Practical recommendations for an efficient local search implementation: a CVRP case study

Daniel Palhazi Cuervo, Kenneth Sörensen and Peter Goos
EU/MEeting 2012 - Copenhagen, May 11th 2012
Outline

- Design phase
  - Dealing with constraints
  - Consideration of infeasible solutions
- Implementation phase
  - Neighbourhood calculation and update
  - Data structure (solution representation)
- Final remarks
Dealing with constraints

Alternatives:

▶ Discard infeasible solutions
  ▶ Continual checking of constraint violations
▶ Penalize the cost function $\mathcal{C}(s) = d(s) + \alpha l(s)$
  ▶ Need of a measure to quantify infeasibility

Only feasible
▶ Fixed $\alpha$ value (usually very large)

Feasible and infeasible
▶ Oscillation strategy (variation of $\alpha$)
▶ Important: guarantee the transition between feasible and infeasible solution space
Consider infeasible solutions - Motivation
Consider infeasible solutions - Motivation
Consider infeasible solutions - Motivation
Consider infeasible solutions - Worst case
Consider infeasible solutions - Important case
Consider infeasible solutions - Important case
Consider infeasible solutions - Important case
Consider infeasible solutions - Important case
Computational experiments

- Set of instances proposed by Augerat et al.
- Oscillation strategy:
  - Initial value of $\alpha = 1$
  - Increase $\alpha$ by 1 every time no better solution can be found
  - In case of getting stuck in the infeasible solution space, set value of $\alpha = 1$

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Set A</th>
<th>Set B</th>
<th>Set P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only feasible sol.</td>
<td>10.13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Force feasibility</td>
<td>10.18</td>
<td>10.80</td>
<td>-</td>
</tr>
<tr>
<td>Oscillation</td>
<td>6.88</td>
<td>3.82</td>
<td>5.96</td>
</tr>
</tbody>
</table>

Table: Average solution gap
Implementation - Goal

Be able to calculate/perform in the least order of time complexity:

- Differential cost of neighbouring solutions
- Update the neighbourhood set
- Apply the neighbourhood operators (neighbouring solution)
Differential cost of neighbouring solutions

- Relocate (intra-route and inter-route)
Differential cost of neighbouring solutions

- Exchange (intra-route and inter-route)
Differential cost of neighbouring solutions

- Two-opt (intra-route)
Differential cost of neighbouring solutions

- Crossover (inter-route)
Update the set of neighbouring solutions

- Only recalculate the differential cost of the neighbouring solutions that involve the modified routes
  - Time complexity reduced from $k^2$ to $2(k - 2) + 2k$
- Reduction of average execution time by 50%
Data Structure
Data Structure
Final remarks

About considering infeasible solutions:
- Can be helpful to find valuable information about the global optimum solution in fewer iterations
- Useful when is difficult to generate a feasible initial solution

Last recommendation:
- Evaluate the trade-off between efficiency and usability of the code