

## Outline

# DMP204 SCHEDULING, TIMETABLING AND ROUTING

## Lecture 27 Vehicle Routing Heuristics

Marco Chiarandini

1. Construction Heuristics
  - Construction Heuristics for CVRP
  - Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

2

## Outline

Construction Heuristics  
Improvement Heuristics  
Metaheuristics  
CP for VRP

Constr. Heur. for CVRP  
Constr. Heur. for VRPTW

1. Construction Heuristics
  - Construction Heuristics for CVRP
  - Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

3

## Construction Heuristics for CVRP

Construction Heuristics  
Improvement Heuristics  
Metaheuristics  
CP for VRP

Constr. Heur. for CVRP  
Constr. Heur. for VRPTW

- TSP based heuristics
- Savings heuristics (Clarke and Wright)
- Insertion heuristics
- **Cluster-first route-second**
  - Sweep algorithm
  - Generalized assignment
  - Location based heuristic
  - Petal algorithm
- Route-first cluster-second

Cluster-first route-second seems to perform better  
(Note: Distinction Construction Heuristic / Iterative Improvement  
is often blurred)

5

## Construction heuristics for TSP

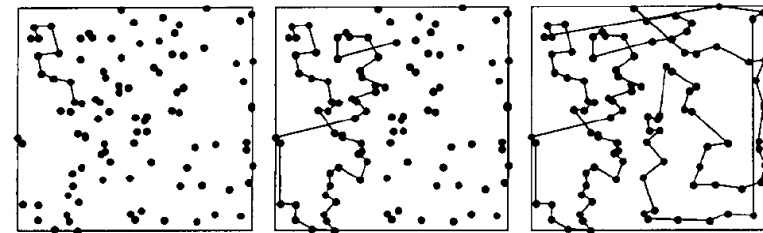
They can be used for route-first cluster-second or for growing multiple tours subject to capacity constraint.

- Heuristics that Grow Fragments
  - Nearest neighborhood heuristics
  - Double-Ended Nearest Neighbor heuristic
  - Multiple Fragment heuristic (aka, greedy heuristic)
- Heuristics that Grow Tours
 

<ul style="list-style-type: none"> <li>• Nearest Addition</li> <li>• Farthest Addition</li> <li>• Random Addition</li> <li>• Clarke-Wright savings heuristic</li> </ul>	<ul style="list-style-type: none"> <li>• Nearest Insertion</li> <li>• Farthest Insertion</li> <li>• Random Insertion</li> </ul>
---	---
- Heuristics based on Trees
  - Minimum spanning tree heuristic
  - Christofides' heuristics

(But recall! Concorde: <http://www.tsp.gatech.edu/>)

[Bentley, 1992]



**Figure 1.** The Nearest Neighbor heuristic.

NN (Flood, 1956)

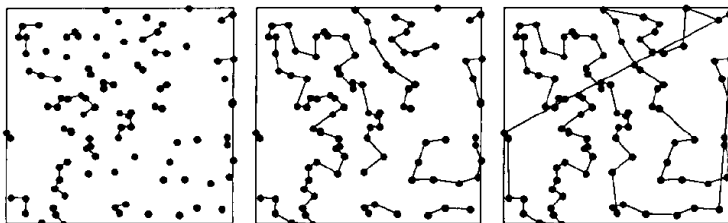
1. Randomly select a starting node
2. Add to the last node the closest node until no more node is available
3. Connect the last node with the first node

Running time  $O(N^2)$

6

7

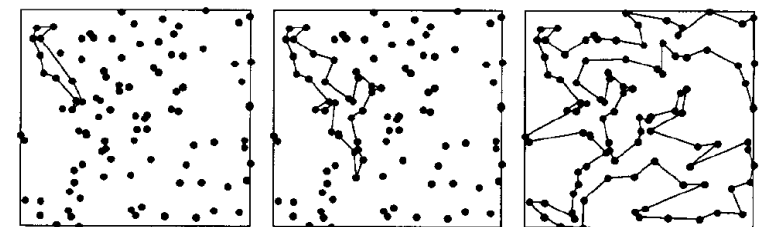
[Bentley, 1992]



**Figure 5.** The Multiple Fragment heuristic.

Add the cheapest edge provided it does not create a cycle.

[Bentley, 1992]



**Figure 8.** The Nearest Addition heuristic.

NA

1. Select a node and its closest node and build a tour of two nodes
2. Insert in the tour the closest node  $v$  until no more node is available

Running time  $O(N^3)$

8

9

[Bentley, 1992]

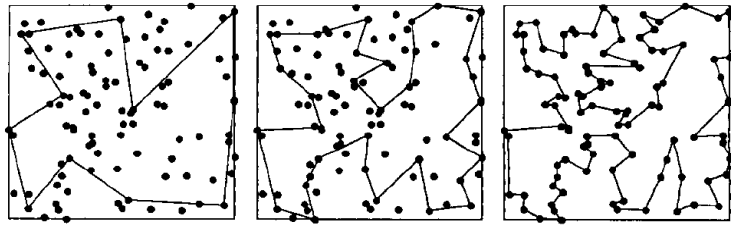


Figure 11. The Farthest Addition heuristic.

FA

1. Select a node and its farthest and build a tour of two nodes
2. Insert in the tour the farthest node  $v$  until no more node is available

FA is more effective than NA because the first few farthest points sketch a broad outline of the tour that is refined after.

Running time  $O(N^3)$

10

[Bentley, 1992]

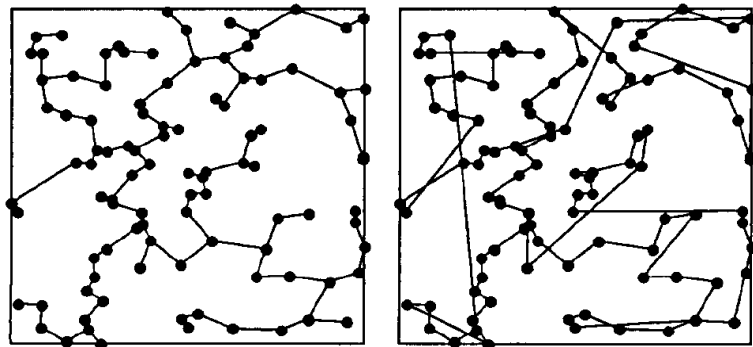


Figure 18. The Minimum Spanning Tree heuristic.

1. Find a minimum spanning tree  $O(N^2)$
2. Append the nodes in the tour in a depth-first, inorder traversal

Running time  $O(N^2)$

$$A = \text{MST}(I)/\text{OPT}(I) \leq 2$$

12

[Bentley, 1992]

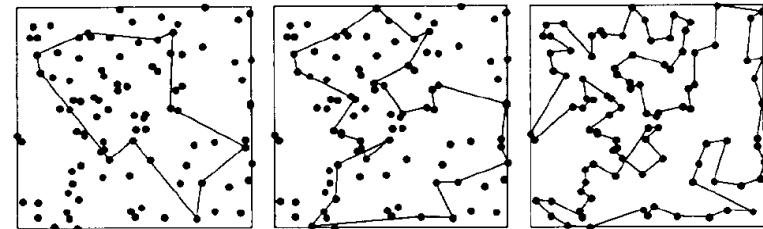


Figure 14. The Random Addition heuristic.

[Bentley, 1992]

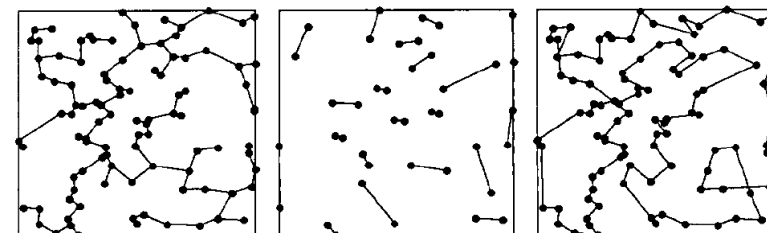


Figure 19. Christofides' heuristic.

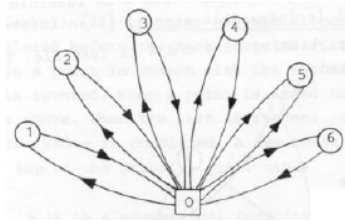
1. Find the minimum spanning tree  $T$ .  $O(N^2)$
2. Find nodes in  $T$  with odd degree and find the cheapest perfect matching  $M$  in the complete graph consisting of these nodes only. Let  $G$  be the multigraph all nodes and edges in  $T$  and  $M$ .  $O(N^3)$
3. Find an Eulerian walk (each node appears at least once and each edge exactly once) on  $G$  and an embedded tour.  $O(N)$

Running time  $O(N^3)$

$$A = \text{CH}(I)/\text{OPT}(I) \leq 3/2$$

13

### Construction Heuristics Specific for VRP



#### Clarke-Wright Saving Heuristic (1964)

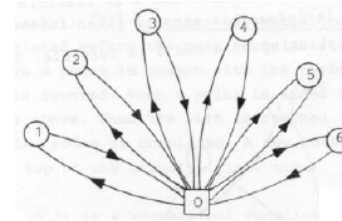
1. Start with an initial allocation of one vehicle to each customer (0 is the depot for VRP or any chosen city for TSP)

Sequential:

2. consider in turn route  $(0, i, \dots, j, 0)$  determine  $s_{ki}$  and  $s_{jl}$
3. merge with  $(k, 0)$  or  $(0, l)$

14

### Construction Heuristics Specific for VRP



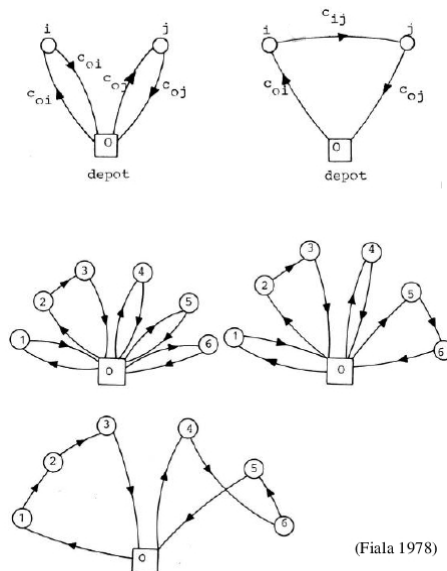
#### Clarke-Wright Saving Heuristic (1964)

1. Start with an initial allocation of one vehicle to each customer (0 is the depot for VRP or any chosen city for TSP)

Parallel:

2. Calculate saving  $s_{ij} = c_{0i} + c_{0j} - c_{ij}$  and order the saving in non-increasing order
3. scan  $s_{ij}$   
 merge routes if i)  $i$  and  $j$  are not in the same tour ii) neither  $i$  and  $j$  are interior to an existing route iii) vehicle and time capacity are not exceeded

14



(Fiala 1978)

#### Matching Based Saving Heuristic

1. Start with an initial allocation of one vehicle to each customer (0 is the depot for VRP or any chosen city for TSP)
2. Compute  $s_{pq} = t(S_p) + t(S_q) - t(S_p \cup S_q)$  where  $t(\cdot)$  is the TSP solution
3. Solve a max weighted matching on the  $S_k$  with weights  $s_{pq}$  on edges. A connection between a route  $p$  and  $q$  exists only if the merging is feasible.

15

16

## Insertion Heuristic

$$\alpha(i, k, j) = c_{ik} + c_{kj} - \lambda c_{ij}$$

$$\beta(i, k, j) = \mu c_{0k} - \alpha(i, k, j)$$

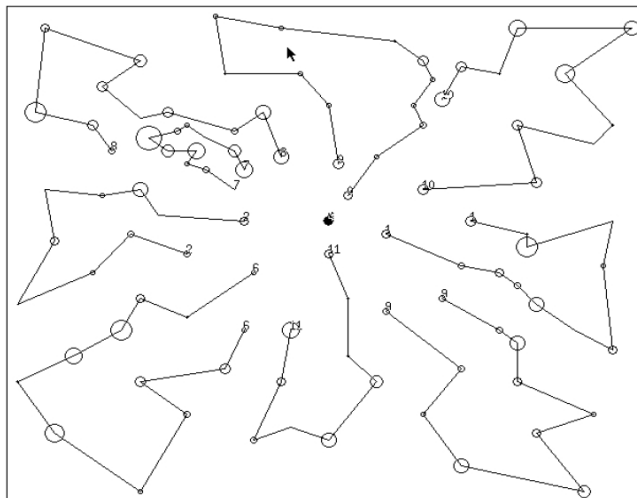
1. construct emerging route  $(0, k, 0)$
2. compute for all  $k$  unrouted the feasible insertion cost:

$$\alpha^*(i_k, k, j_k) = \min_p \{\alpha(i_p, k, i_{p+1})\}$$

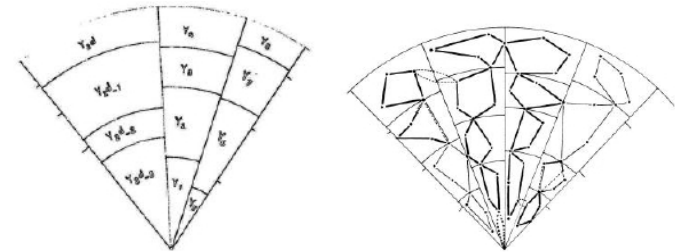
if no feasible insertion go to 1 otherwise choose  $k^*$  such that

$$\beta^*(i_k^*, k^*, j_k^*) = \max_k \{\beta(i_k, k, j_k)\}$$

17



19



Cluster-first route-second: Sweep algorithm [Wren & Holliday (1971)]

1. Choose  $i^*$  and set  $\theta(i^*) = 0$  for the rotating ray
2. Compute and rank the polar coordinates  $(\theta, \rho)$  of each point
3. Assign customers to vehicles until capacity not exceeded. If needed start a new route. Repeat until all customers scheduled.

18

Cluster-first route-second: Generalized-assignment-based algorithm  
[Fisher & Jaikumar (1981)]

1. Choose a  $j_k$  at random for each route  $k$
2. For each point compute

$$d_{ik} = \min\{c_{0,i} + c_{i,j_k} + c_{j_k,0}, c_{0,j_k} + c_{j_k,i} + c_{i,0}\} - (c_{0,j_k} + c_{j_k,0})$$

3. Solve GAP with  $d_{ik}$ ,  $Q$  and  $q_i$

20

Cluster-first route-second: Location based heuristic [Bramel & Simchi-Levi (1995)]

1. Determine seeds by solving a capacited location problem (k-median)
2. Assign customers to closest seed

Cluster-first route-second: Petal Algorithm

1. Construct a subset of feasible routes
2. Solve a set partitioning problem

(better performance than insertion and saving heuristics)

21

## Exercise

Route-first cluster-second [Beasley, 1983]

1. Construct a TSP tour over all customers
2. Choose an arbitrary orientation of the TSP;  
partition the tour according to capacity constraint;  
repeat for several orientations and select the best  
Alternatively, solve a shortest path in an acyclic digraph with costs on arcs:  $d_{ij} = c_{0i} + c_{0j} + l_{ij}$  where  $l_{ij}$  is the cost of traveling from  $i$  to  $j$  in the TSP tour.

(not very competitive)

Which heuristics can be used to minimize  $K$  and which ones need to have  $K$  fixed a priori?

23

24

Extensions of those for CVRP [Solomon (1987)]

- Savings heuristics (Clarke and Wright)
- Time-oriented nearest neighbors
- Insertion heuristics
- Time-oriented sweep heuristic

## Time-Oriented Nearest-Neighbor

- Add the unrouted node “closest” to the depot or the last node added without violating feasibility
- Metric for “closest”:

$$c_{ij} = \delta_1 d_{ij} + \delta_2 T_{ij} + \delta_3 v_{ij}$$

$d_{ij}$  geographical distance  
 $T_{ij}$  time distance  
 $v_{ij}$  urgency to serve  $j$

26

27

## Insertion Heuristics

**Step 1:** Compute for each unrouted customer  $u$  the *best feasible position* in the route:

$$c_1(i(u), u, j(u)) = \min_{p=1, \dots, m} \{c_1(i_{p-1}, u, i_p)\}$$

( $c_1$  is a composition of increased time and increase route length due to the insertion of  $u$ )  
(use push forward rule to check feasibility efficiently)

**Step 2:** Compute for each unrouted customer  $u$  which can be feasibly inserted:

$$c_2(i(u^*), u^*, j(u^*)) = \max_u \{\lambda d_{0u} - c_1(i(u), u, j(u))\}$$

(max the benefit of servicing a node on a partial route rather than on a direct route)

**Step 3:** Insert the customer  $u^*$  from Step 2

- Let's assume waiting is allowed and  $s_i$  indicates service times

- $b_i = \max\{e_i, b_j + s_j + t_{ji}\}$  begin of service

- insertion of  $u$ :  $(i_0, i_1, \dots, i_p, u, i_{p+1}, \dots, i_m)$

- $PF_{i_{p+1}} = b_{i_{p+1}}^{new} - b_{i_{p+1}} \geq 0$  push forward

- $PF_{i_{r+1}} = \max\{0, PF_{i_r} - w_{i_{r+1}}\}, \quad p \leq r \leq m - 1$

## Theorem

The insertion is feasible if and only if:

$$b_u \leq l_u \quad \text{and} \quad PF_{i_r} + b_{i_r} \leq l_{i_r} \quad \forall p < r \leq m$$

Check vertices  $k, u \leq k \leq m$  sequentially.

- if  $b_k + PF_k > l_k$  then stop: the insertion is infeasible
- if  $PF_k = 0$  then stop: the insertion is feasible

28

29

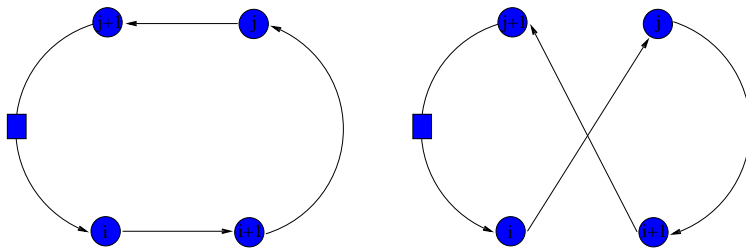
1. Construction Heuristics
  - Construction Heuristics for CVRP
  - Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

- Neighborhoods structures:
  - Intra-route: 2-opt, 3-opt, Lin-Kernighan (not very well suited) 2H-opt, Or-opt
  - Inter-routes:  $\lambda$ -interchange, relocate, exchange, cross, 2-opt\*, ejection chains, GENI
- Solution representation and data structures
  - They depend on the neighborhood.
  - It can be advantageous to change them from one stage to another of the heuristic

## Intra-route Neighborhoods

2-opt

$$\{i, i + 1\}\{j, j + 1\} \longrightarrow \{i, j\}\{i + 1, j + 1\}$$

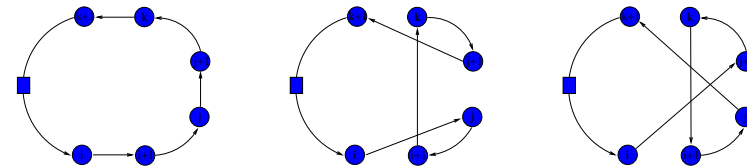


$O(n^2)$  possible exchanges  
 One path is reversed

## Intra-route Neighborhoods

3-opt

$$\{i, i + 1\}\{j, j + 1\}\{k, k + 1\} \longrightarrow \dots$$



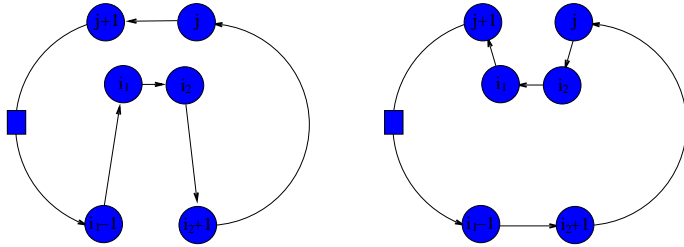
$O(n^3)$  possible exchanges  
 Paths can be reversed



# Intra-route Neighborhoods

Or-opt [Or (1976)]

$$\{i_1 - 1, i_1\}\{i_2, i_2 + 1\}\{j, j + 1\} \rightarrow \{i_1 - 1, i_2 + 1\}\{j, i_1\}\{i_2, j + 1\}$$



sequences of one, two, three consecutive vertices relocated  
 $O(n^2)$  possible exchanges — No paths reversed

# Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]



Figure 6. The exchange neighborhood.

# Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

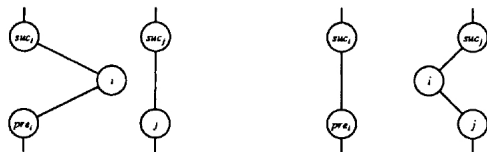


Figure 5. The relocate neighborhood.

# Inter-route Neighborhoods

[Savelsbergh, ORSA (1992)]

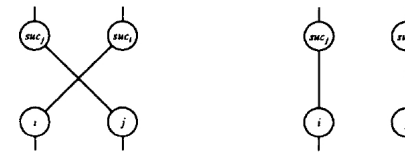


Figure 7. The cross neighborhood.

GENI: generalized insertion [Gendreau, Hertz, Laporte, Oper. Res. (1992)]

- select the insertion restricted to the neighborhood of the vertex to be added (not necessarily between consecutive vertices)
- perform the best 3- or 4-opt restricted to reconnecting arc links that are close to one another.

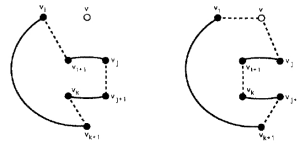


Figure 1. Type I insertion of vertex  $v$  between  $v_i$  and  $v_j$ .

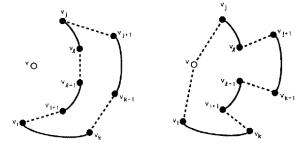
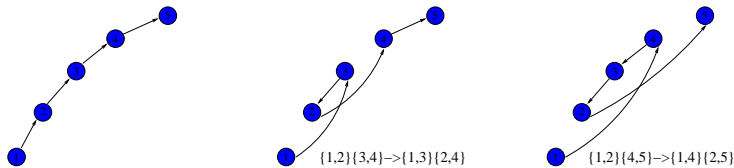


Figure 2. Type II insertion of vertex  $v$  between  $v_i$  and  $v_j$ .

### Search Strategy

- Lexicographic search, for 2-exchange:
  - $i = 1, 2, \dots, n - 2$  (outer loop)
  - $j = i + 2, i + 3, \dots, n$  (inner loop)



Previous path is expanded by the edge  $\{j - 1, j\}$

## Efficient Implementation

### Intra-route

Time windows: Feasibility check

In TSP verifying  $k$ -optimality requires  $O(n^k)$  time  
 In TSPTW feasibility has to be tested then  $O(n^{k+1})$  time

(Savelsbergh 1985) shows how to verify constraints in constant time

Search strategy + Global variables



$O(n^k)$  for  $k$ -optimality in TSPTW

Global variables (auxiliary data structure)

- Maintain auxiliary data such that it is possible to:
  - handle single move in constant time
  - update their values in constant time

Ex.: in case of time windows:

- total travel time of a path
- earliest departure time of a path
- latest arrival time of a path

1. Construction Heuristics
  - Construction Heuristics for CVRP
  - Construction Heuristics for VRPTW
2. Improvement Heuristics
3. Metaheuristics
4. Constraint Programming for VRP

Many and fancy examples, but **first** thing to try:

- Variable Neighborhood Search + Iterated greedy

42

## Basic Variable Neighborhood Descent (BVND)

### Procedure VND

**input** :  $\mathcal{N}_k, k = 1, 2, \dots, k_{\max}$ , and an initial solution  $s$

**output**: a local optimum  $s$  for  $\mathcal{N}_k, k = 1, 2, \dots, k_{\max}$

$k \leftarrow 1$

**repeat**

$s' \leftarrow \text{FindBestNeighbor}(s, \mathcal{N}_k)$

**if**  $g(s') < g(s)$  **then**

$s \leftarrow s'$

$(k \leftarrow 1)$

**else**

$k \leftarrow k + 1$

**until**  $k = k_{\max}$  ;

44

43

## Variable Neighborhood Descent (VND)

### Procedure VND

**input** :  $\mathcal{N}_k, k = 1, 2, \dots, k_{\max}$ , and an initial solution  $s$

**output**: a local optimum  $s$  for  $\mathcal{N}_k, k = 1, 2, \dots, k_{\max}$

$k \leftarrow 1$

**repeat**

$s' \leftarrow \text{IterativeImprovement}(s, \mathcal{N}_k)$

**if**  $g(s') < g(s)$  **then**

$s \leftarrow s'$

$(k \leftarrow 1)$

**else**

$k \leftarrow k + 1$

**until**  $k = k_{\max}$  ;

45

- Final solution is locally optimal w.r.t. all neighborhoods
- First improvement may be applied instead of best improvement
- Typically, order neighborhoods from smallest to largest
- If iterative improvement algorithms  $\Pi_k$ ,  $k = 1, \dots, k_{\max}$  are available as black-box procedures:
  - order black-boxes
  - apply them in the given order
  - possibly iterate starting from the first one
  - order chosen by: *solution quality* and *speed*

General recommendation: use a combination of 2-opt\* + or-opt [Potvin, Rousseau, (1995)]

However,

- Designing a local search algorithm is an **engineering** process in which learnings from other courses in CS might become important.
- It is important to make such algorithms as much efficient as possible.
- Many choices are to be taken (search strategy, order, auxiliary data structures, etc.) and they may interact with instance features. Often a trade-off between examination cost and solution quality must be decided.
- The assessment is conducted through:
  - analytical analysis (computational complexity)
  - **experimental analysis**

46

**Table 5.6.** The effect of 3-opt on the Clarke and Wright algorithm.

Problem	Sequential				Parallel			
	No 3-opt <sup>1</sup>	+ 3-opt FI <sup>2</sup>	+ 3-opt BI <sup>3</sup>	K <sup>4</sup>	No 3-opt <sup>5</sup>	+ 3-opt FI <sup>6</sup>	+ 3-opt BI <sup>7</sup>	K <sup>8</sup>
E051-05e	625.56	624.20	624.20	5	584.64	578.56	578.56	6
E076-10e	1005.25	991.94	991.94	10	900.26	888.04	888.04	10
E101-08e	982.48	980.93	980.93	8	886.83	878.70	878.70	8
E101-10c	939.99	930.78	928.64	10	833.51	824.42	824.42	10
E121-07c	1291.33	1232.90	1237.26	7	1071.07	1049.43	1048.53	7
E151-12c	1299.39	1270.34	1270.34	12	1133.43	1128.24	1128.24	12
E200-17c	1708.00	1667.65	1669.74	16	1395.74	1386.84	1386.84	17
D051-06c	670.01	663.59	663.59	6	618.40	616.66	616.66	6
D076-11c	989.42	988.74	988.74	12	975.46	974.79	974.79	12
D101-09c	1054.70	1046.69	1046.69	10	973.94	968.73	968.73	9
D101-11c	952.53	943.79	943.79	11	875.75	868.50	868.50	11
D121-11c	1646.60	1638.39	1637.07	11	1596.72	1587.93	1587.93	11
D151-14c	1383.87	1374.15	1374.15	15	1287.64	1284.63	1284.63	15
D200-18c	1671.29	1652.58	1652.58	20	1538.66	1523.24	1521.94	19

<sup>1</sup>Sequential savings.

<sup>2</sup>Sequential savings + 3-opt and first improvement.

<sup>3</sup>Sequential savings + 3-opt and best improvement.

<sup>4</sup>Sequential savings: number of vehicles in solution.

<sup>5</sup>Parallel savings.

<sup>6</sup>Parallel savings + 3-opt and first improvement.

<sup>7</sup>Parallel savings + 3-opt and best improvement.

<sup>8</sup>Parallel savings: number of vehicles in solution.

What is best?

## Iterated Greedy

**Key idea:** use the VRP construction heuristics

- alternation of Construction and Deconstruction phases
- an acceptance criterion decides whether the search continues from the new or from the old solution.

### Iterated Greedy (IG):

determine initial candidate solution  $s$

**while** termination criterion is not satisfied **do**

```

  r := s
  greedily deconstruct part of s
  greedily reconstruct the missing part of s
  apply subsidiary iterative improvement procedure (eg, VNS)
  based on acceptance criterion,
  keep s or revert to s := r

```

In the literature, the overall heuristic idea received different names:

- Removal and reinsertion
- Ruin and repair
- Iterated greedy
- Fix and re-optimize

50

Dispersion sub-problem:

choose  $q$  customers to remove with minimal  $r_{ij}$

Heuristic stochastic procedure:

- choose a pair randomly;
- select an already removed  $i$  and find  $j$  that minimizes  $r_{ij}$

52

## Remove

Remove some related customers  
(their re-insertion is likely to change something)

Relatedness measure  $r_{ij}$

- geographical

$$r_{ij} = \frac{1}{D} (d'(i, j) + d'(i, j + n) + d'(i + n, j) + d'(i + n, j + n))$$

- temporal and load based

$$d'(u, v) = |T_{p_i} - T_{p_j}| + |T_{d_i} - T_{d_j}| + |l_i - l_j|$$

- cluster removal
- history based: neighborhood graph removal

51

Insertion procedures:

- Greedy (cheapest insertion)

- Max regret:

$\Delta f_i^q$  due to insert  $i$  into its best position in its  $q^{\text{th}}$  best route

$$i = \arg \max(\Delta f_i^2 - \Delta f_i^1)$$

- Constraint Programming (max 20 costumers)

53

Advantages of removal-reinsert procedure with many side constraints:

- the search space in local search may become **disconnected**
- it is easier to implement feasibility checks
- no need of computing delta functions in the objective function

Further ideas

- Adaptive removal: start by removing 1 pair and increase after a certain number of iterations
- use of roulette wheel to decide which removal and reinsertion heuristic to use

$$p_i = \frac{\pi_i}{\sum \pi_i} \quad \text{for each heuristic } i$$

- SA as accepting criterion after each reconstruction

54

55

## Outline

### 1. Construction Heuristics

Construction Heuristics for CVRP  
Construction Heuristics for VRPTW

### 2. Improvement Heuristics

### 3. Metaheuristics

### 4. Constraint Programming for VRP

## Performance of exact methods

Current limits of exact methods [Ropke, Pisinger (2007)]:

**CVRP:** up to 135 customers by branch and cut and price

**VRPTW:** 50 customers (but 1000 customers can be solved if the instance has some structure)

CP can handle easily side constraints but hardly solve VRPs with more than 30 customers.

56

57

Other approach with CP:

[Shaw, 1998]

- Use an over all local search scheme
- Moves change a large portion of the solution
- CP system is used in the exploration of such moves.
- CP used to **check the validity** of moves and determine the values of constrained variables
- As a part of checking, constraint propagation takes place. Later, iterative improvement can take advantage of the reduced domains to speed up search by performing fast legality checks.

Solution representation:

- Handled by local search:  
**Next pointers:** A variable  $n_i$  for every customer  $i$  representing the next visit performed by the same vehicle

$$n_i \in N \cup S \cup E$$

where  $S = \bigcup S_k$  and  $E = \bigcup E_k$  are additional visits for each vehicle  $k$  marking the start and the end of the route for vehicle  $k$

- Handled by the CP system: time and capacity variables.

58

59

## Insertion

by CP:

- constraint propagation rules: time windows, load and bound considerations
- search heuristic most constrained variable + least constrained valued (for each  $v$  find cheapest insertion and choose  $v$  with largest such cost)
- Complete search: ok for 15 visits (25 for VRPTW) but with heavy tails
- Limited discrepancy search

[Shaw, 1998]

```

Reinsert(RoutingPlan plan, VisitSet visits, integer discrep)
  if |visits| = 0 then
    if Cost(plan) < Cost(bestplan) then
      bestplan := plan
    end if
  else
    Visit v := ChooseFarthestVisit(visits)
    integer i := 0
    for p in rankedPositions(v) and i ≤ discrep do
      Store(plan) // Preserve plan on stack
      InsertVisit(plan, v, p)
      Reinsert(plan, visits - v, discrep - i)
      Restore(plan) // Restore plan from stack
      i := i + 1
    end for
  end if
end Reinsert
    
```

60

61