Vehicle Routing and Scheduling

Martin Savelsbergh
The Logistics Institute
Georgia Institute of Technology
Vehicle Routing and Scheduling

Part II: Algorithmic Enhancements
Handling Practical Complexities
In Construction Heuristics
Insertion Heuristics

Start with a set of unrouted stops

Are there any unrouted stops?

Yes: Select an unrouted stop

Insert selected stop in current set of routes

No: Done
Insertion

Initial

Intermediate

Final
Insertion Heuristics

- Efficient (fast)
- Effective (good quality solutions)
- Easy to implement
- Easy to extend
  - time windows
  - route duration
  - variable delivery quantities
  - ...
Insertion Heuristic

\[ N = \{ \text{unassigned customers} \} \]
\[ R = \{ \text{set of routes} \} \]
\[ \text{WHILE } N \neq \emptyset \text{ DO} \]
\[ p^* = -\infty \]
\[ \text{FOR } j \in N \text{ DO} \]
\[ \text{FOR } r \in R \text{ AND } (i-1,i) \in r \text{ DO} \]
\[ \text{IF Feasible}(i,j) \text{ AND Profit}(i,j) > p^* \text{ THEN} \]
\[ r^* = r, \ i^* = i, \ j^* = j, \ p^* = \text{Profit}(i,j) \]
\[ \text{Insert}(i^*,j^*) \]
\[ \text{Update}(r^*) \]
\[ N = N \setminus \{ j^* \} \]
Insertion Heuristic

- Complexity is $O(n^3)$ if:
  - Feasible() is $O(1)$
  - Profit() is $O(1)$
  - Update() is at most $O(n^2)$

- Vehicle Routing Problem
  - Feasible: $d_j < Q - q_r$
  - Update: $q_r = q_r + d_j$
Time Windows

Arrival time may increase

How to quickly check feasibility?
Latest delivery time

depot

direct travel time

Latest time delivery can take place
Time windows

- **Notation:**
  - $[E_k, L_k]$ time window on start of delivery at $k$

- **Auxiliary information:**
  - $e_k$ earliest time delivery can take place at $k$
  - $l_k$ latest time delivery can take place at $k$
Time windows (cont.)

- Feasible()
  - $d_j < Q - q_r$
  - $e_j = \max\{E_j, e_{i-1} + t_{i-1,j}\}$ & $l_j = \min\{L_j, l_i - t_{j,i}\}$
  - $e_j < l_j$

- Update()
  - for $k=i-1$ to $0$: $l_k = \min\{l_k, l_{k+1} - t_{k,k+1}\}$
  - for $k=i$ to $n+1$: $e_k = \max\{e_k, e_{k-1} + t_{k-1,k}\}$
Route duration

- **Notation:**
  - $T$ planning horizon
  - $L$ route duration limit

- **Auxiliary information:**
  - $e_0$ earliest start time route
    
    $$e_0 = \max\{0, e_{n+1} - L\}$$
  - $l_{n+1}$ latest completion time route
    
    $$l_{n+1} = \min\{l_0 + L, T\}$$
Route duration (cont.)

- Auxiliary information
  - $f_i$ total travel time from $i$ until the end
  - $b_i$ total travel time from the beginning until $i$
Route duration (cont.)

- Feasible()
  - \( e_{n+1} = \max\{e_{n+1}, e_j + t_{j,i} + f_i\} \), \( e_0 = \max\{e_0, e_{n+1} - L\} \)
  - \( l_0 = \min\{l_0, l_j - t_{i-1,j} - b_{i-1}\} \), \( l_{n+1} = \min\{l_{n+1}, l_0 + L\} \)
  - \( d_j < Q - q_r \)
  - \( e_j < l_j \)
  - \( e_0 < l_0 \) & \( l_{n+1} > e_{n+1} \)
Route duration (cont.)

- Update()
  - for k = i-1 to 0:  \( l_k = \min\{l_k, l_{k+1} - t_{k,k+1}\} \)
  - for k = i to n+1:  \( e_k = \max\{e_k, e_{k-1} + t_{k-1,k}\} \)
  - if \( e_0 \) updated
    - for k = 1 to i-1:  \( e_k = \max\{e_k, e_{k-1} + t_{k-1,k}\} \)
  - If \( l_{n+1} \) updated
    - for k = n+1 to i:  \( l_k = \min\{l_k, l_{k+1} - t_{k,k+1}\} \)
Handling Practical Complexities In Improvement Heuristics
2-change

Orientation of traversal is reversed!
Lexicographic search

2-changes
Global variables

- Total travel time $T(u_1,\ldots,u_k)$
  $$\text{sum}(i=1,\ldots,k-1) t(u_i, u_{i+1})$$

- Earliest delivery time $E(u_1,\ldots,u_k)$ at $u_k$
  assuming $u_1$ is left at the opening of its time window
  $$\max(i=1,\ldots,k) \{E_i + T(u_i,\ldots,u_k)\}$$

- Latest delivery time $L(u_1,\ldots,u_k)$ at $u_1$ such that the path remains feasible
  $$\min(i=1,\ldots,k) \{L_i - T(u_1,\ldots,u_i)\}$$
Path concatenation

- \( T(u_1, \ldots, u_k, v_1, \ldots, v_l) = T(u_1, \ldots, u_k) + t(u_k, v_1) + T(v_1, \ldots, v_l) \)
- \( E(u_1, \ldots, u_k, v_1, \ldots, v_l) = \max\{E(u_1, \ldots, u_k) + t(u_k, v_1) + T(v_1, \ldots, v_l), E(v_1, \ldots, v_l)\} \)
- \( L(u_1, \ldots, u_k, v_1, \ldots, v_l) = \min\{L(u_1, \ldots, u_k), L(v_1, \ldots, v_l) - T(v_1, \ldots, v_l) - t(u_k, v_1)\} \)
Path concatenation

Earliest delivery at $v_1$: $E(u_1, \ldots, u_k) + t(u_k, v_1)$

Earliest delivery at $v_l$: $E(u_1, \ldots, u_k) + t(u_k, v_1) + T(v_1, \ldots, v_l)$
Path concatenation

Latest delivery at $u_k$: $L(v_1, \ldots, v_l) - t(u_k, v_1)$

Latest delivery at $u_1$: $L(v_1, \ldots, v_l) - t(u_k, v_1) - T(u_1, \ldots, u_k)$
Observations

- The set of global variables makes it possible to test feasibility of an exchange in constant time.
- The lexicographic search strategy makes it possible to maintain the correct values for the set of global variables in constant time.
Lexicographic search

2-changes
The Dial-a-Ride Problem
Dial-a-Ride Problem

- Dispatch vehicles to pickup person/package at one location (origin) and deliver the person/package at another location (destination)
- Service related constraints
  - pickup window / delivery window
  - maximum wait time
  - maximum ride time
Service related constraints

- Pickup and delivery window
- Maximum wait time
  - Limit waiting time at a stop before departing
- Maximum ride time
  - Limit time between pickup and delivery
Feasibility testing

- Given a sequence of pickups and deliveries does there exist a feasible schedule satisfying pickup and delivery windows, maximum wait time, and maximum ride time constraints?
Example

- Waiting time limit: 10
- Ride time limit: 1.5 * ride time
Example

<table>
<thead>
<tr>
<th>Stop</th>
<th>Early</th>
<th>Late</th>
<th>Arrival</th>
<th>Departure</th>
<th>Waiting</th>
<th>Travel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+</td>
<td>10.15</td>
<td>10.30</td>
<td>10.15</td>
<td>10.15</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>2+</td>
<td>10.45</td>
<td>11.00</td>
<td>10.35</td>
<td>10.45</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>1-</td>
<td>11.00</td>
<td>11.15</td>
<td>11.00</td>
<td>11.00</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2-</td>
<td>11.20</td>
<td>11.40</td>
<td>11.10</td>
<td>11.20</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>3+</td>
<td>11.40</td>
<td>12.00</td>
<td>11.35</td>
<td>11.40</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>3-</td>
<td>12.30</td>
<td>13.00</td>
<td>12.30</td>
<td>12.30</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Stop</th>
<th>Early</th>
<th>Late</th>
<th>Arrival</th>
<th>Departure</th>
<th>Waiting</th>
<th>Travel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+</td>
<td>10.15</td>
<td>10.30</td>
<td>10.15</td>
<td>10.15</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>2+</td>
<td>10.45</td>
<td>11.00</td>
<td>10.35</td>
<td>10.45</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>1-</td>
<td>11.00</td>
<td>11.15</td>
<td>11.00</td>
<td>11.00</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2-</td>
<td>11.20</td>
<td>11.40</td>
<td>11.10</td>
<td>11.20</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>3+</td>
<td>11.40</td>
<td>12.00</td>
<td>11.35</td>
<td>11.40</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>4+</td>
<td>12.10</td>
<td>12.30</td>
<td>11.50</td>
<td>12.10</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>3-</td>
<td>12.30</td>
<td>13.00</td>
<td>12.50</td>
<td>12.50</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>4-</td>
<td>13.00</td>
<td>13.45</td>
<td>13.10</td>
<td>13.10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Stop</th>
<th>Early</th>
<th>Late</th>
<th>Arrival</th>
<th>Departure</th>
<th>Waiting</th>
<th>Travel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+</td>
<td>10.15</td>
<td>10.30</td>
<td>10.15</td>
<td>10.15</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>2+</td>
<td>10.45</td>
<td>11.00</td>
<td>10.35</td>
<td>10.45</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>1-</td>
<td>11.00</td>
<td>11.15</td>
<td>11.00</td>
<td>11.00</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2-</td>
<td>11.20</td>
<td>11.40</td>
<td>11.10</td>
<td>11.20</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>3+</td>
<td>11.40</td>
<td>12.00</td>
<td>11.35</td>
<td>11.40</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>4+</td>
<td>12.10</td>
<td>12.30</td>
<td>11.50</td>
<td>12.10</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>3-</td>
<td>12.30</td>
<td>13.00</td>
<td>12.50</td>
<td>12.50</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>4-</td>
<td>13.00</td>
<td>13.45</td>
<td>13.10</td>
<td>13.10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stop</th>
<th>Early</th>
<th>Late</th>
<th>Arrival</th>
<th>Departure</th>
<th>Waiting</th>
<th>Travel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+</td>
<td>10.15</td>
<td>10.30</td>
<td>10.20</td>
<td>10.20</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>2+</td>
<td>10.45</td>
<td>11.00</td>
<td>10.40</td>
<td>10.50</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>1-</td>
<td>11.00</td>
<td>11.15</td>
<td>11.05</td>
<td>11.05</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2-</td>
<td>11.20</td>
<td>11.40</td>
<td>11.15</td>
<td>11.25</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>3+</td>
<td>11.40</td>
<td>12.00</td>
<td>11.40</td>
<td>11.50</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4+</td>
<td>12.10</td>
<td>12.30</td>
<td>12.00</td>
<td>12.10</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>3-</td>
<td>12.30</td>
<td>13.00</td>
<td>12.50</td>
<td>12.50</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>4-</td>
<td>13.00</td>
<td>13.45</td>
<td>13.10</td>
<td>13.10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Feasibility testing

- Can be done in linear time ...

- Invariant property:
  - No feasible schedule can have an arrival or departure time earlier than the computed arrival and departure times
Feasibility checking

- Notation:
  - \([e_j, l_j]\) time window
  - \(\omega\) waiting time limit
  - \(\alpha t_{i+,i-}\) ride time limit
  - \(A_j\) arrival time
  - \(D_j\) departure time
  - \(L_j\) latest feasible departure time
Pass 1 - Forward

- Account for pickup and delivery windows and maximum waiting time constraints

- Normal updates
  - $A_j = D_{j-1} + t_{j-1,j}$
  - $D_j = \max\{e_j, A_j\}$
  - $L_j = \min\{l_j, L_{j-1} + t_{j-1,j} + \omega\}$
Pass 1 (cont.)

- Infeasibility
  - $A_j > l_j$
  - $L_j < e_j$
- Special update when $A_j + \omega < e_j$
  - $A_j = e_j - \omega$
  - $D_j = e_j$
Pass 2 - Backward

- Update arrival and departure times and “check” ride time constraints
  - Waiting time from j until the end of route W
- Normal updates
  - \( D_j = A_{j+1} - t_{j-1,j} \)
  - \( A_j = \max\{A_j, D_j - \omega\} \)
  - \( W = W + (D_j - A_j) \)
Pass 2 (cont.)

- **Infeasibility (pickup point \( j = i+ \))**
  - \( \Delta = (D_{i-} - D_{i+}) - \alpha t_{i+,i-} \)
  - \( D_{i+} + \Delta > L_{i+} \)
  - \( \Delta > W \)

- **Special update when \( \Delta > W \)**
  - \( D_j = D_j + \Delta \)
  - \( A_j = \max\{A_j, D_j - \omega\} \)
  - \( W = W - \Delta \)
Pass 3 - Forward

- Finalize arrival and departure times and check ride time constraints
- Normal updates
  - $A_j = D_{j-1} + t_{j-1,j}$
  - $D_j = \max\{A_j, D_j\}$
Pass 3 (cont.)

- Infeasibility (drop-off point $j = i-$)
  - $\Delta = (D_{i-} - D_{i+}) - \alpha t_{i+,i-}$
  - $D_{i-} > l_{i-}$
  - $\Delta > 0$
Discussion

- People transportation
  - delivery window \([0, l_j]\)
  - no pickup window
  - waiting time at pickup only
  - different ride time limits
  - consecutive stops at same location (waiting time per location rather than stop)

- Package transportation
  - no waiting time limits
Greedy Randomized Adaptive Search Procedure
Construction + Improvement

- Greedily create feasible set of routes
  - Improve feasible Set of routes
Simple enhancement

- Multi-start neighborhood search:
  - independent neighborhood searches
  - random starting solutions
Multi-start neighborhood search

Initialize: best = ∞

Ready to stop?

Yes

Done

No

Randomly create feasible set of routes

Improve feasible set of routes

Better than best?

Yes

Update best

No
Simple enhancement

- Iterated neighborhood search:
  - independent neighborhood searches
  - starting solutions obtained from a previous local optimum by a suitable perturbation method
Iterated neighborhood search

1. Initialize: best = ∞
2. Greedily create feasible set of routes
3. Ready to stop?
   - Yes: Done
   - No: Perturb feasible set of routes
4. Improve feasible set of routes
5. Better than best?
   - Yes: Update best
   - No: Go back to Ready to stop?
**Greedy algorithm**

- Constructs a solution one element at a time:
  - Define candidate elements
  - Apply greedy function to each candidate element
  - Rank candidate elements according to greedy function value
  - Add best ranked element to solution
Semi-greedy algorithm

- Constructs a solution one element at a time:
  - Define candidate elements
  - Apply greedy function to each candidate element
  - Rank candidate elements according to greedy function value
  - Place well-ranked elements in a restricted candidate list (RCL)
  - Select an element from the RCL at random and add it to the solution
Restricted Candidate List

- Cardinality based:
  - Place $k$ best candidates in RCL

- Value based I:
  - Place all candidates having greedy value better than $\alpha \times \text{max value in RCL}$ (with $0 \leq \alpha \leq 1$)

- Value based II:
  - Place all candidates having greedy value better than $\text{min value} + \alpha \times (\text{max value} - \text{min value})$ in RCL (with $0 \leq \alpha \leq 1$)
Semi-greedy

Initialize: best = ∞

Ready to stop?
Yes
Done

No

Semi-greedily create feasible set of routes

Better than best?
Yes
Update best

No

Greedy Randomized Adaptive Search Procedure

Initialize: best = ∞

Ready to stop?

Yes

Done

No

Semi-greedily create feasible set of routes

Improve feasible set of routes

Better than best?

Yes

Update best

No
GRASP

- GRASP tries to capture good features of greedy & random constructions
- Iteratively
  - samples solution space using a greedy probabilistic bias to construct a feasible solution
  - applies local search to attempt to improve upon the constructed solution
Advanced Neighborhood Search
Neighborhood search - observations

- **Weakness:**
  - looks only one step ahead, and may get trapped in a bad local optimum

- **Strength:**
  - Fast and easy to implement
Neighborhood search - sophisticated enhancements

Goal:
- Increase quality of solution
- Do not increase time to find solution too much

- Tabu Search
- Large Scale Neighborhood Search
Tabu Search
Tabu Search

Local minimum

Global minimum
Tabu Search

- Strategy to escape from a local optimum and continue the search

- Implementation
  - Best move is always performed
  - Avoid cycling using short-term memory
    - Attributes of recent solutions stored in tabu list
    - Moves involving attributes in tabu list are discarded (tabu)
Short-term Memory

- Tabu list
  - Tabu list size - maximum number of attributes stored in the list (FIFO)
  - Tabu list tenure - maximum number of iterations attribute remains in the list
Short-term Memory

- Recency-based
  - Last $t$ moves

- Frequency-based
  - Number of times a specific move is performed
  - Penalize moves with higher frequency
Intensification and Diversification

- Intensification
  - Intensify the search in promising regions
- Diversification
  - Diversify the search across contrasting regions
- Examples
  - Varying the tabu list size
  - Adjusting the cost structure
Observations

- Tabu search can be highly effective
- Tabu search can be prohibitively time consuming

Remedy: speed up neighborhood search
Granular Tabu Search

- Reduce the number of moves evaluated at each iteration
- Routing and scheduling problems:
  - Long connections are unlikely to be part of an optimal solution
Granular neighborhoods

- Restriction of ordinary neighborhoods
  - Consider only connections whose cost is below a threshold
  - Consider only moves involving promising connections
  - Threshold: $v \times (UB / n)$
    - $v$ sparsification parameter
    - $UB/n$ average cost of connection in solution
Granular neighborhoods

- Intensification/diversification tool
  - small $v \rightarrow$ intensification
  - large $v \rightarrow$ diversification
Vehicle routing problem

- Set of connections:
  - connections of the current and best solution
  - connections involving the depot
  - connections with costs less than threshold
- Connections used as move generators
Vehicle routing problem

- Savings heuristic
- 1-relocate, 2-relocate, swap, 2-change
- Tabu tenure: random in $[5,10]$  
- Granularity based intensification/diversification
  - intensification: $v$ in $[1,2]$
  - diversification: no improvement in $15*n$ iterations, then $v = 5$ for $n$ iterations
Large-Scale Neighborhood Search
Compounded 1-relocate

- Given the TSP tour $T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$

- The new TSP tour $T' = (1, 2, 4, 5, 6, 3, 7, 9, 10, 8, 11)$
Compounded 1-relocate

- The size of the 1-relocate neighborhood is $O(n^2)$
- The size of the compounded independent 1-relocate neighborhood is $\Theta(1.7549^n)$ (Proof is by solving a recursion for the number of paths from 1 to n+1)
Compounded 1-relocate

Improvement Graph

T = (1,2,3,4,5,6,7,8,9,10,1):

Construct improvement graph

\[ c_{1,2} = 0 \]
\[ c_{2,7} = -(d_{2,3} + d_{3,4} + d_{6,7}) + (d_{2,4} + d_{6,3} + d_{3,7}) \]
\[ c_{7,11} = -(d_{7,8} + d_{8,9} + d_{10,1}) + (d_{7,9} + d_{10,8} + d_{8,1}) \]
Improvement Graph

- Only forward arcs are allowed
- Node 1 is always kept fixed

- Find shortest path from 1 to n+1 in $O(n^2)$ time
- Negative cost shortest path implies an improving move
Compounded swap

- \( T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1) \)

Compounded swap neighborhood

- \( T' = (1, 2, 3, 5, 4, 6, 10, 8, 9, 7, 11, 1) \)
Compounded 2-change

- $T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1)$

Compounded 2-change neighborhood

- $T = (1, 2, 3, 5, 4, 6, 10, 9, 8, 7, 11, 1)$
Moreover ...

- $T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1)$

Let
- $\rightarrow$ represent 2-change move
- $\Rightarrow$ represent swap move
- $\Rightarrow$ represent 1-relocate move

- $T = (1, 2, 4, 3, 5, 7, 8, 6, 9, 12, 11, 10, 1)$
Vehicle routing problem

Instance

Solution
Two representations

vehicle 1

1_1  2_1  n_1

d

vehicle 2

1_2  2_2  n_2

vehicle m

1_m  2_m  n_m

Multi tour representation
Two representations

Single tour representation
Single tour representation

- Improvement graph is analogous to the TSP improvement graph
- For every ordering of vehicle one obtains a different neighborhood
Multi tour representation

- The cost structure is not well defined for arc \((4_1, 2_3)\)
- Establish an alignment scheme to define and allow only forward arcs
Alignment schemes

Left adjusted

vehicle 1  1₁  2₁  3₁  4₁  5₁  6₁  7₁  8₁
vehicle 2   d   1₂  2₂  3₂
vehicle 3  1₃  2₃  3₃  4₃  5₃  6₃

Right adjusted

vehicle 1  1₁  2₁  3₁  4₁  5₁  6₁  7₁  8₁
vehicle 2   d   1₂  2₂  3₂  3₂
vehicle 3  1₃  2₃  3₃  4₃  5₃  6₃

Arbitrarily adjusted
After applying the exchanges implied by the shortest path:
MT neighborhood

- Additional flexibility:
  - shortest path from left to right
  - shortest path from right to left

- Additional complexity:
  - moves no longer independent due to capacity and distance restriction

- Constructing improvement graph and finding shortest path take $O(n^2)$ time
Handling capacity constraints

- for each node keep a working capacity label of each vehicle as well as a distance label
- available capacity[k] = available capacity of vehicle k in current solution
- working capacity[i,k] = available capacity[k] + effects of changes corresponding to shortest path to i
- allow only feasible arcs with respect to working capacity in shortest path
Searching ST and MT neighborhoods

- Complexity of the search is $O(n^2 + nm)$
  - $O(n^2)$ for creating the improvement graph and running the shortest path algorithm
  - $O(nm)$ for updating the labels at each node once after all the incoming arcs to the node is considered