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 ἐυριστέιν (to search, to find).
• Two classes of exhaustive search methods are very similar to corresponding local search methods:
  – Branching methods.
  – Nogood-based search.
<table>
<thead>
<tr>
<th>Type of search</th>
<th>Exhaustive search examples</th>
<th>Local search examples</th>
<th>Nogood-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching</td>
<td>- Branch and bound</td>
<td>- Simulated annealing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DPL for SAT</td>
<td>- GRASP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Benders decomposition</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- DPL with clause learning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Partial order</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Dynamic backtracking</td>
<td></td>
</tr>
</tbody>
</table>
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:
  - $\text{feas}(P)$ = feasible set of $P$
  - $\text{relax}(P)$ = 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 $\text{relax}(P)$ is better than previous best solution, then
      - If solution of $\text{relax}(P)$ 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.”

Currently at this leaf node

Previously removed nodes

Previously removed nodes
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 $\text{relax}(P)$ is better than previous solution and $P_1, \ldots, P_k$ are not exhaustive.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction $P$</th>
<th>$P$ easy enough to solve?</th>
<th>$\text{relax}(P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and bound (exhaustive)</td>
<td>Created by splitting variable domain</td>
<td>Never. Always solve $\text{relax}(P)$</td>
<td>LP relaxation</td>
</tr>
<tr>
<td>Simulated annealing (inexhaustive)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRASP (inexhaustive)</td>
<td>Greedy phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local search phase</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Heuristic algorithm: Simulated annealing

Search tree has 2 levels.

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

Currently at this leaf node, which was generated because \( \{P_1, \ldots, P_k\} \) is not complete
Heuristic algorithm: Simulated annealing

Currently at this leaf node, which was generated because \{P_1, \ldots, P_k\} is not complete

\[ feas(P) = \text{neighborhood of } x \]

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.

Previously removed nodes
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction</th>
<th>$P$ easy enough to solve?</th>
<th>$\text{relax}(P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and bound (exhaustive)</td>
<td>Created by splitting variable domain</td>
<td>Never. Always solve $\text{relax}(P)$</td>
<td>LP relaxation</td>
</tr>
<tr>
<td>GRASP (inexhaustive)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greedy phase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local search phase</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 = $v$. 

$x_1 = v_1$

$x_2 = v_2$

$x_3 = v_3$
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 = \(v\).

Local search phase

\[ \text{feas}(P) = \text{neighborhood of } v \]
Heuristic algorithm: GRASP

Greedy randomized adaptive search procedure

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

"Hard" to solve. \( \text{relax}(P) \) contains no constraints

"Easy" to solve. Solution = \( \nu \).

Stop local search when "complete" set of neighborhoods have been searched. Process now starts over with new greedy phase.

 feas(\( P_2 \)) = neighborhood of \( \nu \)
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Branch and bound (exhaustive)</th>
<th>Simulated annealing (inexhaustive)</th>
<th>GRASP (inexhaustive) Greedy phase</th>
<th>Local search phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction ( P )</td>
<td>Created by splitting variable domain</td>
<td>Created by defining neighborhood of previous solution</td>
<td>Created by assigning next variable a randomized greedy value.</td>
<td>Created by defining neighborhood of previous solution.</td>
</tr>
<tr>
<td>relax(( P ))</td>
<td>LP relaxation</td>
<td>Not used.</td>
<td>No constraints, so solution is always infeasible and branching necessary.</td>
<td>Not used.</td>
</tr>
</tbody>
</table>
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

Home base

Time window
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:

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

- $T$: Earliest time vehicle can leave customer $k$
- Min time from customer $j$'s predecessor to $j$
- Min time from last customer back to home
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

A \ldots A

Sequence of customers visited
Exhaustive Branch-and-Bound
Exhaustive Branch-and-Bound
Exhaustive Branch-and-Bound

Feasible Value = 36
Exhaustive Branch-and-Bound

Feasible Value = 36

Feasible Value = 34
Exhaustive Branch-and-Bound

A \cdots A

A \cdots A

ADC \cdots A

ADB \cdots A

Relaxation value = 36

Prune

ADCEBA Feasible Value = 36

ADCEBA Feasible Value = 34

AC \cdots A

AB \cdots A

AE \cdots A
Exhaustive Branch-and-Bound

A A

AC A AB A AE A

ADC

Relaxation value = 36

Prune

Feasible Value = 34

ADE A

Relaxation value = 40

Prune

A DB A

Relaxation value = 36

Prune

AD C A

ADCBEA

Feasible Value = 36

A D CBEA

Feasible Value = 36
Exhaustive Branch-and-Bound

A · · · A

Relaxation value = 31

AC · · · A
Relaxation value = 31

Prune

AB · · · A

Prune

AE · · · A

Continue in this fashion

A · · · A

AD · · · A

ADC · · A

Relaxation value = 36

Prune

ADE · · A
Relaxation value = 40

Prune

ADCBEA Feasible Value = 36

ADCEBA Feasible Value = 34
Exhaustive Branch-and-Bound

Optimal solution
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>P easy enough to solve?</th>
<th>Restriction P</th>
<th>relax(P)</th>
<th>TSPTW relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and bound for TSPTW (exhaustive)</td>
<td>Never. Always solve relax(P).</td>
<td>Created by fixing next variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generalized GRASP for TSPTW (inexhaustive)</td>
<td>Greedy phase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local search phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generalized GRASP

Basically, GRASP = greedy solution + local search

Begin with greedy assignments that can be viewed as creating “branches”
Basically, GRASP = greedy solution + local search

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

Visit customer than can be served earliest from A

Greedy phase

Generalized GRASP
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

A ⋅ ⋅ ⋅ A

AD ⋅ ⋅ A

ADC ⋅ A

ADCBEA
Feasible Value = 34

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

Feasible Value = 34

Basically,
GRASP = greedy solution + local search
Begin with greedy assignments that can be viewed as creating “branches”

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

Backtrack randomly

Local search phase
Generalized GRASP

- Local search phase
- AD · ⋅ ⋅ A
- Delete subtree already traversed
- ADC · A
- ADCAEA
  - Feasible Value = 34

Basically, GRASP = greedy solution + local search

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

Algorithm:
1. Randomly select partial solution in neighborhood of current node
2. Local search phase
3. Basically, GRASP = greedy solution + local search
4. Begin with greedy assignments that can be viewed as creating “branches”

Feasible Value = 34
Generalized GRASP

Greedy phase

Complete solution in greedy fashion
Generalized GRASP

Local search phase

Randomly backtrack

Feasible Value = 34

Infeasible
Exhaustive search algorithm (branching search) suggests a generalization of a heuristic algorithm (GRASP).
Generalized GRASP with relaxation

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

Greedy phase

A · · · A

AD · · A

ADC · · A
Generalized GRASP with relaxation

Greedy phase

Feasible Value = 34
Generalized GRASP with relaxation

Local search phase

Backtrack randomly

Feasible Value = 34
Generalized GRASP

Local search phase

ADC ⋅ ⋅ A

AD ⋅ ⋅ A

A ⋅ ⋅ ⋅ A

A DC ⋅ ⋅ A

A DCBEA

Feasible Value = 34

ADE ⋅ ⋅ A

Relaxation value = 40

Prune
Generalized GRASP

Local search phase

Randomly backtrack

AD ⋅ ⋅ ⋅ A

AD ⋅ ⋅ A

ADC ⋅ ⋅ A

ADCBEA

Feasible Value = 34

A ⋅ ⋅ ⋅ A

ADE ⋅ ⋅ A

Relaxation value = 40

Prune
Generalized GRASP

Local search phase

- AD ⋅ ⋅ A
  - ADC ⋅ A
    - ADCBEA
      - Feasible Value = 34
  - ADE ⋅ A
    - Relaxation value = 40
      - Prune
- AB ⋅ ⋅ A
  - Relaxation value = 38
    - Prune
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction $P$</th>
<th>$P$ easy enough to solve?</th>
<th>$\text{relax}(P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and bound for TSPTW</td>
<td>Created by fixing next variable</td>
<td>Never. Always solve $\text{relax}(P)$.</td>
<td>TSPTW relaxation.</td>
</tr>
<tr>
<td>Generalized GRASP for TSPTW</td>
<td>Created by assigning greedy value to next variable.</td>
<td>Never. Always solve $\text{relax}(P)$.</td>
<td>TSPTW relaxation.</td>
</tr>
<tr>
<td>Greedy phase</td>
<td>Randomly backtrack to previous node.</td>
<td>Yes. Randomly select value for next variable.</td>
<td>Not used.</td>
</tr>
<tr>
<td>Local search phase</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 $\text{relax}(P) \cup N$.
  – 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 $= \text{infeasible}$.
  – In a heuristic algorithm, complete $= \text{large enough}$.
Exhaustive search: Benders decomposition

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

$N = \text{master problem constraints.}$

$relax(P) = \emptyset$

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

Start with $N = \{v > \infty\}$

Let $(v^*, x^*)$ minimize $v$ subject to $N$ (master problem)

Master problem feasible? no $(x^*, y^*)$ is solution

Master problem feasible? yes

Let $y^*$ minimize $cy$ subject to $Ay \geq b - g(x^*)$. (subproblem)

Add nogoods $v \geq u(b - g(y)) + f(y)$ (Benders cut) and $v < f(x^*) + cy^*$ to $N$, where $u$ is dual solution of subproblem.
Exhaustive search: Benders decomposition

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

$N = \text{master problem constraints.} \quad \text{relax}(P) = \emptyset$

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.

Start with $N = \{v > \infty\}$

Let $(v^*, x^*)$ minimize $v$ subject to $N$ (master problem)

Master problem feasible? no $(x^*, y^*)$ is solution

yes

Let $y^*$ minimize $cy$ subject to $Ay \geq b - g(x^*)$ (subproblem)

Add nogoods $v \geq u(b - g(y)) + f(y)$ (Benders cut) and $v < f(x^*) + cy^*$ to $N$, where $u$ is dual solution of subproblem
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.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Solution of ( \text{relax}(P) \cup N )</th>
<th>Nogood processing</th>
<th>Nogoods</th>
<th>Relax(( P ))</th>
<th>Restriction ( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optimal solution of master problem + arbitrary values for subproblem variables</td>
<td>None</td>
<td>Benders cuts, obtained by solving subproblem</td>
<td>No constraints on ( P )</td>
<td></td>
</tr>
<tr>
<td>Benders decomposition (exhaustive)</td>
<td>Same as original problem.</td>
<td></td>
<td></td>
<td>Benders decomposition (exhaustive)</td>
<td>Partial-order dynamic backtracking for TSPTW (exhaustive)</td>
</tr>
<tr>
<td></td>
<td>Tabu search (inexhaustive)</td>
<td></td>
<td></td>
<td>Tabu search (inexhaustive)</td>
<td>Partial-order dynamic backtracking for TSPTW (inexhaustive)</td>
</tr>
<tr>
<td></td>
<td>Partial-order dynamic backtracking for TSPTW (inexhaustive)</td>
<td></td>
<td></td>
<td>Partial-order dynamic backtracking for TSPTW (inexhaustive)</td>
<td></td>
</tr>
</tbody>
</table>
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 $\text{feas}(\text{relax}(P))$.

- Start with $N = \emptyset$
- Let feasible set of $P$ be neighborhood of $x^*$.
- Let $x^*$ be best solution of $P \cup N$.
- Add nogood $x \neq x^*$ to $N$. Process $N$ by removing old nogoods.
- $N$ “complete”? 
  - yes: Stop
  - no: Process $N$ by removing old nogoods.
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 $\text{feas}(\text{relax}(P))$.

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

Start with $N = \emptyset$

Let feasible set of $P$ be neighborhood of $x^*$.

$N$ “complete”? yes Stop

no

Let $x^*$ be best solution of $P \cup N$

Add nogood $x \neq x^*$ to $N$.
Process $N$ by removing old nogoods.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction $P$</th>
<th>Relax($P$)</th>
<th>Nogoods</th>
<th>Nogoods processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benders decomposition (exhaustive)</td>
<td>Same as original problem.</td>
<td>No constraints</td>
<td>Benders cuts, obtained by solving subproblem</td>
<td>None</td>
</tr>
<tr>
<td>Tabu search (inexhaustive)</td>
<td>Created by defining neighborhood of previous solution</td>
<td></td>
<td>Tabu list</td>
<td></td>
</tr>
<tr>
<td>Partial-order dynamic backtracking for TSPTW (exhaustive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial-order dynamic backtracking for TSPTW (inexhaustive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example

TSP with Time Windows
**Exhaustive nogood-based search**

<table>
<thead>
<tr>
<th>Iter.</th>
<th>( \text{relax}(P) \cup N )</th>
<th>Solution of ( N )</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADCB</td>
<td>ADCB</td>
<td>36</td>
<td>ADCB</td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this problem, \( P \) is original problem, and \( \text{relax}(P) \) has no constraints. So \( \text{relax}(P) \cup N = N \).

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

Excludes current solution by excluding any solution that begins ADCB.
### Exhaustive nogood-based search

<table>
<thead>
<tr>
<th>Iter.</th>
<th>$relax(P) \cup N$</th>
<th>Solution of $N$</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADCB, ADCE</td>
<td>ADCB, ADCEBA</td>
<td>36</td>
<td>ADCB</td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCEBA</td>
<td>34</td>
<td>ADCE</td>
</tr>
<tr>
<td>2</td>
<td>ADC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Iter.</th>
<th>$relax(P) \cup N$</th>
<th>Solution of $N$</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADCBEA</td>
<td>36</td>
<td>ADCB</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCEBA</td>
<td>34</td>
<td>ADCE</td>
</tr>
<tr>
<td>2</td>
<td>ADC</td>
<td>ADBEAC</td>
<td>infeasible</td>
<td>ADB</td>
</tr>
<tr>
<td>3</td>
<td>ADB, ADC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not only is ADBEAC infeasible, but we observe that no solution beginning ADB can be completed within time windows.
### Exhaustive nogood-based search

<table>
<thead>
<tr>
<th>Iter.</th>
<th>$\text{relax}(P) \cup N$</th>
<th>Solution of $N$</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADCB</td>
<td>ADCB</td>
<td>36</td>
<td>ADCB</td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCEBA</td>
<td>34</td>
<td>ADCE</td>
</tr>
<tr>
<td>2</td>
<td>ADC</td>
<td>ADBEAC</td>
<td>infeasible</td>
<td>ADB</td>
</tr>
<tr>
<td>3</td>
<td>ADB, ADC</td>
<td>ADEBCA</td>
<td>infeasible</td>
<td>ADE</td>
</tr>
<tr>
<td>4</td>
<td>AD</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process nogoods ADB, ADC, ADE to obtain parallel resolvent AD.
Exhaustive nogood-based search

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

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

Optimal solution.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Restriction P</th>
<th>Relax(P)</th>
<th>Nogoods</th>
<th>Nogood processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benders decomposition (exhaustive)</td>
<td>Same as original problem.</td>
<td>Same as P</td>
<td>Benders cuts, obtained by solving subproblem</td>
<td>None</td>
</tr>
<tr>
<td>Tabu search (inexhaustive)</td>
<td>Created by defining neighborhood of previous solution</td>
<td>Tabu list</td>
<td>Tabu list</td>
<td>Parallel resolution</td>
</tr>
<tr>
<td>Partial-order dynamic backtracking for TSPTW (exhaustive)</td>
<td>Same as original problem</td>
<td>No constraints</td>
<td>Rule out last sequence tried</td>
<td></td>
</tr>
<tr>
<td>Partial-order dynamic backtracking for TSPTW (inexhaustive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Solution of $\text{relax}(P) \cup N$**
  - Optimal solution of master problem + arbitrary values for subproblem variables
  - Find best solution in neighborhood not on taboo list

- **Greedy solution**
  - Parallel resolution
Inexhaustive nogood-based search

<table>
<thead>
<tr>
<th>Iter.</th>
<th>$relax(P) \cup N$</th>
<th>Solution of $N$</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>ADCBKEA</td>
<td>36</td>
<td>ADCB</td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCEBA</td>
<td>34</td>
<td>ADCE</td>
</tr>
<tr>
<td>2</td>
<td>ADC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Iter.</th>
<th>$relax(P) \cup N$</th>
<th>Solution of $N$</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>ADCB</td>
<td>36</td>
<td>ADCB</td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCEBA</td>
<td>34</td>
<td>ADCE</td>
</tr>
<tr>
<td>2</td>
<td>ADC</td>
<td>ADBECA</td>
<td>infeasible</td>
<td>ADB, EC</td>
</tr>
<tr>
<td>3</td>
<td>ABEC, ADB, ADC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Iter.</th>
<th>( \text{relax}(P) \cup N )</th>
<th>Solution of ( N )</th>
<th>Sol. value</th>
<th>New nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADCB</td>
<td>ADCB</td>
<td>36</td>
<td>ADCB</td>
</tr>
<tr>
<td>1</td>
<td>ADCB</td>
<td>ADCEBA</td>
<td>34</td>
<td>ADCE</td>
</tr>
<tr>
<td>2</td>
<td>ADC</td>
<td>ADBECA</td>
<td>infeasible</td>
<td>ADB,EC</td>
</tr>
<tr>
<td>3</td>
<td>ABEC, ADB, ADC</td>
<td>ADEBCA</td>
<td>infeasible</td>
<td>ADE, EB</td>
</tr>
<tr>
<td>4</td>
<td>ABEC, ACEB, AD, AEB</td>
<td>ACDBEBA</td>
<td>38</td>
<td>ACBD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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