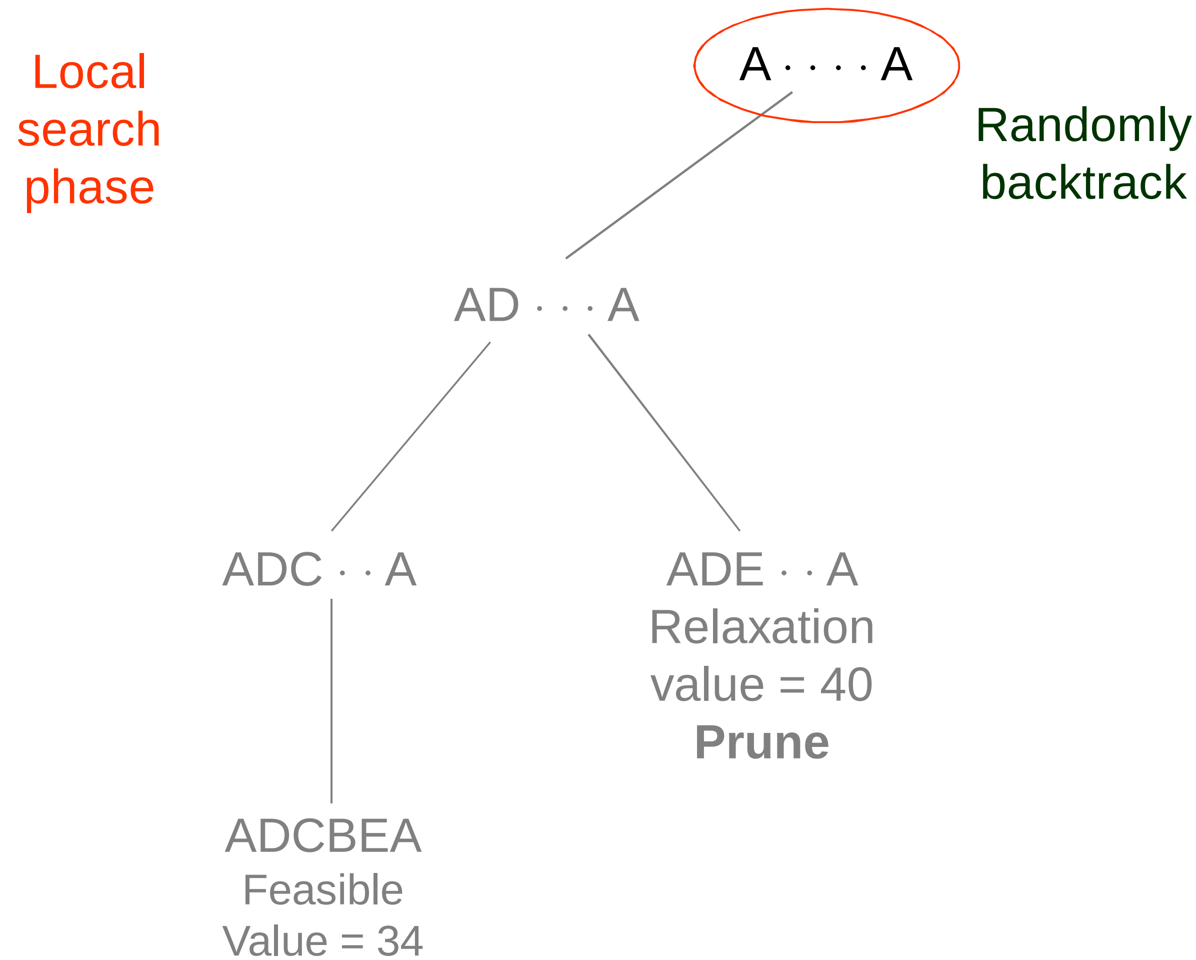


Generalized GRASP

Local search phase

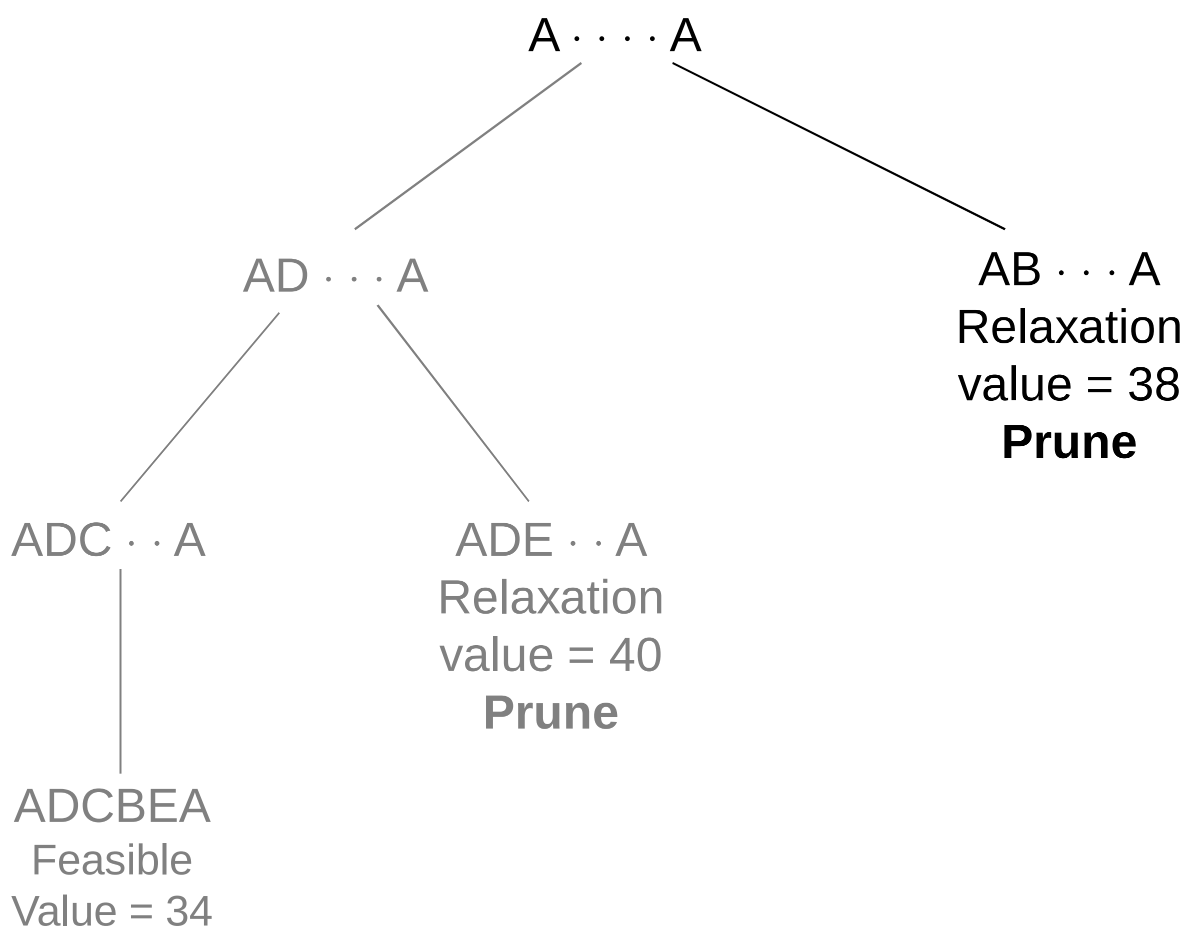


Generalized GRASP



Generalized GRASP

Local search phase



Algorithm	*Restriction P*	*P easy enough to solve?*	*relax(P)*
Branch and bound for TSPTW (exhaustive)	Created by fixing next variable	Never. Always solve *relax*(*P*).	TSPTW relaxation
Generalized GRASP for TSPTW (inexhaustive) *Greedy phase*	Created by assigning greedy value to next variable.	Never. Always solve *relax*(*P*).	TSPTW relaxation.
Local search phase	Randomly backtrack to previous node.	Yes. Randomly select value for next variable.	Not used.

Nogood-Based Search

- Search is directed by *nogoods*.
  - Nogood = constraint that excludes solutions already examined (explicitly or implicitly).
- Next solution examined is solution of current nogood set.
  - Nogoods may be *processed* so that nogood set is easy to solve.
- Search stops when nogood set is *complete* in some sense.

Nogood-Based Search Algorithm

- Let N be the set of nogoods, initially empty.
- Repeat while N is *incomplete*:
  - Select a restriction P of the original problem.
  - Select a solution x of *relax*(P) $\cup$ N.
  - If x is feasible for P then
    - If x is the best solution so far, keep it.
    - Add to N a nogood that excludes x and perhaps other solutions that are no better.
  - Else add to N a nogood that excludes x and perhaps other solutions that are infeasible.
  - *Process* nogoods in N.

Nogood-Based Search Algorithm

- To process the nogood set N:
  - Infer new nogoods from existing ones.
    - Delete (redundant) nogoods if desired.
  - Goal: make it easy to find feasible solution of N.

Nogood-Based Search Algorithm

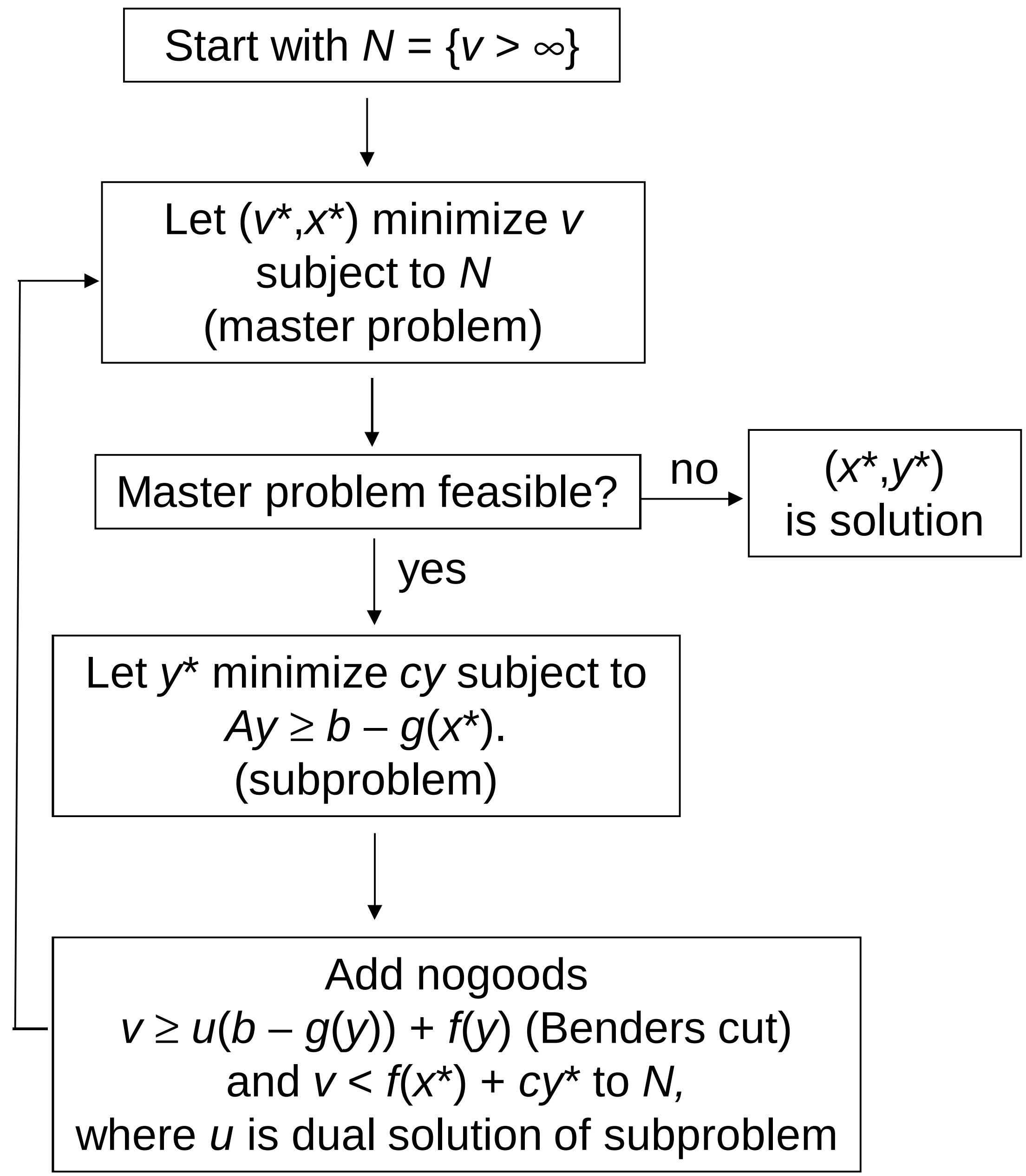
- To process the nogood set N:
  - Infer new nogoods from existing ones.
    - Delete (redundant) nogoods if desired.
  - Goal: make it easy to find feasible solution of N.
- Exhaustive vs. heuristic algorithm.
  - In an exhaustive search, complete = infeasible.
  - In a heuristic algorithm, complete = large enough.

Exhaustive search: Benders decomposition

Minimize $f(x) + cy$ subject to $g(x) + Ay \geq b$.

N = master problem constraints.
$relax(P) = \emptyset$

Nogoods are Benders cuts.
They are not processed.
N is complete when infeasible.



Exhaustive search: Benders decomposition

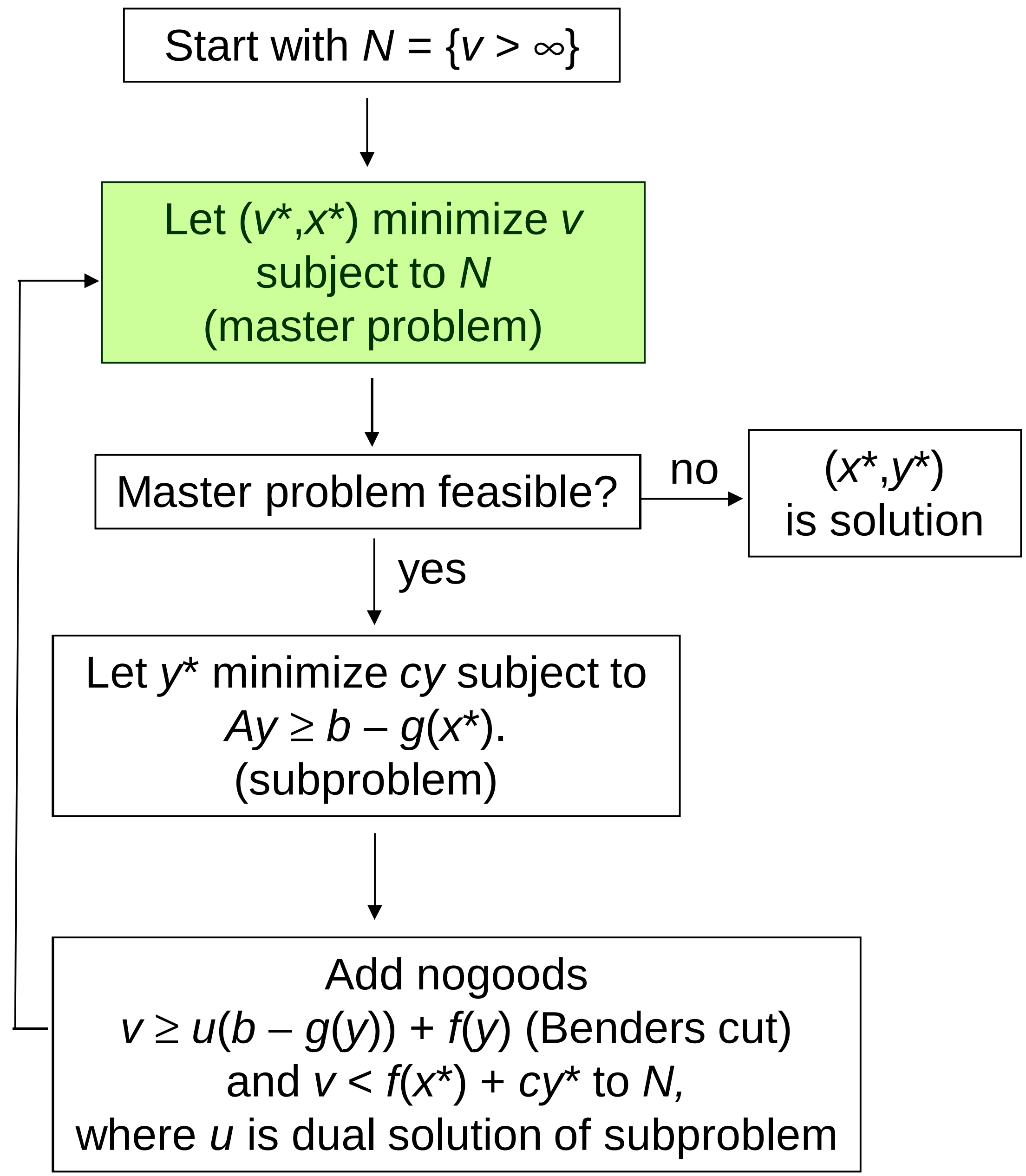
Minimize $f(x) + cy$ subject to $g(x) + Ay \geq b$.

N = master problem constraints.
$relax(P) = \emptyset$

Nogoods are Benders cuts.
They are not processed.
N is complete when infeasible.

Select optimal solution of N.

Formally, selected solution is (x^*, y) where y is arbitrary



Exhaustive search: Benders decomposition

Minimize $f(x) + cy$ subject to $g(x) + Ay \geq b$.

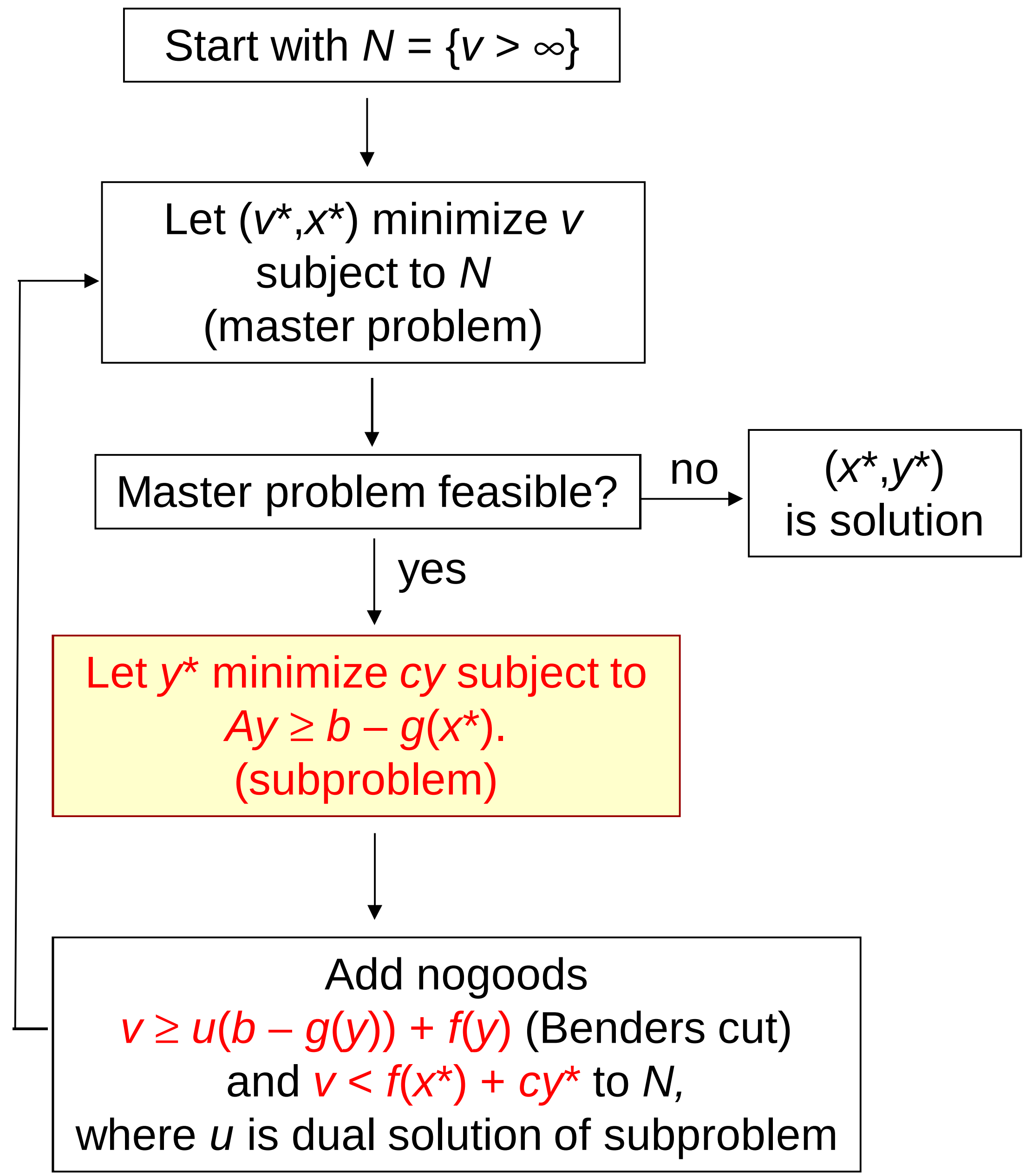
N = master problem constraints.
$relax(P) = \emptyset$

Nogoods are Benders cuts.
They are not processed.
N is complete when infeasible.

Select optimal solution of N.

Formally, selected solution is (x^*, y) where y is arbitrary

Subproblem generates nogoods.



Algorithm	*Restriction P*	*Relax(P)*	*Nogoods*	*Nogood processing*	*Solution of relax(P) ∪ N*
Benders decomposition (exhaustive)	Same as original problem.	No constraints	Benders cuts, obtained by solving subproblem	None	Optimal solution of master problem + arbitrary values for subproblem variables
Tabu search (inexhaustive)					
Partial-order dynamic backtracking for TSPTW (exhaustive)					
Partial-order dynamic backtracking for TSPTW (inexhaustive)					

Nogood-Based Search Algorithm

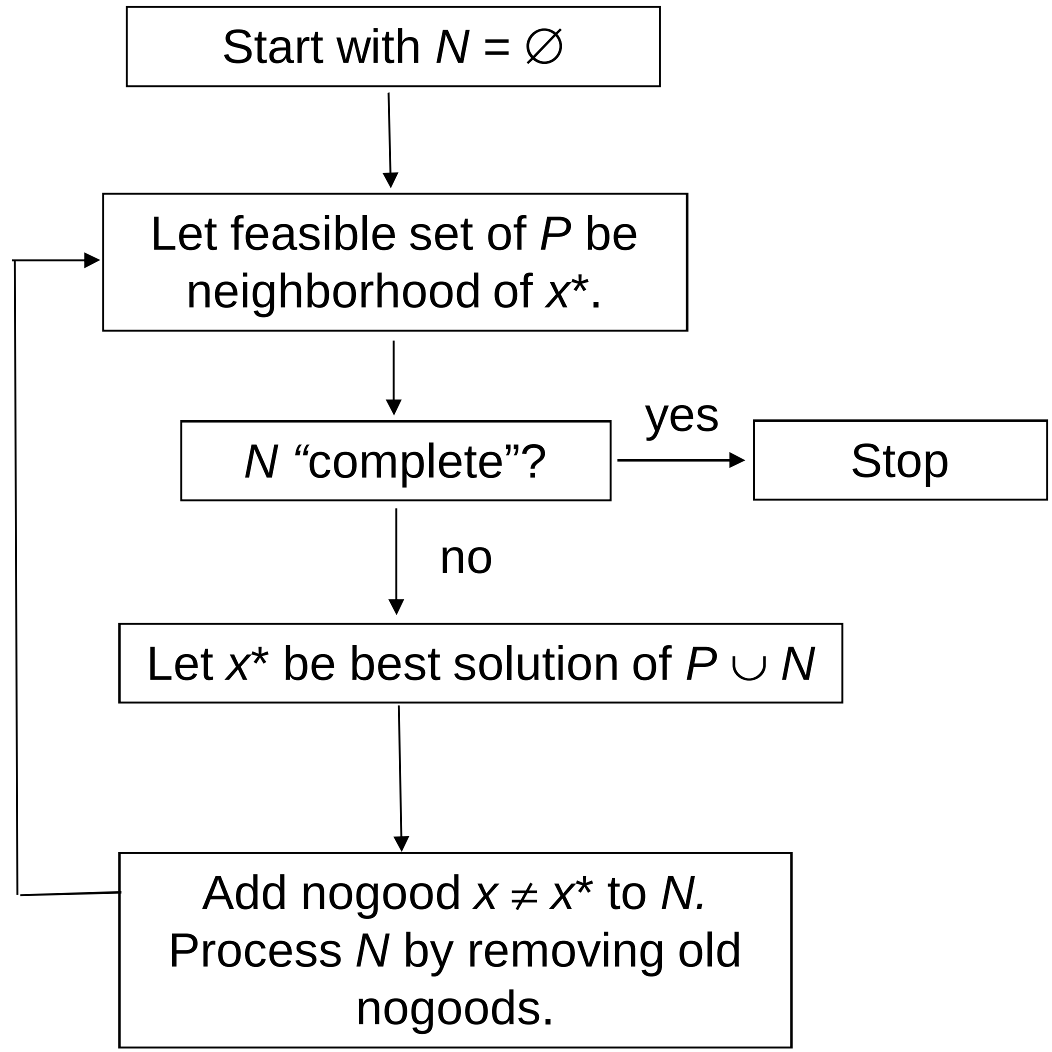
- Other forms of exhaustive nogood-based search:
  - Davis-Putnam-Loveland method for with clause learning (for propositional satisfiability problem).
  - Partial-order dynamic backtracking.

Heuristic algorithm: Tabu search

The nogood set N is the tabu list.

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

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



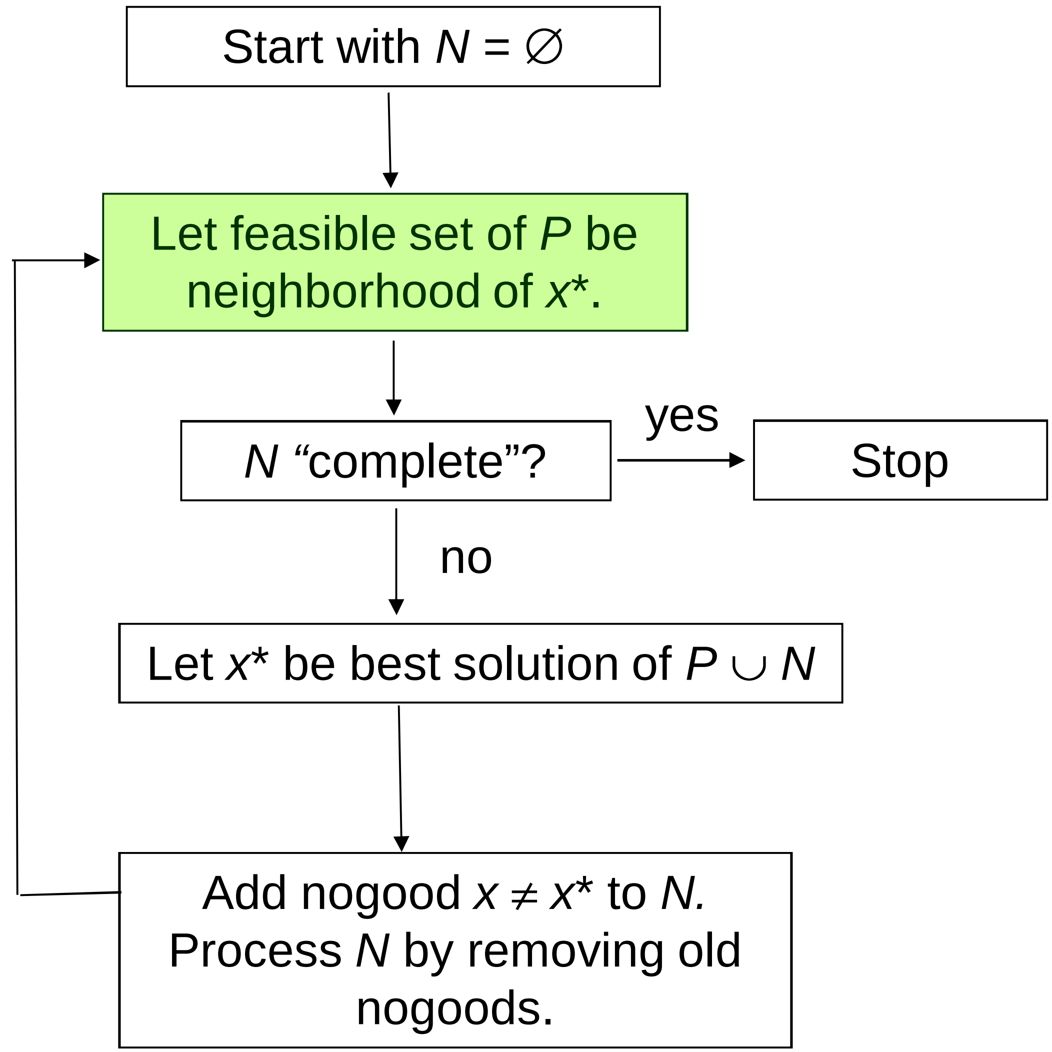
Heuristic algorithm: Tabu search

The nogood set N is the tabu list.

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

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

Neighborhood of current solution x^* is *feas*(*relax*(*P*)).



Heuristic algorithm: Tabu search

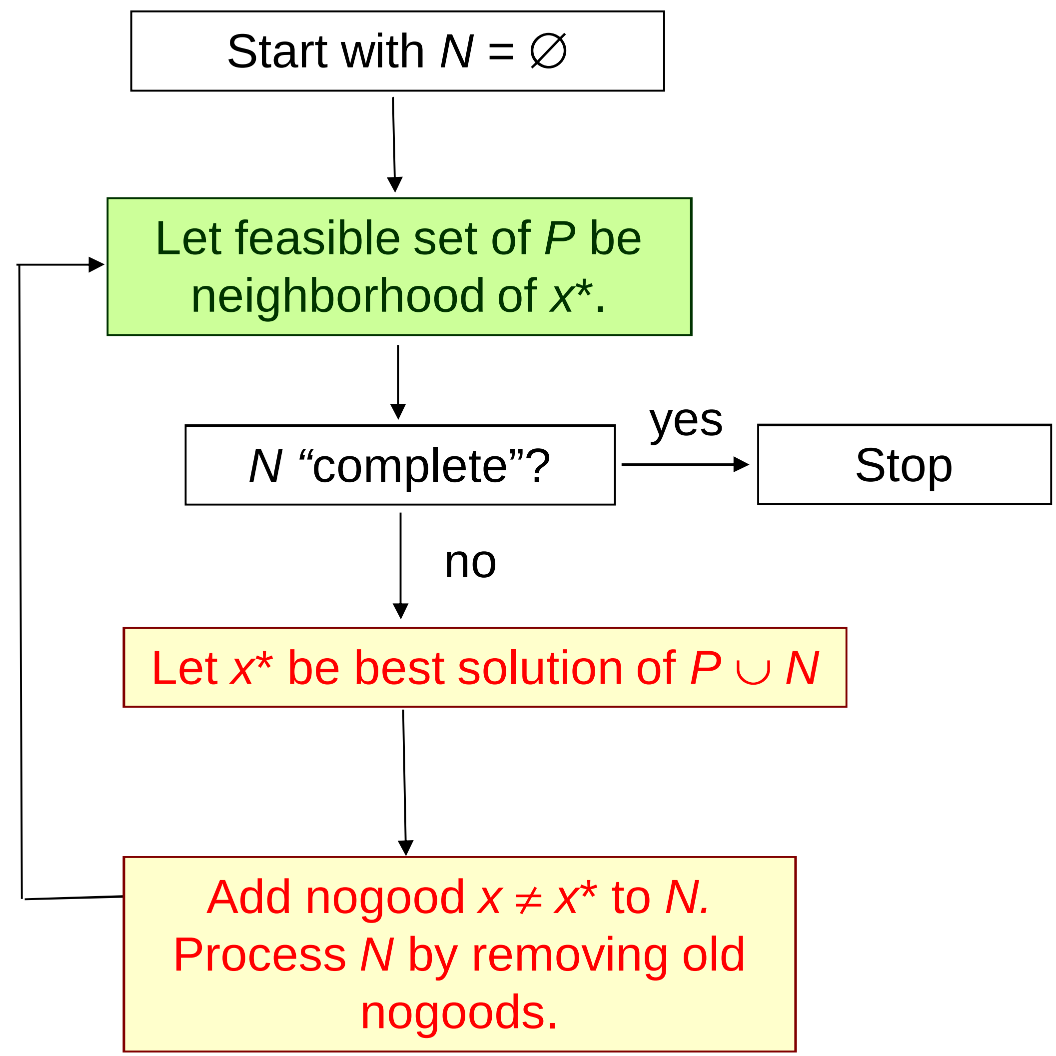
The nogood set N is the tabu list.

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

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

Neighborhood of current solution x^* is *feas*(*relax*(*P*)).

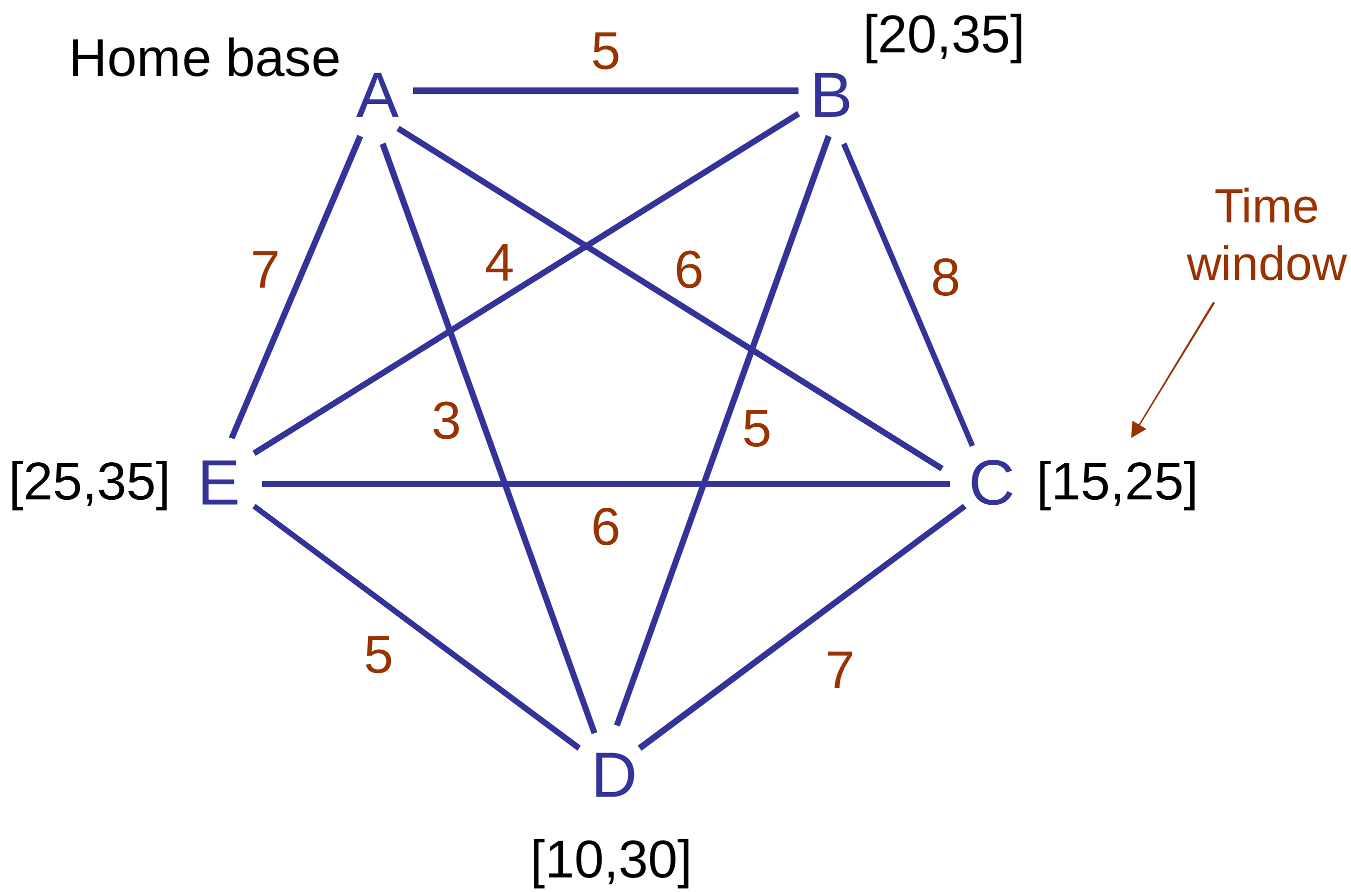
Solve $P \cup N$ by searching neighborhood.
Remove old nogoods from tabu list.



Algorithm	*Restriction P*	*Relax(P)*	*Nogoods*	*Nogood processing*	*Solution of relax(P) $\cup N$*
Benders decomposition (exhaustive)	Same as original problem.	No constraints	Benders cuts, obtained by solving subproblem	None	Optimal solution of master problem + arbitrary values for subproblem variables
Tabu search (inexhaustive)	Created by defining neighborhood of previous solution	Same as P	Tabu list	None	Find best solution in neighborhood not on tabu list.
Partial-order dynamic backtracking for TSPTW (exhaustive)					
Partial-order dynamic backtracking for TSPTW (inexhaustive)					

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

In this problem, *P* is original problem, and *relax*(*P*) has no constraints.

So $relax(P) \cup N = N$

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.

The current nogoods ADCB, ADCE rule out any solution beginning ADC.

So *process* the nogood set by replacing ADCB, ADCE with their *parallel resolvent* ADC.

This makes it possible to solve the nogood set with a 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 we observe that 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

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			

Optimal solution.

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

Algorithm	*Restriction P*	*Relax(P)*	*Nogoods*	*Nogood processing*	*Solution of relax(P)* $\cup$ *N*
Benders decomposition (exhaustive)	Same as original problem.	No constraints	Benders cuts, obtained by solving subproblem	None	Optimal solution of master problem + arbitrary values for subproblem variables
Tabu search (inexhaustive)	Created by defining neighborhood of previous solution	Same as P	Tabu list	Delete old nogoods.	Find best solution in neighborhood not on tabu list.
Partial-order dynamic backtracking for TSPTW (exhaustive)	Same as original problem	No constraints	Rule out last sequence tried	Parallel resolution	Greedy solution.
Partial-order dynamic backtracking for TSPTW (inexhaustive)					

Inexhaustive 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. So method is inexhaustive

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.

Inexhaustive 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.

Inexhaustive 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
$\vdots$				

Continue in this fashion, but start dropping old nogoods.

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

Stopping point is arbitrary.

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

Algorithm	*Restriction P*	*Relax(P)*	*Nogoods*	*Nogood processing*	*Solution of relax(P) $\cup$ N*
Benders decomposition (exhaustive)	Same as original problem.	No constraints	Benders cuts, obtained by solving subproblem	None	Optimal solution of master problem + arbitrary values for subproblem variables
Tabu search (inexhaustive)	Created by defining neighborhood of previous solution	Same as P	Tabu list	Delete old nogoods.	Find best solution in neighborhood not on tabu list.
Partial-order dynamic backtracking for TSPTW (exhaustive)	Same as original problem	No constraints	Rule out last sequence tried	Parallel resolution	Greedy solution.
Partial-order dynamic backtracking for TSPTW (inexhaustive)	Same as original problem	No constraints	Rule out last seq. & infeasible sub-sequences	Full resolution, delete old nogoods.	Greedy solution.