Università della Svizzera italiana

Scuola universitaria professionale della Svizzera italiana

IDSIA
Istituto Dalle Molle di studi sull'intelligenza artificiale

Advanced Algorithms

Prof. Luca Maria Gambardella

IDSIA, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
Manno, Lugano, Switzerland
www.idsia.ch

Università della Svizzera italiana

Scuola universitaria professionale della Svizzera italiana

IDSIA
Istituto Dalle Molle di studi sull'intelligenza artificiale

Research Institute (~50 people) in Lugano since 1988

Basic Research **(Swiss National Science Foundation)**

- Optimization, Machine Learning,
- Bio-Inspired Algorithms, Artificial Neural Networks
- Business week in 1997 classified IDSIA among the best 10 worldwide AI institutes

Applied Research **(CTI, European Commission, Companies)**

- Optimization in transport (multimodal terminals, fleet of vehicles) and production.
- Data Mining

Contents

Most of the real life problems are difficult (NP-hard)

Most of the problems can be represented and modeled as combinatorial optimization problems

Exact Algorithms are not effective due to time limitation and size of the search space.

Metaheuristics are new-generation heuristic algorithms to face difficult combinatorial problems whose dimensions in real life applications prevent the use exact approaches

- **Contents**:
- MetaHeuristics
  - Simulated Annealing
  - Iterated local search
  - Tabu search
  - Variable Neighborhood search
  - Genetic Algorithm
  - Ant Colony Optimization
- Traveling Salesman Problems
  - Constructive (NN, insertion, convex hull)
  - Local searches (2-opt 3-opt lin-kernighan)
  - Meta-heuristics (all)
  - Mathematical formulation
  - Branch and bound

- **Contents**:
- Sequential ordering problem (scheduling with precedence constraints and one machine)
  - Formulation and properties
  - Fast Constructive algorithms (SOP-init)
  - Local searches (SOP-3-Exchange)
  - Meta-heuristics (HAS-SOP, Maximum Partial Order/Arbitrary Insertion Genetic Algorithm), results and comparisons
- Vehicle routing problems
  - Formulation, classification and properties
  - Capacitated VRP. VRP with Time windows
  - Local searches (Cross-Exchange)
  - Meta-heuristics (MACS-VRPTW, VRP-TABU), results and comparisons

Course Contribution

Metaheuristic Algorithms - Massimo Paolucci

Nur Evin Özdemirel - IE 505 Heuristic Search

Holger H. Hoss - Thomas Stuetzle – Stochastic Local search Foundations and Applications

Combinatorial Optimization Problems

COP is an optimization problem with discrete decision variables

Definition:

Let $M=\{1,.., m\}$ a finite set, $c=(c_1,...,c_m)$ an m-vector.

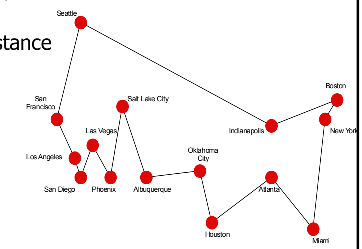
For $F \subseteq M$ let $c(F) = \sum_{j \in F} c_j$ and $\mathbf{F}$ a collection of subsets of M defined according to some rules.

Then a COP is $\min\{c(F) : F \in \mathbf{F}\}$

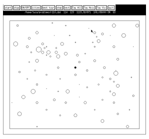
TSP: Traveling Salesman Problems

Problem: given N cities, and a distance function d between cities (usually time or kilometres), find a tour that:

- goes through every city once and only once
- minimizes the total distance



Vehicle Routing Problems



Clients
Requests
Time Windows
Pick-up and delivery
Access Limitation

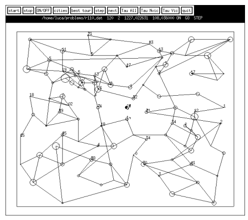
Fleet
Non-homogeneous vehicles
Costs (trucks own/external)
Drivers
Time limitation

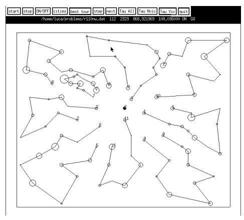
Information
Driving time, distances
Limitation on max km.
Depots, number, location

Vehicles Routing Problems

Objectives: (multiples)
- Create a set of tours
- Total distance minimization
- Travel time minimization
- Number of vehicles minimization
- Fleet optimization

… cost function minimization

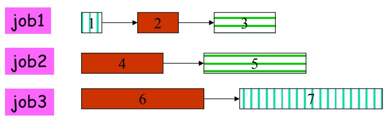




Job Shop Scheduling Problems

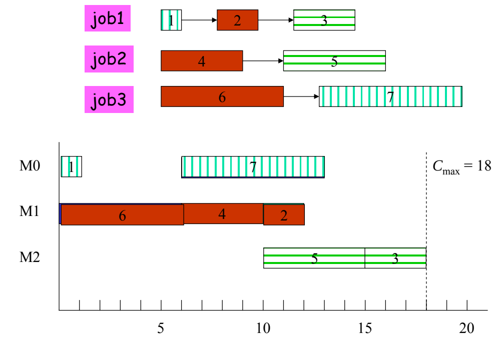
We have

- a set of resources (machines)
- a set of jobs
    - a job is a sequence of operations/activities
- sequence the activities on the resources
- An objective function to minimize



Job	Operations Processing Time				
J1	1	2	3		
J2	4	5			
J3	6	7			

Macchine	Operations		
M0	1	7	
M1	2	4	6
M2	3	5	



The goal is to assign to each operation a starting time in order to respect scheduling constraints.
Objective function: minimize the makespan (the completion time of the last operation)

COP. Are easy problems?

Direct solution

Try all the possible permutations (ordered combinations) and see which one is the cheapest (using brute force)

The number of permutations is n! (factorial on the number of cities, n)

The problem is NP-Hard

Compute the optimal solution ?

Evaluate all the possible combinations of customers and trucks

The factorial number of solutions grows as a function of 2^n

Clients	N. Solutions
2	4
4	16
8	256
16	65'536
32	4.29.E+09
64	1.84.E+19
128	3.40.E+38
256	1.16.E+77
512	1.34.E+154
1'024	1.79E+308

Time	Number of Operations		Clients
Less than 10 sec.	1'000'000'000'000	1000 mil.	40
1hour	60'000'000'000'000	6.00.E+13	46
1day	3'600'000'000'000'000	3.60.E+15	52
1 year	1'281'600'000'000'000'000	1.28.E+18	60
100 years	128'160'000'000'000'000'000	1.28.E+20	67
1000 years	1'281'600'000'000'000'000'000	1.28.E+21	70

Compute the optimal solution ?

Evaluate all the possible combinations of customers and trucks

The factorial number of solutions grows as a function of 2^n

Clients	N. Solutions
2	4
4	16
8	256
16	65'536
32	4.29.E+09
64	1.84.E+19
128	3.40.E+38
256	1.16.E+77
512	1.34.E+154
1'024	1.79E+308

Time	Number of Operations, 1000 time	faster	Clients
Less than 10 sec.	1'000'000'000'000'000	1'000'000 mil.	50
1 hour	60'000'000'000'000'000	6.00.E+16	56
1 day	3'600'000'000'000'000'000	3.60.E+18	62
1 year	1'281'600'000'000'000'000'000	1.28.E+21	70
100 years	128'160'000'000'000'000'000'000	1.28.E+23	77
1000 years	1'281'600'000'000'000'000'000'000	1.28.E+24	80

How to solve these complex problems?

1) Exact methods

- search algorithms (brute force)
- linear integer programming formulation
- search algorithm based on branch&bound
  - They guarantee to find and optimal solution but they are only applicable to problem of small size or they require long computational time.

How to solve these complex problems?

2) Heuristic and approximated algorithms

- They try to compute in a short time a solution that it is as close as possible to the optimal one.
- Sometimes, uncertainties or imprecisions in the problem parameters make the search of the optimal solution not worthy
- Therefore, it is often more practical to accept a "good" solution, hopefully not too "far" from an optimal one

How to solve these complex problems?

Heuristic/Meta-Heuristic algorithm:

An algorithm that solves an optimization problem by means of sensible rules (e.g., rules of thumb), finding a feasible solution which is not necessarily an optimal one

Approximated algorithm:

An algorithm that solves an optimization problem in polynomial time finding a feasible solution with a performance guarantee with respect to an optimal one

Approximated and heuristic algorithms

For approximated algorithms an upper bound of the distance (error) of its solutions from the optimal one must be given

Two types of errors:

Given a COP let

$Z_{OPT} = \min\{c(x) : x \in X\}$ the optimal objective value and

Z_A the objective value computed by an algorithm A

Absolute error: $E_A = Z_A - Z_{OPT}$

Relative error: $R_A = (Z_A - Z_{OPT}) / Z_{OPT}$

Approximated and heuristic algorithms

Approximated algorithms should be preferred when available

No performance guarantee is defined for heuristic algorithms

Approximated algorithms are not always available or the upper bound for the error they guarantee is not so good (e.g., ≥50%)

Design (and prove) an approximated algorithm is often difficult

Very often heuristic algorithm are preferred since they are:

- simpler to implement
- generally provide good/acceptable performance
- generally faster

Definitions

G=(V,E) is a graph where

V is a set of *nodes*

$E \subseteq V \times V$ is a set of *archs* or *edges* (i, j)

$d_{i,j}$ is the cost to go from node *i* to node *j*,

In case edges are

oriented the graph is **directed** and we talk about **digraph**

otherwise the graph is **undirected** and we talk about **graph**.

Walks, paths, tours and cycles

•A graph G=(V,E) is given where |V| = *n*

•An edge set P = $\{v_1v_2, v_2v_3, \ldots, v_{k-1}v_k\}$ is a v_1v_k **walk**. If $v_i \neq v_j$ for each $i \neq j$ than P is a v_1v_k **path**. A tour C = $\{v_1v_2, v_2v_3, \ldots, v_{k-1}v_k, v_kv_1\}$ is a **cycle**.

•**Hamiltonian cycle**: a cycle of length *n* in a graph on *n* nodes is called an hamiltonian cycle or hamiltonian tour. I.E. an hamiltonian tour visits all nodes only once and returns to the starting node

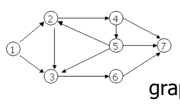
•**Eulerian tour**: a closed walk that traverses every edge of a graph exactly once.

Graphs and trees

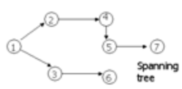
A graph G=(V,E) is **connected** if it contains for every pair of nodes a path connecting them. Otherwise is called **disconnected.** A graph G is **complete** if for all $i,j \in V$ it contains both arcs (i,j) and (j,i).

A **tree** T=(V,E) is a graph with the following properties: T is connected and T does not contain cycles.

A **spanning tree** S=(V,E) is a tree that covers all the *n* nodes in V. Each spanning tree has *n* nodes and *n*-1 edges.



graph



Combinatorial Optimization Problems

The Travelling Salesman Problem is a COP

- Given a graph $G=(V, E)$ let:
  - $M=\{1,\dots, m\}$ the set of edge indexes $E=\{e_1,\dots, e_m\}$, $m=|E|$
  - $c=(c_1,\dots,c_m)$ the edge costs
  - $\mathbf{F}$ a collection of subsets F of M such that

    $F=\{\text{an edge sequence corresponding to a hamiltonian cycle in } G\}$

Then the TSP is the COP

$$\min\{c(F) : F \in \mathbf{F}\}$$

25

Combinatorial Optimization Problems

The Travelling Salesman Problem is a COP (2)

- Given a graph $G=(V, E)$ let:
  - $N=\{1,\dots, n\}$ the set of vertex indexes $V=\{v_1,\dots, v_n\}$, $n=|V|$
  - $D=[d_{ij}]$ an $n\times n$ distance matrix
  - $\mathbf{F}$ a collection of subsets F of V such that

    $F=\{\text{a cyclic permutation } \pi \text{ of } n \text{ items}\}$
  - $\pi(i)$ the vertex visited after vertex i in π
  - $c(F) = c(\pi) = \sum_{j=1}^{n} d_{j\pi(j)} \quad \forall F \in \mathbf{F}$

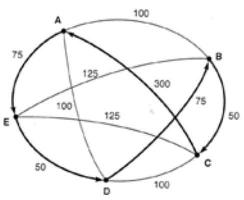
Then the TSP is the COP $\min\{c(F) : F \in \mathbf{F}\}$

26

Most studied COP
TSP: Traveling Salesman Problems

Problem: given N cities, and a distance function d between all couples of cities (usually time or kilometres), find a tour that:

- goes through every city once and only once
- minimizes the total distance



27

Traveling Salesman Problems

Symmetric TSP: given a complete graph G=(V,E) with edge weight d_{ij}, find a shortest Hamiltonian tour in G.

A symmetric TSP is said to satisfy the triangle inequality if $d_{ij} \leq d_{ik} + d_{kj}$ for all distinct nodes i,j,k

Of particular interest are the **metric TSP** where nodes corresponds to points in some space and edge weights are given by evaluating some metric distance between corresponding points. For example the **Euclidean TSP** is defined by a set of points the the plane. The correspondent graph contains a node for every point and edge weights are given by the Euclidean distance of the points associated with the end nodes

Asymmetric TSP: given a complete digraph G=(V,E) for some edge $d_{ij} \neq d_{ij}$. Find a shortest Hamiltonian tour in G.

28

Traveling Salesman Problems

A game as first TSP example

Hamilton's Icosian Game (1800)

It is required to complete a tour along 20 points with a restricted number of connections

Hamilton's Iconsian game

29

TSP history

- First description in 1800 by the Irish mathematician Sir William Rowan Hamilton and the British mathematician Thomas Penyngton Kirkman.
- The general form is presented for the first time in the mathematic studies in 1930 by Karl Menger in Vienna and Harvard. The problem was also promoted by Whitney and Merrill Flood a Princeton.
- A detailed description of Menger and Whitney work and of TSP diffusion can be found in Alexander Schrijver "*On the history of combinatorial optimization*", 1960.

30

TSP History

•A breakthrough by George Dantzig, Ray Fulkerson, and Selmer Johnson in 1954.

•49 - 120 – 550 - 2,392 - 7,397 – 19,509 cities. From year 1954 to year 2001.

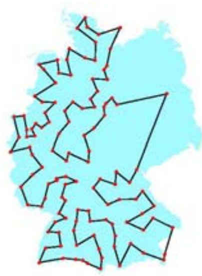
•24,098 cities by David Applegate, Robert Bixby, Vasek Chvatal, William Cook, and Keld Helsgaun in May 2004.

TSP instances

years	Research team	Problem size
1954	G.Dantzig, R. Fulkerson, and S. Johnson	49 cities
1971	M. Held and R.M.Karp	64 cities
1975	P.M.Camerini, L. Fratta, and F. Maffioli	100 cities
1977	M.Grötschel	120 cities
1980	H.Crowder and M.W.Padberg	318 cities
1987	M.Padberg and G.Rinaldi	532 cities
1987	M. Grötschel and O.Holland	666 cities
1987	M. Padberg and G.Rinaldi	2.392 cities
1994	D.Applegate, R.Bixby, V.Chvàtal, e W.Cook	7.397 cities
1998	D.Applegate, R.Bixby, V.Chvàtal, e W.Cook	13.509 cities
2001	D.Applegate, R.Bixby, V.Chvàtal, e W.Cook	15.112 cities
2004	D.Applegate, R.Bixby, V.Chvàtal, e W.Cook	24.978 cities

1954
G.Dantzig, R. Fulkerson, and S. Johnson
49 città

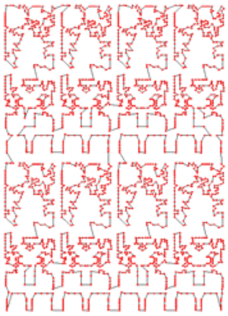
1977
M.Grotschel
120 città



1987
M.Padberg e G.Rinaldi
532 città

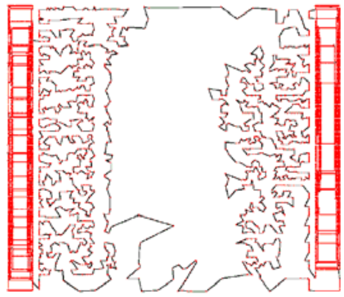
1987
M. Grötschel e O.Holland
666 città

1987
M.Padberg e G.Rinaldi
2.392 città



37

1994
D.Applegate, R.Bixby, V.Chvàtal, e W.Cook
7.397



38

1998
D.Applegate, R.Bixby, V.Chvàtal, e W.Cook
13.509

39

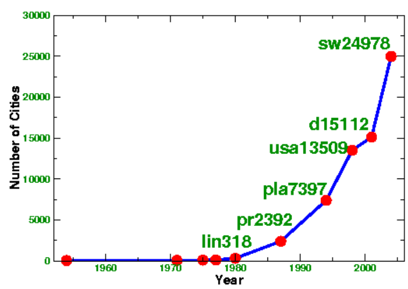
2001
D.Applegate, R.Bixby, V.Chvàtal, e W.Cook
15.112

40

2004
D.Applegate, R.Bixby, V.Chvàtal, e W.Cook
24.978 cities in Sweden

41

Major progress due to Concorde software available in http://www.tsp.gatech.edu/index.html



TSPLIB library with hundred of benchmark problems

42

Complete Search approach: model and solve

1. Model the problem as a state space (usually a graph)
2. Search for the solution (with certain properties e.g. min/max objective function) using a search strategy in the state space (usually a tree)
3. The solution is a sequence of states

Problem definition

States: the set of possible problem configurations

Initial state: the state where the search process starts.

Actions: Operators: state → { state }
Set of all possible actions

Goal: A function GOAL?: state → {*true, false*}
It check if a given state is a goal

Cost function: gives a cost to the solution path

Search Algorithm

A search algorithm takes as input a problem space and a starting state and tries to compute a path (solution) in the best possible way.

The algorithm produces a search tree over the problem space (or state space) that it is *usually a graph*

Strategy: search which node to *expand* among the nodes not yet been explored. (This is the *fringe* = leaves of the search tree)

To expand a node means to consider all nodes reachable in one step (one action) from the selected node

General search algorithm

Solution: is a sequence of operators that bring you from current state to the goal state

Basic idea: offline, systematic exploration of simulated state-space by generating successors of explored states (expanding)

```
Function General-Search(problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state problem
   loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding
   solution
        else expand the node and add resulting nodes to the search tree
   end
```

Strategy: The search strategy is determined by the **order** in which the nodes are expanded.

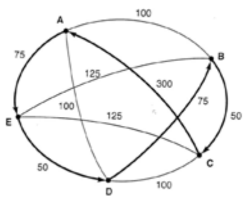
Search Tree

A *node* in the search tree has five components:

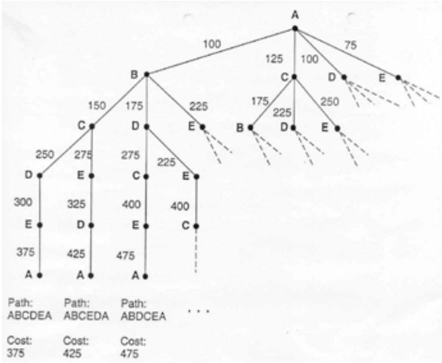
- A state
- The node who has generated it
- The action used to generate it
- The depth of the tree
- The cost of the path from the root

Traveling Salesman Problem

Goal: to visit all the cities only once
States: cities, costs on the edge should be the same in the two directions (Symmetric TSP) or different (Asymmetric TSP)
Initial state: a city
Actions: to travel from one city to another city
Cost Function: sum of the edges on the traveled tour



Traveling Salesman Problem



Breadth-first search

Expand shallowest unexpanded node

Implementation:
QUEUEINGFN = put successors at end of queue

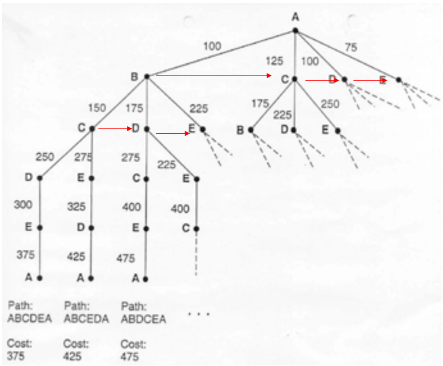
Breadth-first search, before visiting the children of a node it visits his brothers.

The search tree is expanded in breadth.

Nodes at distance *d* from the root are expanded before nodes at distance *d*+1.

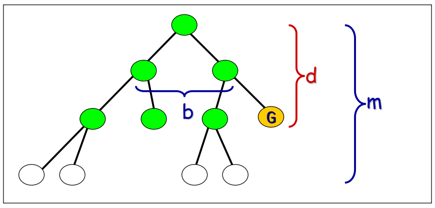
In order to obtain this behavior breadth-first search uses as Open data structure a FIFO queue.

Traveling Salesman Problem



Space complexity of breadth-first

If a goal node is found on depth d of the tree, all nodes up till that depth are created.



Thus: O(**b^d**)

Breadth-first search

Based on a FIFO data structure

b = branching factor
d = solution depth

Expanded nodes: $1 + b + b^2 + \dots + b^{d-1} + b^d \rightarrow O(b^d)$

Complexity in time and space: $O(b^d)$

Breadth-first search

Example: *b*=10; 1000 nodes expanded x second;
1 node use 100 byte

Depth	nodes	Time	Memory
0	1	1 millisec	100 byte
2	111	0.1 sec	11 Kilobyte
4	11111	11 sec	1 megabyte
6	10^6	18 min	111 megabyte
8	10^8	31 hours	11 gigabyte
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabyte
14	10^{14}	3500 years	11111 terabyte

Depth-first search

Expand deepest unexpanded node

Implementation:

QUEUEINGFN = insert successors at front of queue

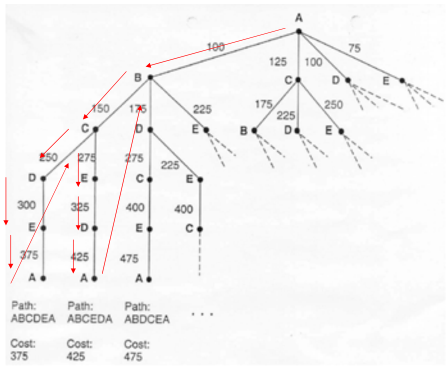
At each step we expand a node generated immediately in the previous step.

First version is based on a list (open) which contains nodes still to be expanded (this is our search fringe) .

Open is managed following to LIFO procedure

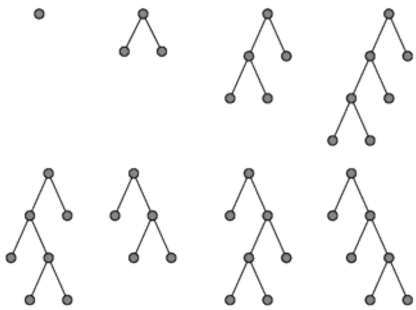
55

Traveling Salesman Problem



56

Depth-first data structure



57

Properties of depth-first search

- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

Remember:

b = branching factor

m = max depth of search tree

58

Heuristic Algorithms (1)

Basic Heuristics

They fast (in polinomial time) produce a **feasible solution** to the problem by constructive a solution from scratch or by the modification of a starting solution

This is not considered as a real optimization process.

This is a fast way to produce a feasible (good) solution

59

Heuristic Algorithms (2)

MetaHeuristics procedures

They start from a solution (or a set of solutions)

This solution(s) is(are) iteratively modified using stochastic processes.

Previous results are used to update the search and to generate new better solutions.

This is an optimization procedure

60

Basic heuristic algorithms

Two main kinds of classic heuristics:

Constructive heuristics

Build the solution step by step at each iteration

Examples (TSP): Nearest Neighbourhood, Insertion, Christofides alg.

Improvement heuristics

Start from a complete feasible solution and try at each iteration to improve it

Examples (TSP): 2-OPT, 3-OPT, Lin-Kernigham

Note that this classification is not comprehensive
E.g., Lagrangean heuristics basically found non-feasible solutions that try to improve towards feasibility

61

Heuristics for TSP

For large instances (or when short time is available) is not possible to use exact algorithms.

It is needed to approximate the optimal solution with heuristic approaches

Heuristic comes from the Greek Euristikein = discovery

Complexity from **$O(n^2)$** e **$O(n^4 \log n)$**.

62

Constructive algorithms

1. Start from a random node (not a complete solution)
2. Expand the starting node generating all possible next nodes (not yet included in the partial solution).
3. Choose the best next node according to a local strategy
4. Extend the solution with this new node. This node become the new starting node.
5. Iteratively adds element to the partial solution (going back to point 2) until a feasible solution is computed.

63

Nearest Neighbour algorithm

Proposed by Flood (1956) is one of the most common for solving TSP and ATSP problems.

Given n cities:

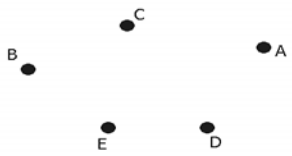
1. Consider a starting tour made by a random city a_1;
2. When the current tour is $a_1,\dots,a_k$ with $k<n$, be a_{k+1} the city that does not belong to the tour and that is **closest to a_k**: a_{k+1} Is added at the end of the tour
3. When no more cities are available we stop the procedure.

64

Nearest Neighbour (1/2)

Starting from E

	A	B	C	D	E
A	-	13	8	10	14
B	13	-	4	7	5
C	8	4	-	6	4
D	10	7	9	-	2
E	14	5	4	2	-

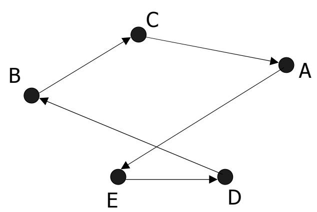


Example from Ercoli C., Re B., Progetto TSP, Università di Camerino, 2003-2004

65

Nearest Neighbour (2/2)

	A	B	C	D	E
A	-	13	8	10	14
B	13	-	4	7	5
C	8	4	-	6	4
D	10	7	9	-	2
E	14	5	4	2	-



66

Nearest Neighbour: conclusions

- The algorithm is not very efficient. The first edges are very short while the final edges are usually very long
- In general the length of the tour in relation with the optimal tour length grows following a **log n** formula
- Computational complexity is

Figure 1. The Nearest Neighbor heuristic.

Nearest Neighbour: conclusions

- The algorithm is not very efficient. The first edges are very short while the final edges are usually very long
- In general the length of the tour in relation with the optimal tour length grows following a **log n** formula
- Computational complexity is **$O(n^2)$.**

Figure 1. The Nearest Neighbor heuristic.

Nearest Neighbour : results for random problems

Problem	10^2	10^3	10^4	10^5	10^6
% Error Over the Held&Karp lower bound	25.6	26.2	24.3	23.6	23.3

D.S. Johnson and L.A. McGeoch, 1997.

Nearest Neighbour : results for TSP LIB problems

Problem	Variant 1
d198	25.79
lin318	26.85
fl417	21.28
pcb442	21.36
u574	29.60
p654	31.02
rat783	27.13
pr1002	24.35
u1060	30.43
pcb1173	28.18
d1291	22.97
rl1323	22.30
fl1400	42.42
u1432	25.50
fl1577	27.65
d1655	25.99
vm1748	25.67
rl1889	28.37
u2152	25.80
pr2392	24.96
pcb3038	23.63
fl3795	24.44
fnl4461	25.31
rl5934	22.93
Average	26.27

G. Reinelt, 1994.

Greedy Heuristic

The Greedy heuristic gradually constructs a tour by repeatedly selecting the shortest edge and adding it to the tour as long as it doesn't create a cycle with less than N edges, or increases the degree of any node to more than 2. We must not add the same edge twice of course.

procedure TSP_greedy

(1) Sort E_n such that $c_1 \leq c_2 \leq \ldots, \leq c_m$.

(2) Set $T = \emptyset$.

(3) For $i = 1, 2, \ldots, m$:

(3.1) If $T \cup \{e_i\}$ can be extended to a Hamiltonian tour (or is a Hamiltonian tour), then set $T = T \cup \{e_i\}$.

end of TSP_greedy

Complexity $O(n^2 \log(n))$

Multi-fragment greedy heuristic

1. We start with the shortest edge ad we add the edges in increasing order only if they do not create a 3-degree city

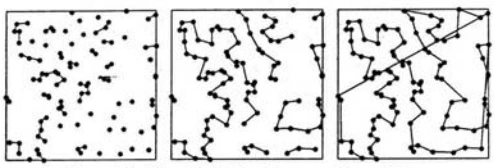


Figure 5. The Multiple Fragment heuristic.

Improvement heuristics:

Enlarging a feasible initial solution

Starts from a feasible solution (a tour) in a **subset** of the search space iteratively adds element to the partial solution according to some strategy until a feasible results is computed.

Usually it has better performance than greedy constructive procedures

Insertion heuristics

procedure insertion

(1) Select a starting tour through k nodes $v_1, v_2, \ldots, v_k$ ($k \geq 1$) and set $W = V \setminus \{v_1, v_2, \ldots, v_k\}$.

(2) As long as $W \neq \emptyset$ do the following.

(2.1) Select a node $j \in W$ according to some criterion.

(2.2) Insert j at some position in the tour and set $W = W \setminus \{j\}$.

end of insertion

For $j \in W$, let

$$d_{\min}(j) = \min\{c_{ij} \mid i \in V \setminus W\}$$

$$d_{\max}(j) = \max\{c_{ij} \mid i \in V \setminus W\}$$

$$s(j) = \sum_{i \in V \setminus W} c_{ij}$$

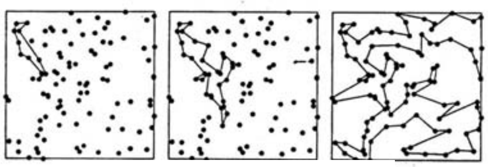
Insertion heuristics

1. Nearest insertion: insert the node that has the shortest distance to a tour node, i.e. select j with

   $d_{min}(j)$ = **min**$\{d_{min}(l) \mid l \in W\}$

   1. Build an initial tour W with cities i_1 e i_2 such that

   $$c_{i1i2} + c_{i2i1} = \min_{i \neq j} (c_{ij} + c_{ji})$$



c.

Insertion heuristics

2 Farthest insertion 1: insert the node whose minimal distance to a tour node is maximal, i.e. select

$d_{min}(j)$ = **max**$\{d_{min}(l) \mid l \in W\}$

3 Farthest insertion 2: insert the node that has the farthest distance to a tour node, i.e. select.

$d_{max}(j)$ = **max**$\{d_{max}(l) \mid l \in W\}$

4 Farthest insertion 3: insert the node whose maximal distance to a tour node is minimal, i.e. select

$d_{max}(j)$ = **min**$\{d_{max}(l) \mid l \in W\}$

Insertion heuristics

5 Cheapest insertion 1: choose the node whose insertion causes the lowest increase in the tour length (update of best insertion points for non-tour nodes after each insertion is expensive)

6. Cheapest insertion 2: only partial update of best insertion points

7. Random insertion: select the node to be inserted at random

Insertion heuristics

8. Largest sum insertion: insert the node whose sum of distances to tour nodes is maximal, i.e. select j with

   $s(j)$ = **max**$\{s(l) \mid l \in W\}$

9. Smallest sum insertion: insert the node whose sum of distances to tour nodes is minimal, i.e. select j with

   $s(j)$ = **min**$\{s(l) \mid l \in W\}$

The selected node is usually inserted at the point causing shortest increase in the tour length (there are other rules)

Insertion heuristics

All standard versions (except cheapest insertion) run in $O(n^2)$

Cheapest insertion can run in
$O(n^2 \log n)$, but requires $O(n^2)$ memory

Nearest and cheapest insertion tours are less than twice as long as optimal when triangle inequality is satisfied

Random and farthest insertion can be 13/2 times longer than optimal

Insertion heuristics

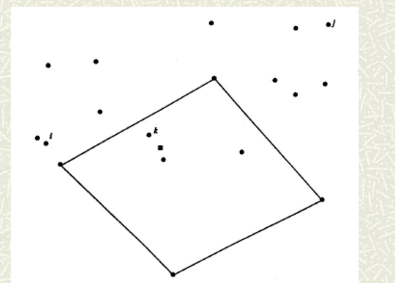


Figure 6.10 Illustration of insertion heuristics

Nearest insertion adds node i, farthest adds node j, cheapest adds node k

Comparison of standard insertions:
Percent deviation of tour length from best lower bound

Problem	1	2	3	4	5	6	7	8	9
d198	13.19	3.85*	7.57	14.80	11.08	11.08	8.17	8.15	7.78
lin318	21.62	10.87	18.41	24.30	18.39	18.39	9.18*	20.02	16.27
fl417	12.50	5.48	13.37	13.37	12.39	12.39	3.29*	7.84	9.04
pcb442	20.89	13.83	16.99	29.06	21.15	21.15	12.23*	27.07	20.43
u574	22.33	11.39*	22.68	26.32	19.12	19.12	11.64	23.32	22.21
p654	10.81	6.89	11.33	5.94	5.79*	5.79*	9.87	11.30	12.64
rat783	23.04	12.09*	22.52	28.72	16.02	16.02	13.37	26.37	25.02
pr1002	18.57	10.85*	24.81	27.24	16.61	16.61	12.50	23.98	25.42
u1060	21.39	12.68	21.52	27.55	18.67	18.67	11.43*	23.94	21.58
pcb1173	25.84	14.22*	26.82	32.67	21.50	21.50	16.58	29.56	28.80
d1291	22.90	23.78	27.29	29.50	17.01*	17.01*	22.13	31.06	18.70
rl1323	31.01	18.89*	29.30	27.80	24.81	24.81	20.64	29.30	26.56
fl1400	20.28	8.45*	14.56	24.78	17.98	17.76	8.47	16.30	16.44
u1432	15.26	12.59*	20.43	20.08	12.65	12.65	12.63	23.84	20.54
fl1577	21.61	15.17*	20.04	25.21	17.08	17.08	18.70	26.66	17.97
d1655	20.18	17.09*	22.22	27.80	18.83	18.77	17.69	28.20	23.95
vm1748	21.26	13.54*	25.37	33.59	18.86	18.86	13.87	29.52	24.26
rl1889	23.82	19.10	27.74	32.70	21.24	21.24	17.30*	29.99	27.53
u2152	21.09	19.55	28.64	32.84	16.12*	16.12*	19.76	28.26	28.98
pr2392	24.70	14.32*	28.26	33.55	20.50	20.50	16.65	31.75	28.32
pcb3038	23.12	14.89*	24.54	27.84	17.08	17.08	16.69	27.57	27.28
fl3795	19.61	21.97	19.58	29.45	12.79*	12.79*	19.77	21.62	25.62
fnl4461	21.10	12.03*	27.69	28.90	15.97	15.97	12.99	28.99	28.03
rl5934	27.40	22.17	30.12	33.42	21.84*	21.84*	22.71	33.56	30.36
Average	20.98	13.99	22.16	26.56	17.23	17.22	14.51	24.51	22.24

Table 6.11 Results of insertion heuristics

CPU time for Insertion Heursitics

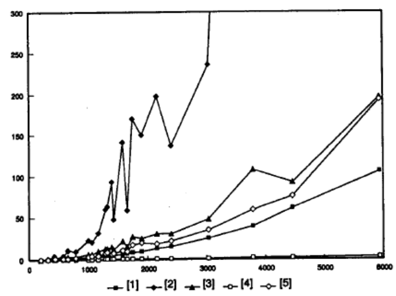
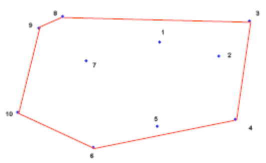
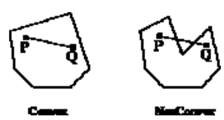


Figure 6.13 CPU times for some insertion heuristics

Comparison of standard insertions with convex hull start





Comparison of standard insertions with convex hull start

Problem	1	2	3	4	5	6	7	8	9
d198	12.86	6.73	6.41	6.58	8.51	8.51	4.37*	7.03	7.34
lin318	15.08	10.82	19.70	16.82	11.42	11.42	7.97*	18.01	16.39
fl417	14.24	5.28	14.95	5.65	7.61	7.61	2.77 *	6.83	8.36
pcb442	16.52	10.37*	17.54	18.89	11.83	11.83	13.62	19.17	22.71
u574	17.24	9.95*	23.55	18.47	14.67	14.67	10.73	19.83	20.62
p654	17.07	3.05*	12.40	6.38	8.15	8.15	6.49	13.16	10.85
rat783	16.90	12.72	24.68	23.89	15.16	15.16	11.90*	22.31	23.80
pr1002	20.05	11.10*	25.65	21.66	14.23	14.23	13.27	26.71	21.76
u1060	22.78	10.69	24.71	22.79	16.65	16.65	10.39*	23.99	22.65
pcb1173	21.61	15.44*	26.14	26.62	19.18	19.18	18.25	28.45	26.35
d1291	25.58	21.80	25.52	26.22	14.69*	14.69*	21.03	22.06	22.37
rl1323	25.86	15.10*	28.57	25.74	20.30	20.30	20.18	27.73	27.59
fl1400	14.04	5.79*	14.62	12.05	9.73	9.73	8.35	13.69	17.90
u1432	15.34	12.65	21.06	18.73	11.73*	11.73*	13.19	22.48	21.66
fl1577	20.30	15.18*	18.72	28.25	18.09	18.09	15.58	37.73	24.71
d1655	20.94	15.05	21.55	27.26	13.23*	13.23*	15.99	26.67	24.03
vm1748	19.83	10.77*	25.31	24.04	16.94	16.94	12.03	26.52	25.48
rl1889	25.74	17.98*	29.40	31.63	18.72	18.72	18.15	30.07	27.26
u2152	19.03	18.26	29.05	27.32	13.98*	13.98*	19.73	27.73	27.05
pr2392	21.26	15.24*	28.88	27.07	17.52	17.52	15.83	29.87	26.18
pcb3038	22.41	14.31*	25.57	24.62	16.47	16.47	15.44	25.81	27.12
fl3795	24.06	21.60	16.23	27.65	13.81*	13.81*	18.35	25.40	19.04
fnl4461	22.21	11.94*	29.49	27.94	15.42	15.42	13.07	29.14	27.67
rl5934	26.54	20.28*	30.27	31.89	21.24	21.24	21.71	29.66	29.58
Average	19.90	13.00	22.50	22.01	14.55	14.55	13.68	23.34	22.02

Table 6.14 Results of insertion heuristics with convex hull start

Heuristics using spanning trees

Based on the observation that, given a Eulerian tour containing all nodes, if the triangle inequality is satisfied then we can derive a Hamiltonian tour which is not longer than the Eulerian tour

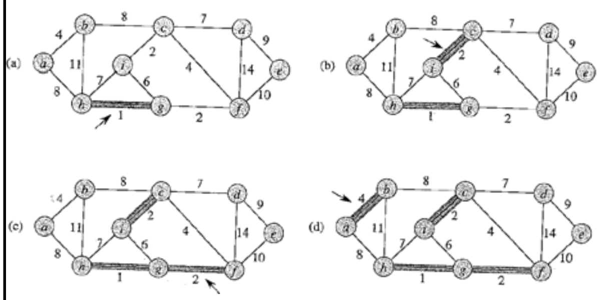
Hence, particularly useful when triangle inequality holds

Minimum Spanning tree (Kruskal)

Kruskal(G,w)
A = ∅
For each vertex v ∈ V[G]
    do Make-Set (v)
Sort the edges of E by non decreasing weight w
For each edge (e,v) ∈ E, in order by non decreasing weight
    do if Find-Set(u) ≠ Find-Set(u) then
            A := A ∪ {(u,v)}
            Union(u,v)
Return A

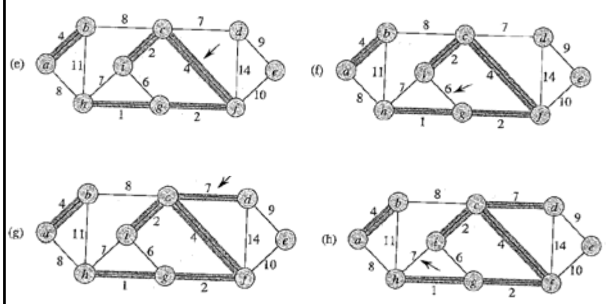
Edge (u,v) is incrementally added to the forest if their two endpoints do not belong to the same set (i.e. they do not create a cycle)

Minimum Spanning tree (Kruskal)

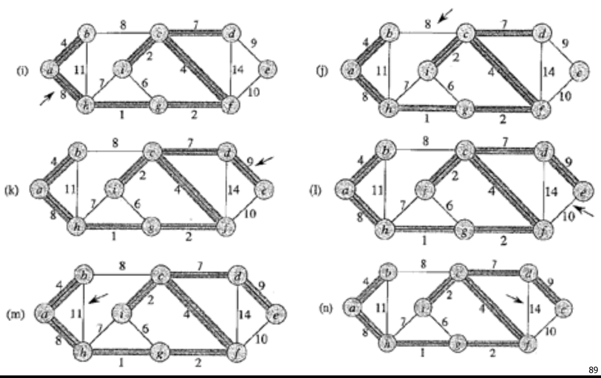


Example from Introduction to Algorithms, Cormen et all, MIT press, 1991

Minimum Spanning tree (Kruskal)



Minimum Spanning tree (Kruskal)



Approximation algorithms for TSP

These algorithms produces feasible solutions in a “short time”.

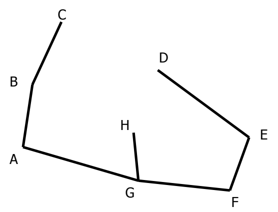
The first algorithm is for Euclidean TSP and it is based on the mentioned MST

Approx2-TSP-tour(G)
    Select a vertex r ∈ V[G] to be a root vertex
    Grow a minimum spanning tree T from G from root r
    Let L the list of vertices visited in a preorder tree walk of T
    Return the Hamiltonian cycle H that visit the vertex in the order of L

Approximation algorithms for TSP

Starting vertex is A

Here is the minimum spanning tree T

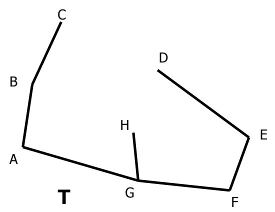


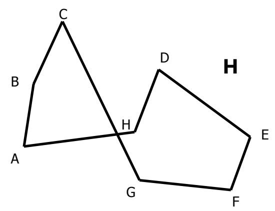
	A	B	C	D	E	F	G	H
A	0	11	24	25	30	29	15	15
B	11	0	13	20	32	37	17	17
C	24	13	0	16	30	39	29	22
D	25	20	16	0	15	23	18	12
E	30	32	30	15	0	9	23	15
F	29	37	39	23	9	0	14	21
G	15	17	29	18	23	14	0	7
H	15	17	22	12	15	21	7	0

Approximation algorithms for TSP

a walk W on T gives the following W=ABCBAGFEDEFGHGA

A preorder walk on T list the vertex when they are first encountered PW=ABCGFEDH that produces the tour H





Approximation algorithms for TSP

The mentioned algorithm guarantees that the cost of the solution c(H) $\leq$ 2*C(best_solution)

Since T is a minimum spanning tree we have

c(T) $\leq$ C(best_solution)

The full walk W=ABCBAGFEDEFGHGA traverses every edges exactly twice

c(W) = 2C(T)

so

C(W) $\leq$ 2*C(best_solution)

but W is not usually a tour since he visits some vertex more than one

Approximation algorithms for TSP

However by the triangle inequality we can delete a visit to any vertex from W and the cost does not increase

If a vertex v is deleted from W between u and w the resulting ordering specifies going directly from u to w

Appling this operation we can remove from W all but the first visit to each vertex.

In our example this leaves the order ABCGFEDH that is the same of the preorder PW.

Approximation algorithms for TSP

Let H be the cycle corresponding to this preorder walk.

This is exactly the Hamilton cycle produced by the algorithm Approx2-TSP-tour so we have

C(H)$\leq$C(W) $\leq$ 2*C(best_solution)

In spite of the nice ratio bound and his **Complexity O(V^2)** this algorithm is not so effective in practice. Other approaches are usually used.

Savings method

- Originally developed for VRP (Clarke and Wright, 1964)
- Starting with n-1 two-node tours all connected to a base node, merges short subtours to obtain a Hamiltonian tour
- The crucial point is to find the best merging possibility
- Runs in O(n^3), or O(n^2log n) with O(n^2) memory to store the matrix of possible savings

Clarke-Wright Saving Heuristic (1964). A constructive procedure proposed for VRP

procedure savings

(1) Select a base node $b \in V$ and set up the $n-1$ tours (b, v), $v \in V \setminus \{b\}$ consisting of two nodes each.

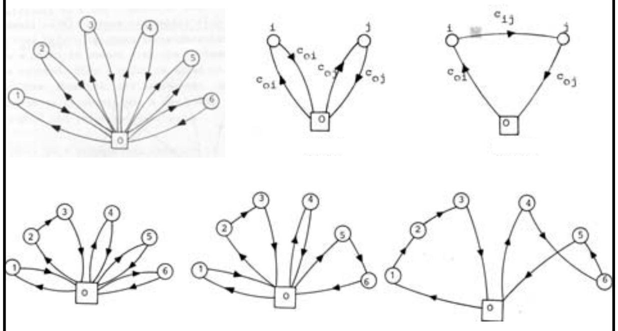
(2) As long as more than one tour is left perform the following steps.

(2.1) For every pair of tours T_1 and T_2 compute the savings that is achieved if the tours are merged by deleting in each tour an edge to the base node and connecting the two open ends. More precisely, if ub and vb are edges in different tours then these tours can be merged by eliminating ub and vb and adding edge uv resulting in a savings of $c_{ub} + c_{vb} - c_{uv}$.

(2.2) Merge the two tours giving the largest savings.

end of savings

Clarke-Wright Saving Heuristic



(Fiala 1978)

Saving Vs Greedy Heuristic

Problem	Standard	Fast version	Greedy
d198	8.72	6.96	6.69*
lin318	8.14*	8.34	14.13
fl417	11.63	13.74	9.48*
pcb442	10.05*	10.20	14.62
u574	11.62	12.36	10.20*
p654	10.64*	10.66	18.82
rat783	10.06	9.88*	10.04
pr1002	11.24	10.24*	12.96
u1060	11.79	11.69*	12.16
pcb1173	9.98*	10.53	13.66
d1291	7.77	7.55*	9.89
rl1323	7.48*	8.07	8.13
fl1400	12.12*	14.41	16.30
u1432	10.27*	10.41	15.04
fl1577	14.43	15.90	8.63*
d1655	10.97	12.39	10.49*
vm1748	13.66	13.68	10.49*
rl1889	13.65	13.32	10.82*
u2152	10.69	10.67*	12.10
pr2392	12.24*	12.40	13.42
pcb3038	10.73	10.51*	13.28
fl3795	16.52	16.25	14.60*
fnl4461	10.84	10.65*	11.09
rl5934	11.36	12.53	10.04*
Average	11.11	11.39	11.96

Table 6.20 Results for savings heuristics and greedy

Saving Vs Greedy Heuristic: Time

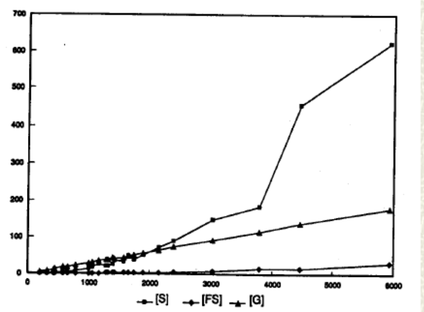


Figure 6.21 CPU times for savings heuristics and greedy

Comparison of TSP heuristics
(with best solution by any other heuristic)

Heuristic	No. of best solutions	Relative quality
Savings (standard)	5	1.96
Savings (fast)	6	2.22
Savings (approx. greedy)	5	2.75
Farthest insertion 1 (convhull)	3	3.64
Random insertion (convhull)	3	4.26
Farthest insertion 1	1	4.55
Random insertion	–	5.03
Cheapest insertion (convhull)	–	5.08
Fast cheapest insertion (convhull)	–	5.08
Fast cheapest insertion	–	7.53
Cheapest insertion	1	7.54
Christofides	–	9.67
Nearest insertion (convhull)	–	9.99
Nearest insertion	–	10.98
Nearest neigbor variant 4	–	11.55
Farthest insertion 3 (convhull)	–	11.83
Minimum sum insertion (convhull)	–	11.88
Farthest insertion 2	–	12.02
Minimum sum insertion	–	12.07
Farthest insertion 2 (convhull)	–	12.33
Farthest insertion (fast)	–	12.81
Maximum sum insertion (convhull)	–	13.07
Random insertion (fast)	–	13.18
Maximum sum insertion	–	14.14
Maximum sum insertion (fast)	–	14.31
Farthest insertion 2 (fast)	–	14.37
Farthest insertion 3 (fast)	–	15.66
Farthest insertion 3	–	16.03
Nearest neigbor variant 2	–	16.20
Nearest neigbor variant 3	–	16.61
Standard nearest neigbor	–	16.83
Nearest neigbor variant 5	–	17.77
Nearest insertion (fast)	–	18.93
Minimum sum insertion (fast)	–	19.60
Cheapest insertion (fast)	–	23.39
Fast cheapest insertion (fast)	–	23.94
Double tree	–	27.90

TSP Heuristics

Table IV. Local Optimization Applied to Each Heuristic

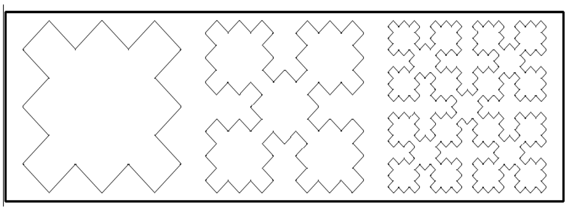
Heuristic Name	Percent Over Lower Bound Start	CPU Seconds Start
nn	24.2	4
denn	24.2	4
mf	15.7	14
na	26.9	26
fa	13.2	38
ra	15.2	16
ni	26.8	46
fi	13.0	76
ri	14.8	57
mst	44.5	16
ch	14.9	24
frp	55.2	2

(Bentley 1992)

NN=Nearest Neighbourhood, *DENN*=Double Ended NN, *MF*=Multiple Fragment, *NA, FA, RA*=Nearest, Farthest, Random Addition, *NI, FI, RI*=Nearest, Farthest, Random Insertion, *MST*=Min. Spanning Three, *CH*=Christofides, *FRP*=Fast Recursive Partition.

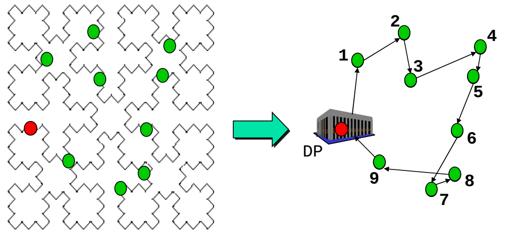
SPACE FILLING CURVE (something differente!!)

"A space filling curve is a continuous mapping from a lower-dimensional space into a higher-dimensional one. A famous space filling curve (due to Sierpinski), is formed by repeatedly copying and shrinking a simple pattern"



SPACE FILLING CURVE

Constructive technique: overlap the space filling area with cities. Each point is associated to the closest line. Following the line the visit order is determined.



SPACE FILLING CURVE

A TSP tour of 15,112 cities in Germany.

This tour was induced by the Sierpinski spacefilling curve in less than a second and is about 1/3 again as long as the shortest possible.

Notice that for the optimal solution the computation was carried out on a network of 110 processors. The total computer time used in the computation was 22.6 years, scaled to a Compaq EV6 Alpha processor running at 500 MHz.

Local search algorithms

We now start from a **complete solution**

A is the search space, i.e. all problem solutions

We have an objective function min $\{f(s) \mid s \in A\}$

We define a *neighborhood function N*
N is a mapping from $A \rightarrow 2^A$ that defines for each solution $s \in A$ a subset of solutions $N(s) \in A$, the *neighborhood* of s.

Local Search (LS)

LS algorithm is basically an improvement heuristic

LS starts from a feasible initial solution and tries to improve it by exploring the solution neighbourhood

LS iterates the exploration step from the new solution until no further improvement is possible

LS is a descent method: it founds a local optimum

The computation time needed by LS (improvement heuristics) is generally much longer than the one of constructive algorithms

LS for COP needs a proper definition of the neighbourhood of solutions

Hill climbing

The algorithm explores the entire neighborhood and search always for a better solution until no improvement is possible

For each solution *current* it generates and evaluates all the neighborhoods *N*(*current*)

Greedy search: in case the best solution in *N*(*current*) is better than the actual best we restart from *current* (random search in case of conflict) otherwise we stop

Simplified version: as soon as we found a better neighborhood we continue the search from the associated solution avoiding to visit all the neighborhoods

Hill climbing

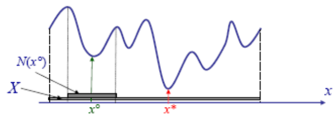
```
Hill climbing
    input
    initial solution s_start
    objective function f
    neighborhood function N

    current ← s_start
    while terminal condition is met (usually no
improvement or time limit)
        next ← the best solution in N(current)
                if   f(next)   <   f(current)
current ← next
    end while
    output current
```

Greedy algorithm is only based on local information.
It is not able to escape from local minimum

Local Search: Local and global optimum

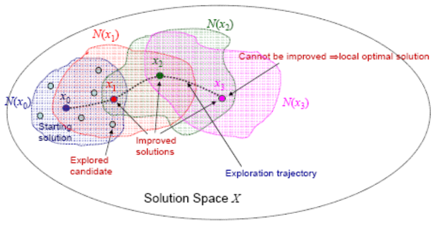
☐ Local optimum (min)
A locally optimal solution (or local optimum) with respect to a neighbourhood structure N(x) is a solution x° such that
$\forall x \in N(x°)\ Z(x°) \leq Z(x)$

☐ Global optimum (min)
A global optimal solution (or global optimum) is a solution x* such that $\forall\ x \in X\ Z(x^*) \leq Z(x)$



Local Search

☐ Basic LS tracks a trajectory in the solution space, from a feasible solution to another, until no improvement is found



Local Search

☐ 2-OPT, 3-OPT and Lin-Kerninghan are example of LS based improvement heuristics for TSP

☐ 2-OPT, 3-OPT and Lin-Kerninghan differ for the kind of neighbourhood they explore

☐ Several variations exist for the basic LS applied to COP:
- Selection of the next solution strategy
  - Best improvement (complete exploration of N(x))
  - First improvement (partial exploration)
- Neighbourhood exploration strategy
  - Complete exploration of N(x)
  - Candidate List Strategy (define a smaller N'(x)⊆N(x))
  - Final intensification
- Termination criterion
  - Maximum number of iterations
  - Maximum CPU time

Local Search – N(x)

Comments:
The larger is |N(x)| the more likely is the possibility of finding a high quality solution

The larger is |N(x)| the higher is the computational time required

A trade-off between solution quality and exploration time is needed

Techniques have been proposed to deeply explore neighbourhood of exponential dimension in polynomial time (e.g. Dynasearch)

Local Search – N(x)

Comments:

The dimension of a neighbourhood can also dynamically varied:

|N(x)| is enlarged when no improvements is found after a fixed number of iterations (e.g., Variable Neighbourhood Descent)

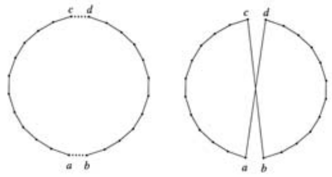
Nevertheless the **main drawback** of LS is its propensity to be trapped in a (possibly bad) local optimal solution

Local Search K-Opt

1. Start from complete tour computed by another heuristic
2. Compute the best (the first) k edge exchange that improves the tour.
3. Execute this exchange
4. Search for another exchange until no improvement is possible

115

2-opt



GAIN = (a,d)+(b,c)-(c,d)-(a,b)
Subpath (b,…,d) is reverted

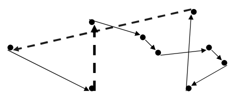
Computational complexity **O(n²)** .

116

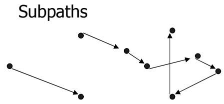
2 – OPT: example (1/3)

- - - - Edges to be removed

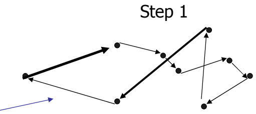
Initial Tour



Subpaths



Step 1



Subtour is inverted

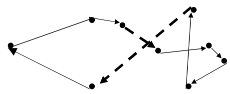
New edges

117

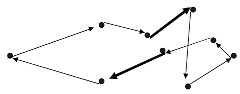
2 – OPT: example (2/3)

- - - - Edges to b e removed

Step 1



Step 2



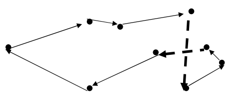
New edges

118

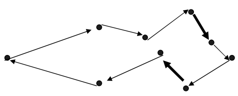
2 – OPT: example (3/3)

- - - - Edges to be removed

Step 2



Step 3



New edges

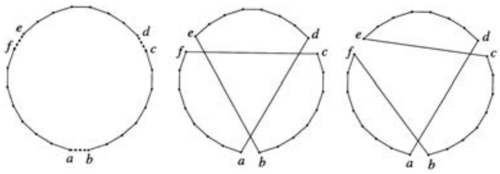
119

2-opt

While best_gain≠0
    best_gain=0
    For i = 1 **to** n
        For j = 1 **to** n
            gain=compute_gain(i,j)
            if (gain<best_gain) **then**
                best_gain=gain
                best_i=i
                best_j=j
                if first_improvement=TRUE **then** break
        End for
        if (best_gain<0 & first_improvement=TRUE) **then** break
    End for
    exchange(best_i,best_j)
End while

120

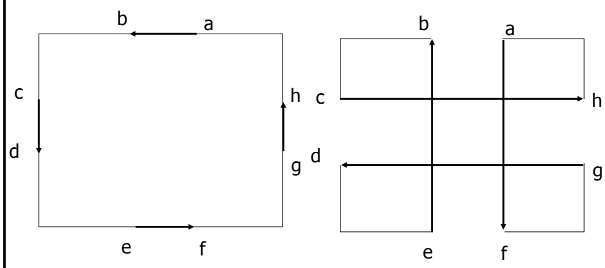
3-opt



Two possible new tours
GAIN1 = (a,d)+(e,b)+(c,f)-(a,b)-(c,d)-(e,f) no path is reverted
GAIN2 = (a,d)+(e,c)+(b,f)-(a,b)-(c,d)-(e-f) path (c,…,b) is reverted

Computational complexity **O(n³)** .

Double bridge



Does not invert subtours
Computational complexity **O(n²)** .

2-opt 3opt : results for random problems

Table IV. Local Optimization Applied to Each Heuristic

Heuristic Name	Percent Over Lower Bound				CPU Seconds			
	Start	2-Opt	2H-Opt	3-Opt	Start	2-Opt	2H-Opt	3-Opt
nn	24.2	8.7	6.8	4.5	4	27	28	54
denn	24.2	8.6	6.7	4.6	4	28	28	57
mf	15.7	5.8	4.7	3.5	14	30	33	55
na	26.9	16.9	11.2	6.9	26	45	47	83
fa	13.2	11.8	9.6	6.9	38	52	52	78
ra	15.2	12.0	9.8	6.8	16	31	31	56
ni	26.8	16.9	11.3	6.8	46	65	67	101
fi	13.0	11.9	9.7	6.9	76	89	89	112
ri	14.8	12.3	9.9	7.0	57	72	72	97
mst	44.5	12.8	9.3	5.6	16	44	45	80
ch	14.9	6.7	4.9	3.8	24	40	40	60
frp	55.2	14.9	10.5	5.8	2	35	34	73

(Bentley 1992)

NN=Nearest Neighbourhood, *DENN*=Double Ended NN, *MF*=Multiple Fragment, *NA, FA, RA*=Nearest, Farthest, Random Addition, *NI, FI, RI*=Nearest, Farthest, Random Insertion, *MST*=Min. Spanning Three, *CH*=Christofides, *FRP*=Fast Recursive Partition.

2-opt for TSPLIB problems

Problem	Random	Nearest N.	Savings
d198	8.04	3.18*	5.29
lin318	13.05	5.94*	8.43
fl417	12.25	7.25	5.38*
pcb442	12.64	7.82	7.70*
u574	14.24	7.02*	8.82
p654	12.40	12.37	8.66*
rat783	12.31	8.39	8.03*
pr1002	14.91	8.48*	9.07
u1060	13.05	9.11	8.94*
pcb1173	12.85	9.42	7.78*
d1291	17.72	9.62	6.22*
rl1323	15.89	7.88	6.56*
fl1400	12.50	9.79	8.85*
u1432	14.24	10.07	8.83*
fl1577	21.42	8.15*	12.59
d1655	16.42	8.29*	12.36
vm1748	12.74	8.58*	9.20
rl1889	14.22	8.64	8.55*
u2152	19.89	10.02	9.64*
pr2392	16.20	8.27*	9.57
pcb3038	16.29	8.34*	8.36
fl3795	13.52	8.57*	11.37
fnl4461	14.09	7.77*	8.90
rl5934	21.07	9.19*	10.98
Average	14.67	8.42	8.75

From different starting tours

G. Reinelt, 1994.

3opt for TSPLIB problems

Problem	Random	Nearest N.	Savings	Christofides
d198	2.86	5.27	1.69	1.24*
lin318	2.93	2.80	2.16*	4.62
fl417	5.90	3.68	0.62*	5.24
pcb442	5.67	1.66*	2.54	3.03
u574	5.83	3.98	4.21	3.07*
p654	7.69	3.80	1.06*	5.78
rat783	4.60	3.47*	4.26	3.88
pr1002	4.45	3.61	4.02	3.24*
u1060	7.06	6.17	4.04	3.20*
pcb1173	5.82	5.70	3.93*	4.29
d1291	14.40	4.26	2.74*	6.30
rl1323	7.61	4.51	3.43	3.35*
fl1400	10.06	6.34	3.86*	7.20
u1432	7.47	4.83	3.51	3.36*
fl1577	15.85	8.40	8.71	4.89*
d1655	9.43	4.23*	5.09	5.17
vm1748	5.98	6.05	3.84	3.69*
rl1889	8.22	6.91	5.54	3.78*
u2152	9.82	5.78	4.85*	4.62
pr2392	7.15	4.37	4.94	3.62*
pcb3038	6.82	4.12	4.63	4.09*
fl3795	17.48	10.27	7.35	4.53*
fnl4461	4.77	3.45	5.13	3.40*
rl5934	14.39	6.37	6.74	4.11*
Average	8.01	5.00	4.12	4.15

From different starting tours

G. Reinelt, 1994.

Lin-Kernighan for TSPLIB problems

Problem	Random	Nearest N.	Savings	Christofides
d198	0.75*	5.55	1.48	1.03
lin318	1.68	2.48	1.64	1.44*
fl417	3.10	0.61*	1.29	2.74
pcb442	1.49	1.95	2.33	1.33*
u574	1.96	2.48	2.11	0.93*
p654	1.71	3.07	0.05*	1.12
rat783	2.07	2.03*	2.92	3.21
pr1002	3.06	2.69	2.77	2.30*
u1060	2.88	2.41	2.73	1.78*
pcb1173	2.87	2.64	2.65	2.41*
d1291	4.77	4.51	2.97*	3.26
rl1323	2.25	2.68	1.79*	2.90
fl1400	2.30*	3.16	3.89	5.14
u1432	2.34	2.29	2.04*	2.20
fl1577	6.38	10.90	6.27	2.29*
d1655	2.63*	3.54	3.85	3.27
vm1748	2.11	2.00*	2.80	2.60
rl1889	2.46	3.45	3.90	2.35*
u2152	3.88	3.00	4.33	2.49*
pr2392	3.17	3.05	3.15	2.04*
pcb3038	2.50	1.82*	2.67	2.58
fl3795	3.46*	6.81	3.55	3.64
fnl4461	2.06	1.98*	2.47	2.26
rl5934	3.27	2.39	3.55	2.40*
Average	2.71	3.23	2.80	2.40

From different starting tours

G. Reinelt, 1994.

How to speed up the search

Also with approximation algorithms for very large problems the time required to compute feasible solution is too large

The objective is to reduce the search space, the running time and to keep high quality solutions.

Two methods are presented:

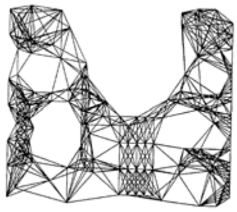
Static approach: nearest-neighbor-list

Dynamic approach: don't look bit

Nearest-neighbor list

For each node we compute a priori the set of the n-nearest-neighbor-list

These are the n nodes that are closest (according to some metric) to the actual node



nearest-neighbor-list with n=10 for the problem u159

Nearest-neighbor list

All the previous algorithms can be executed only considering the restricted set given by the nearest-neighbor-list

A reasonable number for n is between 15 to 20

For random Euclidean TSP increasing n from 20 to 80 only improves the final tour of 0.1%

Fast insertions with candidate list:

Percent deviation of tour length from best lower bound

Problem	1	2	3	4	5	6	7	8	9
d198	15.31	7.84*	9.41	13.67	13.47	13.47	10.42	13.80	11.13
lin318	25.69	20.03	18.85	23.78	42.95	42.95	18.09	17.63*	24.41
fl417	36.20	31.08	33.92	44.57	24.36*	24.36*	26.82	38.90	31.99
pcb442	28.85	18.59	20.16	19.93	29.66	29.66	20.07	14.08*	27.33
u574	22.54	17.34*	19.97	18.60	25.28	25.28	17.53	19.02	26.72
p654	48.59	46.22	36.84*	42.62	78.21	78.81	49.36	44.91	54.21
rat783	26.07	15.35*	18.31	20.00	24.90	24.90	17.47	16.11	29.58
pr1002	19.89*	21.30	29.43	20.37	26.54	26.50	22.52	20.74	28.17
u1060	25.39	17.54*	20.42	20.78	22.95	24.07	18.52	19.97	25.55
pcb1173	28.93	19.28*	21.60	21.87	34.27	34.27	21.84	22.42	28.86
d1291	31.24	25.33	26.61	27.29	20.91	20.73*	24.78	26.81	28.16
rl1323	37.34	22.46*	26.82	32.97	31.19	31.43	26.04	31.16	35.37
fl1400	30.83	31.66	28.69	29.67	85.17	94.98	19.07*	27.06	30.59
u1432	21.61	17.81	20.29	20.27	28.08	29.89	20.25	16.51*	25.52
fl1577	34.75	27.27	28.95	28.19	31.09	31.09	23.67*	29.51	36.09
d1655	28.95	23.22	23.74	26.05	33.35	35.48	22.40*	24.38	29.23
vm1748	26.05	21.07*	21.82	23.34	22.90	22.90	22.27	21.20	29.31
rl1889	35.45	25.60*	29.58	30.32	42.91	42.39	31.51	28.60	35.12
u2152	28.99	24.68	28.89	24.46	21.34*	21.34*	25.03	25.06	30.82
pr2392	27.01	23.14*	27.88	28.22	35.15	32.68	24.56	24.28	31.41
pcb3038	25.19	18.48*	21.47	19.67	25.61	25.72	20.05	20.00	28.57
fl3795	35.77	24.96*	29.32	30.18	40.31	40.62	25.80	33.85	32.41
fnl4461	23.47	16.88*	17.23	20.27	31.74	36.16	17.64	18.11	28.51
rl5934	44.63	31.26*	29.81	35.55	51.60	48.17	32.91	33.31	37.97
Average	29.53	22.85	24.58	25.94	34.33	34.91	23.28	24.48	30.29

Table 6.12 Results of fast insertion heuristics

From local search to meta heuristics

Local search procedures explores in a systematic way the neighborhood of a given solution

The goal is to search the best move and to execute it.

It is usually efficient but it is not able to escape from local minimum

In same case the neighborhood is to large

A way to solve the mentioned problems is the following:

1) Stochastically explore only a subset of the neighborhood.

2) Accept solutions that are worst than the previous

Meta-Heuristic Algorithms

There is no unique definition for Metaheuristic (MH) Algorithms:

- MHs are strategies to guide the exploration of a solution (search) space
- The term metaheuristic (Glover, 1986) was used to denote a high level strategy that iterates a lower level heuristic whose parameters are progressively updated
- The first MHs were developed to overcome the drawbacks of LS algorithm
- Metaheuristic is used also to denote modern heuristics

The best way to start MHs understanding is to analyze their main (common) characteristics and to define a classification

Meta-Heuristic Algorithms

Two possible definitions:

"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." (Osman and Laporte 1996)

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method." (Voß et al. 1999)

133

Meta-Heuristic Algorithms

Characteristics (Blum and Roli, 2003):

- MHs are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near) optimal solutions.
- Techniques which constitute MH algorithms range from simple local search procedures to complex learning processes.
- MH algorithms are approximate and usually non-deterministic.
- MH may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of MHs permit an abstract level description.
- MHs are not problem-specific.
- MHs may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Todays more advanced MHs use search experience (embodied in some form of memory) to guide the search.

134

Meta-Heuristics

MH "philosophies":

Intelligent extensions of LS algorithms (Trajectory methods):
The goal is to escape from local minima in order to proceed in the exploration of the search space and to move on to find other hopefully better local minima. They use one or more neighbourhood structure(s) Examples: Tabu Search, Iterated Local Search, Variable Neighbourhood Search, GRASP and Simulated Annealing

Use of learning components (Learning Population-based methods):
They implicitly or explicitly try to learn correlations between decision variables to identify high quality areas in the search space. They perform a biased sampling of the search space Examples: Ant Colony Optimization, Particle Swarm Optimization, Genetic Algorithms and Evolutionary Computation.

135

Meta-Heuristics

MH "philosophies":

Intelligent extensions of LS algorithms (Trajectory methods):
- The goal is to escape from local minima in order to proceed in the exploration of the search space and to move on to find other hopefully better local minima.
- They use one or more neighbourhood structure(s)
- Examples: Tabu Search, Iterated Local Search, Variable Neighbourhood Search, GRASP and Simulated Annealing

Use of learning components (Learning Population-based methods):
- They implicitly or explicitly try to learn correlations between decision variables to identify high quality areas in the search space.
- They perform a biased sampling of the search space
- Examples: Ant Colony Optimization, Particle Swarm Optimization, Genetic Algorithms and Evolutionary Computation.

136

Meta-Heuristic Algorithms

Possible MH classifications:

- Nature-inspired vs non-nature inspired
- Population-based vs single point search
- Dynamic vs static objective function
- One vs various neighbourhood structures
- Memory usage vs memory-less methods

137

Meta-Heuristic Algorithms

MHs outline:

Trajectory methods:

Simulated Annealing

Tabu Search

Variable Neighbourhood Search

Population-based methods:

Evolutionary Computation (Genetic Algorithms)

Ant Colony Optimization

Particle Swarm Optimization

138

MetaHeuristic search (Trajectory Methods)

```
Meta Heuristic search
    input
    initial solution s_start (or an initial set of solutions)
    objective function f
    neighborhood function N
    current ← s_start (current should also be a set of solutions)

    while terminal condition is met
            stochastically compute a solution next ∈ N(current)
                    Following a criterium decide whether or not
               to continue the search from next
               by setting current ← next
    end while

    output current
```

Very efficient: we keep active only one solution (or a set) but we do not have any guarantee to reach the optimum

Simulated Annealing [Kirkpatrick, Gelatt, Vecchi 1983]

Simulated Annealing is an MH method that tries to avoid local optima by accepting probabilistically moves to worse solutions.

Simulated Annealing was one of the first MH methods

now a "mature" MH method
- many applications available (ca. 1,000 papers)
- (strong) convergence results

simple to implement

inspired by an analogy to physical annealing of metals

Simulated Annealing

Annealing is a thermal process for obtaining low energy states of a solid through a heat bath.
1. increase the temperature of the solid until it melds
2. carefully decrease the temperature of the solid to reach a ground state (minimal energy state, cristaline structure)

Computer simulations of the annealing process
- models exist for this process based on Monte Carlo techniques
- Metropolis algorithm: simulation algorithm for the annealing process proposed by Metropolis et al. in 1953

Simulated Annealing [Kirkpatrick, Gelatt, Vecchi 1983]

It starts from an initial *current* solution

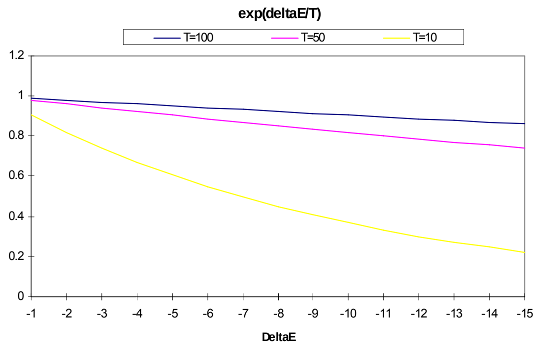
At each iteration a new solution *next* is randomly chosen from the neighborhoods of the *current* solution

If *f*(*next*) < *f*(*current*) we start the next iteration from *next*

Otherwise the choice between *next* and *current* is done in using a probabilistic function $e^{-\Delta E/T}$ that is based on $\Delta E = f(next) - f(current)$ and on a parameter T (temperature) that decreases during the search

Simulated Annealing(problem**) return** a solution
- *T* ← determine a starting temperature
- *current* ← generate an initial solution
- *best* ← current
- **While** not yet frozen **do**
  - **While** not yet at equilibrium for this temperature **do**
    - *next* ← a random solution selected from *Neigh*(*current*)
    - ΔE ← f(*next*) - f(*current*)
    - **if** ΔE<0 **then** *current* ← *next*
      - **if** f(*next*) < f(*best*) **then** *best* ← *next*
    - **else**
      - choose a random number *r* uniformly from [0.1]
      - **if** *r* < $e^{-\Delta E/T}$ **then** *current* ← *next*
  - **end while**
  - lower the temperature *T*
- **end while**
- Return *best*

Simulated Annealing



Simulated Annealing

T is usually decremented with a formula where $T_{i+1}=T_i$*const where const in most applications is close to 0.95

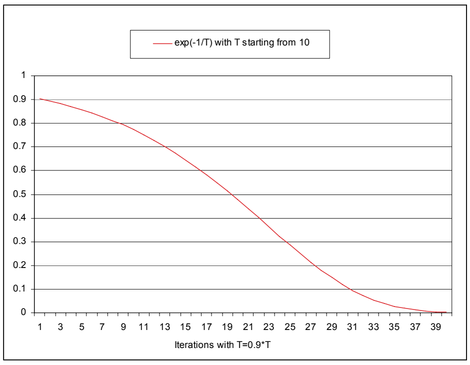
For Euclidean instances initial temperature is usually based on [Bonomi and Lutton, 1984] $\frac{L}{\sqrt{n}}$

Johnson proposes $\frac{1.5 \bullet L}{\sqrt{n}}$ that allows an initial acceptance rate of about 50%

Temperature length (steps from one temperature to the next) is usually computed by α*NN_list_length with α varying from 1 to 100

For TSP applications the neighborhood is usually given by a random 2-opt move

Simulated Annealing



Simulated Annealing

		Random Euclidean Instances					
		Average Percent Excess			Running Time in Seconds		
Variant		10^2	$10^{2.5}$	10^3	10^2	$10^{2.5}$	10^3
SA_1 (Baseline Annealing)	$\alpha=1$	3.4	3.7	4.0	12.40	188.00	3170.00
SA_1 + Pruning	$\alpha=1$	2.7	3.2	3.8	3.20	18.00	81.00
SA_1 + Pruning	$\alpha=10$	1.7	1.9	2.2	32.00	155.00	758.00
SA_2 (Pruning + Low Temp)	$\alpha=10$	1.6	1.8	2.0	14.30	50.30	229.00
SA_2	$\alpha=40$	1.3	1.5	1.7	58.00	204.00	805.00
SA_2	$\alpha=100$	1.1	1.3	1.6	141.00	655.00	1910.00
2-Opt		4.5	4.8	4.9	0.03	0.09	0.34
Best of 1000 2-Opts		1.9	2.8	3.6	6.60	16.20	52.00
Best of 10000 2-Opts		1.7	2.6	3.4	66.00	161.00	517.00
3-Opt		2.5	2.5	3.1	0.04	0.11	0.41
Best of 1000 3-Opts		1.0	1.3	2.1	11.30	33.00	104.00
Best of 10000 3-Opts		0.9	1.2	1.9	113.00	326.00	1040.00
Lin-Kernighan		1.5	1.7	2.0	0.06	0.20	0.77
Best of 100 LK's		0.9	1.0	1.4	4.10	14.50	48.00

D.S. Johnson and L.A. McGeoch, 1997.

Tabu Search [Glover, 1989]

Like simulated annealing moves from one solution to the neighborhood but avoiding inverse transformation that would bring us in previous solutions

A deterministic method first introduced by Glover (1986)

TS explicitly uses the history of the search, both to escape from local minima and to implement an explorative strategy

TS is an extended LS since it can continue the exploration after a local optimal solution is found

The Tabu List (TL) is a short-term memory to escape from local Optima

Tabu Search

A tabu-list TL with the recent moves is maintained. Tabu moves are forbidden for a certain number of steps

We choose the **best** allowed move

Basic TS

```
x ← GenerateInitialSolution( )
TabuList ← ∅
while termination conditions not met
x ← ChooseBestOf(N(x)\TabuList )
Update(TabuList)
endwhile
```

TL keeps memory of the recent search history

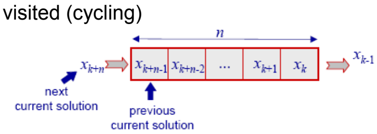
The next solution may not improve the current one

Tabu Search

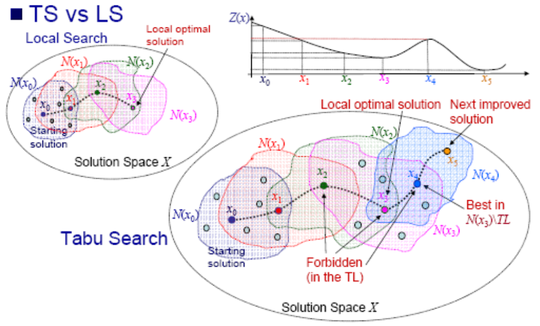
The TL restricts the neighbourhood of the current solution

Allowed(x) = N(x)\TabuList

The Tabu List:
- a FIFO list
- store information about the latest solutions of the exploration trajectory
- used to forbid the selection of solutions recently visited (cycling)



Tabu Search



151

Tabu Search

The TL restricts the neighbourhood of the current solution

TL prevents from returning to recently visited solutions (cycling)

TL forces the search to accept even uphill moves

The tabu tenure controls the memory of the search process:
Small → the search concentrates on small areas of the search space
Large → the search process is forced to explore larger regions

The tabu tenure can be fixed or varied during the search

152

Tabu Search

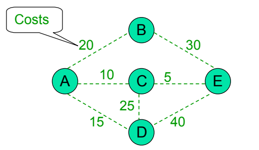
- Storing complete solutions in the TL is highly inefficient
- TL usually stores solution attributes :
  - •solution components
  - •moves
  - •differences between two solutions
- A single or more attributes → a single or more TLs
- The set of TLs define the tabu conditions filtering N(x)

- **Aspiration criteria**: allow promising solutions that are forbidden
  - •Best Objective criterion

153

Tabu Search

An example: minimum spanning tree with additional constraints (NP hard) (Glover and Laguna, 1997)



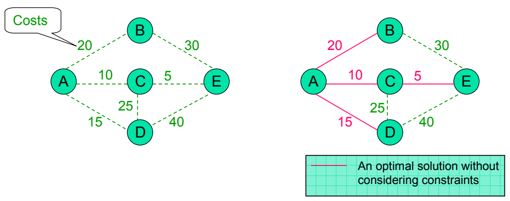
TS model
- $N(x)$ is defined by edge exchange moves
- TL: the inserted edge
- Tabu tenure = 2
- Aspiration criterion: Best Objective
- Penalty for a single constraint violation= 50

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

154

Example

- Minimum spanning tree problem with constraints.
- *Objective: Connects all nodes with minimum costs*

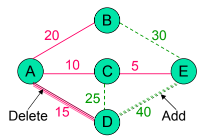


Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

155

Example

Iteration 1
Cost=50+200 (constraint penalties)



Add	Delete	Cost
BE	CE	75+200=275
BE	AC	70+200=270
BE	AB	60+100=160
CD	AD	60+100=160
CD	AC	65+300=365
DE	CE	85+100=185
DE	AC	80+100=180
DE	AD	**75+0=75**

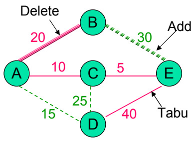
New cost = 75 (iteration 2)
(local optimum)

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

156

Example

Tabu list: DE
Iteration 2   Cost=75



Add	Delete	Cost
AD	DE*	Tabu move
AD	CE	85+100=185
AD	AC	80+100=180
BE	CE	100+0=100
BE	AC	95+0=95
BE	AB	**85+0=85**
CD	DE*	60+100=160
CD	CE	95+100=195

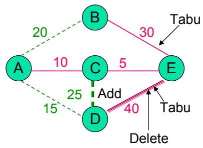
**** A tabu move will be considered only if it would result in a better solution than the best trial solution found previously (Aspiration Condition)***

Iteration 3 new cost = 85 Escape local optimum

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

Example

Tabu list: DE & BE
Iteration 3 Cost=85



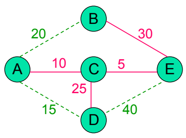
Add	Delete	Cost
AB	BE*	Tabu move
AB	CE	100+0=100
AB	AC	95+0=95
AD	DE*	60+100=160
AD	CE	95+0=95
AD	AC	90+0=90
CD	DE*	**70+0=70**
CD	CE	105+0=105

**** A tabu move will be considered only if it would result in a better solution than the best trial solution found previously (Aspiration Condition)***

Iteration 4 new cost = **70** Override tabu status

Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included.
(Penalty of 100 if selected two of the three, 200 if all three are selected.)

Example



Optimal Solution
Cost = 70
Additional iterations only find inferior solutions

Tabu Search

Candidate List Strategies (CLS):
- Used to heuristically restrict the N(x) dimension to the subset of most promising solutions (e.g., execute the moves that should produce the greater improvements)

Long-Term Memory (LTM) can be used for :
- Storing *elite* complete solutions: quality solutions whose improvement require a great number of iterations
- Storing solution attributes frequent appeared during the search

TS may include two mechanisms based on LTM:
- **Intensification**: a thorough LS is finally executed starting from elite solutions (especially if CLS are used)
- **Diversification**: it forces the search to abandon the already visited regions of the solution space after a fixed number of iteration without any improvements (non-improving iterations)

Tabu Search

Step 1.
k=1, *Nstep*= 0, Create an initial solution S_1; $S_{best} = S_1$

Step 2.
At step k select the best { $S_c \in$Neighborhood(S_k):
notViolateTabuConditions or SadisfyAspirationCriteria}
If $F(S_c) < F(S_{best})$ **then** $S_{best} = S_c$, *Nstep* = 0**,** go to Step 3 (aspiration criteria)
Nstep= *Nstep* + 1
If the move $S_k \rightarrow S_c$ is not forbidden
then $S_{k+1} = S_c$
insert the inverted move in the tabu list
remove the last tabu move from the tabu-llist
If $F(S_c) < F(S_{best})$ **then** $S_{best} = S_c$, *Nstep* = 0
If *Nstep* > MaxNonImprovingIteration **then** Diversification()
Go to Step 3.

Step 3.
$k = k$+1 ;
If *stopping condition* = true **then** STOP
else go to Step 2

Tabu search

Tabu moves

Usually 2-opt moves

Example of tabu list

The two removed edges in a 2-opt (avoid to insert them again)

The shortest edge involved in a 2-opt (avoid a move that involves this edge)

The endpoints involved in a 2-opt (avoid a move that uses one of them)

How to go beyand LocalSearch? Random Restart

Iteratively random generate solution s independently

Apply local search to s obtaining s*

practically not very effective

for large instances leads to costs with
- fixed percentage excess above optimum
- distribution becomes arbitrarily peaked around the mean in the instance size limit

163

Random restart policy for TSP

The main idea is to use a local improvement algorithm such as 2-opt, 3-opt or LK and to iteratively restart the search for different random starting points until a local optimum is reached

The performance gain is usually not so good due to the limited capability of each run to increase the starting solution

For instance 100 runs of 2-opt on a 100-city random geometric instance will be typically better than an average 3-opt

For 1000-city instance the best 100 runs of 2-opt is typically worse than the worst 100 runs of 3-opt

164

Iterated local search

How to improve the search?

Iterated local search (ILS) is an MH method that generates a sequence of solutions generated by an embedded heuristic, leading to far better results than if one were to use repeated random trials of that heuristic.

simple principle
easy to implement
state-of-the-art results
long history

165

Iterated local search: notation

S: set of (candidate) solutions

s: solution in *S*

f: cost function

f(*s*): cost function value of solution

s*: locally optimal solution

S*: set of locally optimal solutions

LocalSearch defines mapping from S → S*

166

Iterated local search

The ideas is to search in S*

LocalSearch leads from a large space S to a smaller space S*

define a biased walk in S*

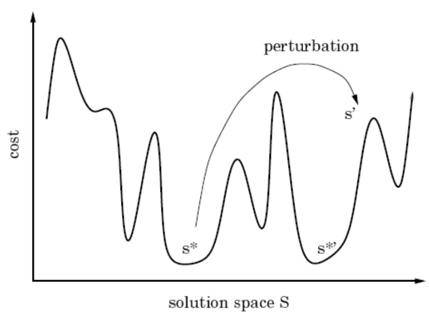
given a solution s* perturb it s* → s'

apply LocalSearch: s' → s*'

apply acceptance test: s*, s*' → s^*_{new}

167

Iterated local search



168

Iterated local search

Procedure IteratedLocalSearch

s=Random generate a solution
s*= LocalSearch(s)

Loop until a terminal condition is met
  s' = Perturbation (s*, history)
  s*'= LocalSearch(s')
  s* = AcceptanceCriterion (s*, s*', history)

End Loop

169

Iterated local search

Performance depends on interaction among all modules

basic version of ILS

GenerateInitialSolution: random or construction heuristic
LocalSearch: often readily available
Perturbation: random move in higher order neighborhood
AcceptanceCriterion: force cost to decrease

basic version often leads to very good performance
basic version only requires few lines of additional code
state-of-the-art results with further optimizations

170

Iterated local search for TSP

GenerateInitialSolution: greedy heuristic

LocalSearch: 2-opt, 3-opt, LK, (whatever available)

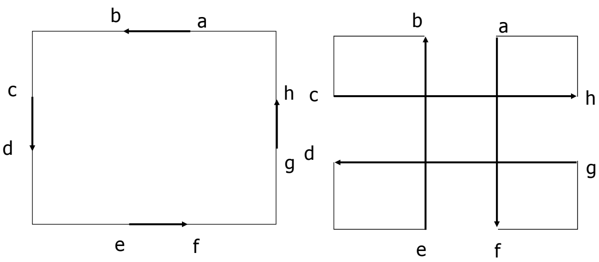
Perturbation: double-bridge move (a 4-opt move)
Double bridge for its non-sequential nature can not be easily reverted by 3-opt or lin-kernighan

AcceptanceCriterion: accept s*' only if f(s*')≤f(s*)

171

Iterated local search

Perturbation used by Martin, Otto, Felten, 1991
is called double-bridge a 4-opt move.



172

ILS is a modular approach

Optimization of individual modules

complexity can be added step-by-step

different implementation possibilities

Optimize single modules without considering interactions among modules
  → **local optimization** of ILS

global optimization of ILS has to take into account interactions among components

173

ILS Initial Solution

determines starting point s_0* of walk in S*

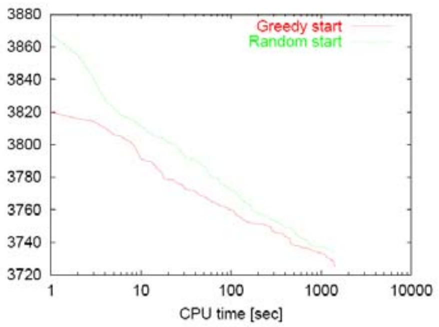
random vs. greedy initial solution

greedy initial solutions appear to be better

for long runs dependence on s_0* should be very low

174

ILS Initial Solution



ILS Perturbation

Important: strength of perturbation
- too strong: close to random restart
- too weak: LocalSearch may undo perturbation

strength of perturbation may vary at run-time

perturbation should be complementary to LocalSearch

Adaptive perturbations
- single perturbation size not necessarily optimal
- perturbation size may vary at run-time
  - basic Variable Neighborhood Search
- perturbation size may be adapted at run-time
  - reactive search

Iterated local search for TSP

Results were quite promising (with a 3-opt local search with don't look bits).

The lin318 problem was solved in an hour on a SparcStation 1 (4-6 minutes on a SGI Challenge)

For the att532 it could get within 0.07% from the optimal in 15 hours

Johnson reports finding optimal solution for lin318, pcb442, att532, gr666, pr1002 and pr2392 with a lin-kernighan local search

The idea to hybridize meta-heuristics with local search is currently one of the most effective idea to solve TSPs.

Comparison for TSP

Table 5: Library instances: Comparison of local search heuristics.

Problem	ID L-K		TS L-K		Markov L-K
lin318	0.33		0.35		0.22
att532	0.31		0.22		0.01
rat783	0.60		0.28		0.06
pr2392	1.15		0.68		0.32

Average excess (%) from optimum of five independent runs.
ID: Iterated descent.
TS: Tabu search (disc. 4-Change in ascent mode).
Markov: Large-Step Markov chains [Martin et al. 1992].

Variable Neighborhood Search (VNS)

Proposed by Hansen and Mladenovc (1999, 2001)

Variable Neighborhood Search is an MH method that is based on the systematic change of the neighborhood during the search.

central observations

a local minimum w.r.t. one neighborhood structure is not necessarily locally minimal w.r.t. another neighborhood structure a global optimum is locally optimal w.r.t. all neighborhood structures

Variable Neighborhood Search

principle: change the neighborhood during the search

several adaptations of this central principle
- variable neighborhood descent
- basic variable neighborhood search
- reduced variable neighborhood search
- variable neighborhood decomposition search

notation

N_k, k=1, …..k_{max} is a set of neighborhood structures

$N_k(s)$ is the set of solutions in the k-th neighborhood of s

Population Based Heuristics

Common characteristics:
At every iteration search process considers a set (a population) of solutions instead of a single one

The performance of the algorithms depends on the way the solution population is manipulated

Take inspiration from natural phenomena

Three approaches:
- Evolutionary Computation (Genetic Algorithms)
- Ant Colony Optimization
- Particle Swarm Optimization

Genetic Algorithms

They are based on the Darwin theory of evolution

Individuals that better fit with the environment have more chance to survive

Auto-organization as in the biological systems

Evolution as natural selection mechanism

Populations of individuals move from one generation to the next.

Genetic Algorithms

Individual reproduction capabilities are "proportional" to their ability to fit with the environment

Reproduction allows the best individual to generate children similar to them

Generation after generation the population always fit better with the environment

The environment is the objective function (**fitness**) to optimize, and the individuals are a population of solutions.

Genetic Algorithms

Introduced by Holland (1960) at UNI Michigan

It is a parallel search in the solution space, where the search is driven by past experiences

Components
individual is described by his *chromosome*.
chromosome is defined by a set (sequence) of *genes.*
population is a set of *individuals.*
generations are defined by a sequence of different *populations*
Individuals are evaluated using a *fitness function* (to be optimized) that is their adaptation to the environment

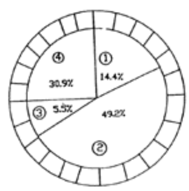
Genetic Algorithms

Procedure **GA**
begin
    t ← 0
    initialize population P(t) with m individuals
    evaluate population P(t)
    while termination condition is not met
      begin
          select parents from P(t)
          generates new individuals using reproduction rules
          some individuals die in P(t)
          form a new population P(t+1)
          evaluate population P(t+1)
          t ← t + 1
      end
return the best individual
end

Reproduction

Reproduction: it takes inspiration form Darwin natural selection process
Individuals with higher fitness have higher probability to reproduce.
String x is the binary code of a number. Fitness(x) = x^2

No.	String	Fitness	% of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0



Genetic Algorithms

From 2 parents 2 children are generated following a crossover operator

Afather = *0 1 1* | *1 0 0 0*
Amother = 0 0 1 | 0 1 1 0

Achild1 = *0 1 1* | 0 1 1 0
Achild2 = 0 0 1 | *1 0 0 0*

To each child a random mutation process is applied to modify some of the gene components

A1 = 0 1 *1* 0 1 1 0
A1 = 0 1 *0* 0 1 1 0

187

TABLE 1.2 A Genetic Algorithm by Hand

String No.	Initial Population (Randomly Generated)	x Value (Unsigned Integer)	$f(x)$ x^2	pselect, $\frac{f_i}{\Sigma f}$	Expected count $\frac{f_i}{\bar{f}}$	Actual Count from Roulette Wheel
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4.0
Average			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

Mating Pool after Reproduction (Cross Site Shown)	Mate (Randomly Selected)	Crossover Site (Randomly Selected)	New Population	x Value	$f(x)$ x^2
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 1 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256
					1754
					439
					729

188

GA Parameters

Number of individuals: Usually is a compromise between search space coverage and the need to escape from local minimum. A good starting number is 100.

Crossover Probability (defined for each couple of individuals): define the crossover probability. Parents have also the possibility to reproduce without combining their chromosome. This parameter is crucial to guarantee the search space coverage and that new individuals are generated. A good value is 0.5

Mutation Probability (defined for each gene)*:* Usually *0.005*

189

GA Parameters

Generation Gap: percentage of individuals replaced between one generation and the next. In case is 100% all the children substitute the parents. In case is lower (e.g. 80%) we keep the best 20% of the parents and the best 80% of the children. Value close to 100% are usually used.

Selection strategy: In case of *elitist* strategy (with parameter *n*) the best *n* individuals are moved automatically to the next generation. This prevent that the best individual are not reproduced in the next population. Usually *n*=0 or *n*=1.

190

Genetic Algorithms

We define these functions
Random: generates a random number between 0 and 1
Flip: return a true Boolean value according to a probability
Rnd: return an integer randomly chosen between two parameters lower and upper)
Fitness[j] is the fitness of individual j
v is the chromosome length

Selection (sum_fitness**)**
sum_tmp ← 0, j ← 0
rand ← random*sum_fitness
repeat
    j ← j +1
    sum_tmp ← sum_tmp + fitness[j]
until (sum_tmp >= rand)
return (selected_individual ← j)

191

Genetic Algorithms

Crossover (parent1, parent2)
if flip(*pcross*) **then**
    pos_cross ← rnd(1, v-1)
    for j←1 **to** pos_cross
        child1[j] ← mutation(parent1[j])
        child2[j] ← mutation(parent2[j])
    if pos_cross <> v **then**
        for j ← pos_cross+1 **to** v
            child1[j] ← mutation(parent2[j])
            child2[j] ← mutation(parent1[j])

Mutation (gene)
if flip(*pmutaztone*) **then**
    return (gene mutation)
    else
        return (gene)

192

Genetic Algorithms

Scale the fitness: in same case is needed to avoid that fitness values are too close. Fitness values are rescaled starting from the max and the min value.

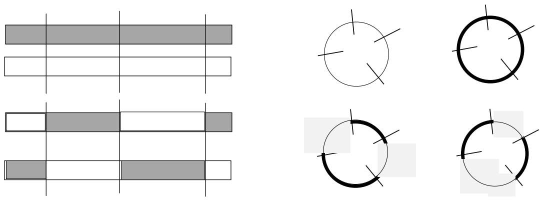
Ranking: an alternative way to define the selection probability. First individuals are sorted and next the probability is given by the position in the sort. This avoid to always select individuals with very high fitness.

Chromosome as sequence of parameters:
A = par1 par2 par3 ... parn
A = 000 111 010 001

193

Genetic Algorithms

Multiple Crossover



194

Genetic Algorithms for TSP

An individual is a tour. Normal crossover does not work

P1 = [2 1 3|4 5 6 7] → O1 = [2 1 3 2 5 7 6]
P2 = [4 3 1|2 5 7 6] → O2 = [4 3 1 4 5 6 7]

City 2 and 4 are visited twice in the two solutions.

A possibility is to maintain absolute position for the first part of the individual and relative position for the second part

P1 = [2 1 3 4 5 6 7] → O1 = [2 1 4 3 5 7 6]
P2 = [4 3 1 2 5 7 6] → O2 = [4 3 2 1 5 6 7]

195

Genetic Algorithms for TSP

Greedy Crossover by J. Grefenstette

Gene presentation, a sequential representation where the cities are listed in the order in which they are visited. Example: [9 3 4 0 1 2 5 7 6 8]

Greedy crossover selects the first city of one parent, compares the cities leaving that city in both parents, and chooses the closer one to extend the tour.
If one city has already appeared in the tour, we choose the other city.
If both cities have already appeared, we randomly select a non-selected city

196

Genetic Algorithms

Infeasibility

Recombining individuals, the offspring might be potentially infeasible.
Three basic strategies:
- reject (simplest)
- penalizing infeasible individuals in the quality function
- Repair

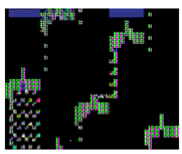
Intensification Strategy

- Application of LS to improve the fitness of individuals. Approaches with LS applied to every individual of a population are often called Memetic Algorithms
- Linkage learning or building block learning: a strategy that uses recombination operators to explicitly try to combine "good" parts of individuals (rather than, e.g., a simple one-point crossover for bitstrings)

197

ACO, Ant Colony Optimization

- 10^{18} living insects (rough estimate)
- ~2% of all insects are social
- Social insects are:
  - All ants
  - All termites
  - Some bees
  - Some wasps



- 50% of all social insects are ants
- Avg weight of one ant between 1 and 5 mg
- Tot weight ants ~ Tot weight humans

198

How Do Ants Coordinate their Activities?

- Ants do not directly communicate. The basic principle is stigmergy, a particular kind of indirect communication based on environmental modification
- Stimulation of workers by the performance they have achieved Grassé P. P., 1959
- Foraging behavior: searching for food by parallel exploration of the environment

199

Shortest paths: an emerging behavior from stigmergy

- **Foraging ant colonies can synergistically find *shortest paths* in *distributed / dynamic* environments:**
  - While moving back and forth between nest and food ants mark their path by *pheromone* laying
  - Step-by-step routing decisions are biased by the *local* intensity of pheromone field (*stigmergy*)
  - Pheromone is the colony's collective and distributed *memory*: it encodes the collectively learned quality of local routing choices toward destination target

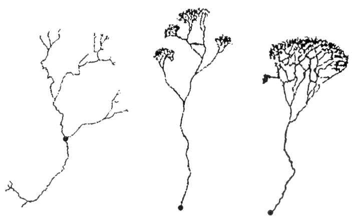
R. Beckers, J. L. Deneubourg and S. Goss, Trails and U-turns in the selection of the shortest path by the ant Lasius Niger, *J. of Theoretical Biology*, 159, 1992

200

How Ants Find Food

201

Ants Foraging Behavior



202

Pheromone Trail Following

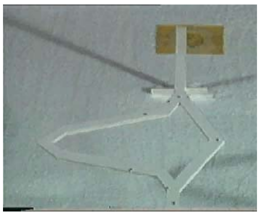
Ants and termites follow pheromone trails

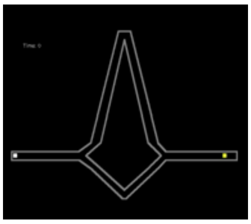
203

Asymmetric Bridge Experiment

Goss et al., 1989

Dorigo & Bertolissi, 1998

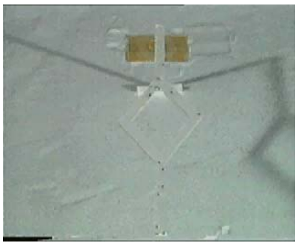


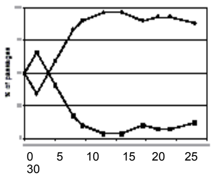


204

Simple Bridge Experiment

Goss et al., 1989, Deneubourg et al., 1990





205

Ant Colony Optimization

- *ACO algorithms* are multi-agent systems that exploit *artificial stigmergy* for the solution of combinatorial optimization problems.
- Artificial ants live in a discrete world. They construct solutions making stochastic transition from state to state.
- They deposit artificial pheromone to modify some aspects of their environment (search space). Pheromone is used to dynamically store past history of the colony.
- Artificial Ants are sometime "augmented" with extra capabilities like local optimization or backtracking

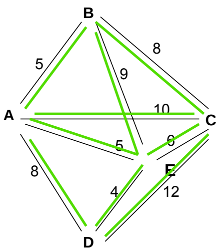
206

Search Space

Discrete Graph

To each edge is associated a *static value* returned by an heuristic function *h*(*r*,*s*) based on the edge-cost

Each edge of the graph is augmented with a pheromone trail *t*(*r*,*s*) deposited by ants. *Pheromone is dynamic* and it is learned at run-time



207

ACO: Ant Colony Optimization

Basic ACO Algorithm

InitializePheromone( )
while termination conditions not met **do**
  ScheduleActivities
    SimulateAntSolutionConstruction( )
    PheromoneUpdate( )
    DeamonActions( )
  end ScheduleActivities
endwhile

No fixed constraint about the order of the phases of the algorithm

Ants build the solution with a random walk

Pheromone trails are updated according to the solutions' quality

Optional global actions (e.g., offline pheromone updates, local search steps)

208

Ant Colony Optimization

- There are many variants of Ant Colony Optimization algorithms.
- They vary in the way solutions are constructed by artificial ants and in the way the pheromone is updated
- Main variants are
  - Ant Colony System (ACS), Gambardella, Dorigo, 1996
  - Ant System (AS), Dorigo, 1991
  - Max Min Ant System (MMAS),Stützle and Hoos (2000)

209

ACS: Ant Colony System for TSP

Loop
  Randomly position m artificial ants on n cities
  For city=1 to n
    For ant=1 to m
      {*Each ant builds a solution by adding one city after the other*}
      Select probabilistically the next city according to exploration and exploitation mechanism
      Apply the local trail updating rule
    End for
    calculate the length L_m of the tour generated by ant m
  End for
  Apply the global trail updating rule using the best ant so far
Until End_condition

Gambardella L.M, Dorigo M., 1996

210

ACS state transition rule: formulae

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\} & \text{if } q \le q_0 \quad \text{(Exploitation)} \\ S & \text{otherwise} \quad \text{(Exploration)} \end{cases}$$

where

- S is a stochastic variable distributed as follows:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases}$$

- t is the trail
- h is the inverse of the distance
- $J_k(r)$ is the set of cities still to be visited by ant k positioned on city r
- β and q_0 are parameters

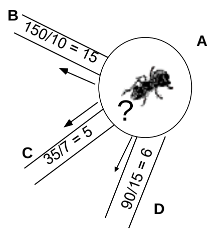
ACS state transition rule: example

t(A,B) = 150 h (A,B) = 1/10
t(A,C) = 35 h (A,C) = 1/7
t(A,D) = 90 h (A,D) = 1/15

with probability q_0 exploitation (Edge AB = 15)

with probability $(1-q_0)$ exploration

AC with probability 5/11
AD with probability 6/11



ACS local trail updating
... similar to evaporation

If an edge (r,s) is visited by an ant

$$\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s)$$

with $\Delta\tau(r,s) = \tau_0$
That is the initial value of the pheromone equal for all edges

$$\tau_0 \leftarrow 1 / nnei * ncities$$

Where nnei is the length of a tour computed
with a nearest neighbor heuristic

ACS's global trail updating

At the end of each iteration the best ant is allowed
to reinforce its tour by depositing additional pheromone
proportional to the length of the tour

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s)_{Global}$$

where

$$\Delta\tau(r,s)_{Global} = \frac{1}{L_{best}}$$

Ant System: construction phase

Only exploitation in the construction phase

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases}$$

- t is the trail
- h is the inverse of the distance
- $J_k(r)$ is the set of cities still to be visited by ant k positioned on city r
- β is a parameter

Ant System: pheromone updating

At the end of the constructive phase all ants are involved in updating the pheromone (In ACS only the best ant)
Pheromone also evaporated on all edges (in ACS evaporation is done only for visited edge during the construction phase)

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \,,$$

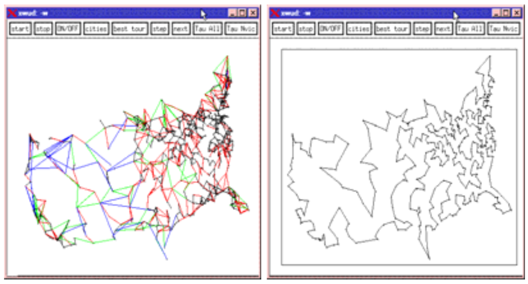
$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise,} \end{cases}$$

Max Min Ant System (MMAS)

MMAS differs from AS in that
(i) As in ACS only the best ant adds pheromone trails,
(ii) No local pheromone updating
(iii) the minimum and maximum values of the pheromone are explicitly limited
(in AS and ACS these values are limited implicitly, that is, the value of the limits is a result of the algorithm working rather than a value set explicitly by the algorithm designer).

217

TSP problem



218

ACS comparison with other heuristics on random TSPs

Problem name	ACS (average)	SA (average)	EN (average)	SOM (average)
City set 1	**5.88**	**5.88**	5.98	6.06
City set 2	6.05	**6.01**	6.03	6.25
City set 3	**5.58**	5.65	5.70	5.83
City set 4	**5.74**	5.81	5.86	5.87
City set 5	**6.18**	6.33	6.49	6.70

Comparisons on average (25 trials) tour length obtained on five random 50-city symmetric TSP

219

Comparison of ACS with other natural algorithms on geometric TSPs

Problem name	ACS	GA	EP	SA	Optimum
Eil50 (50-city problem)	**425** **(427.96)** **[1,830]**	428 (N/A) [25,000]	426 **(427.86)** [100,000]	443 (N/A) [68,512]	**425** (N/A)
Eil75 (75-city problem)	**535** **(542.37)** **[3,480]**	545 (N/A) [80,000]	542 (549.18) [325,000]	580 (N/A) [173,250]	**535** (N/A)
KroA100 (100-city problem)	**21,282** **(21,285.44)** **[4,820]**	21,761 (N/A) [103,000]	N/A (N/A) [N/A]	N/A (N/A) [N/A]	**21,282** (N/A)

Best integer tour length, best real tour length (in parentheses) and number of tours required to find the best integer tour length (in square brackets)
Optimal length is available only for integer tour lengths ACS results on 25 trials

220

ACS on some geometric TSP problems in TSPLIB

Problem name	ACS best integer length (1)	ACS number of tours generated to best	ACS average integer length	Standard deviation	Optimum (2)	Relative error $\frac{(1)-(2)}{(2)} * 100$	CPU sec to generate a tour
d198 (198-city problem)	15,888	585,000	16,054	71	15,780	0.68 %	0.02
pcb442 (442-city problem)	51,268	595,000	51,690	188	50,779	0.96 %	0.05
att532 (532-city problem)	28,147	830,658	28,523	275	27,686	1.67 %	0.07
rat783 (783-city problem)	9,015	991,276	9,066	28	8,806	2.37 %	0.13
fl1577 (1577-city problem)	22,977	942,000	23,163	116	[22,204 – 22,249]	3.27÷3.48 %	0.48

Integer length of the shortest tour found, number of tours to find it, avg integer length (over 15 trials), its std dev, optimal solution, and the relative error of ACS

221

Hybrid ACS: ACS plus local search

Loop
  Randomly position m agents on n cities
  For step=1 to n
    For ant=1 to m
      Apply the state transition rule
      Apply the local trail updating rule
  Apply local search
  Apply the global trail updating rule
Until End_condition

222

ACS-3-opt applied to TSP

Problem name	ACS-3-opt best result (length)	ACS-3-opt best result (sec)	ACS-3-opt average (length) (1)	ACS-3-opt average (sec)	Optimum (2)	% Error (1)-(2) ------ (2)
d198 (198-city problem)	15,780	16	15,781.7	238	15,780	0.01 %
lin318* (318-city problem)	42,029	101	42,029	537	42,029	0.00 %
att532 (532-city problem)	27,693	133	27,718.2	810	27,686	0.11 %
rat783 (783-city problem)	8,818	1,317	8,837.9	1,280	8,806	0.36 %

Results obtained by ACS-3-opt on TSP problems taken from the First International Contest on Evolutionary Optimization, IEEE-EC 96, May 20-22, 1996, Nagoya, Japan

Comparison of ACS-3-opt and GA+local search on TSPs

Problem name	ACS-3-opt average (length) (1)	ACS-3-opt average (sec)	ACS-3-opt % error (1)-(3) ------ (3)	STSP-GA average (length) (2)	STSP-GA average (sec)	STSP-GA % error (2)-(3) ------ (3)	Optimum (3)
d198 (198-city problem)	15,781.7	238	0.01 %	15,780	253	0.00 %	15,780
lin318 (318-city problem)	42,029	537	0.00 %	42,029	2,054	0.00 %	42,029
att532 (532-city problem)	27,718.2	810	0.11 %	27,693.7	11,780	0.03 %	27,686
rat783 (783-city problem)	8,837.9	1,280	0.36 %	8,807.3	21,210	0.01 %	8,806

Results obtained by ACS-3-opt and by STSP-GA on ATSP problems taken from the First International Contest on Evolutionary Optimization, IEEE-EC 96, May 20-22, 1996, Nagoya, Japan

ACS-3-opt applied to ATSP

Problem name	ACS-3-opt best result (length)	ACS-3-opt best result (sec)	ACS-3-opt average (length) (1)	ACS-3-opt average (sec)	Optimum (2)	% Error (1)-(2) ------ (2)
p43 (43-city problem)	2,810	1	2,810	2	2,810	0.00 %
ry48p (48-city problem)	14,422	2	14,422	19	14,422	0.00 %
ft70 (70-city problem)	38,673	3	38,679.8	6	38,673	0.02 %
kro124p (100-city problem)	36,230	3	36,230	25	36,230	0.00 %
ftv170* (170-city problem)	2,755	17	2,755	68	2,755	0.00 %

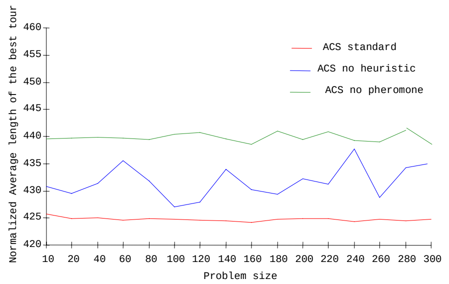
Results obtained by ACS-3-opt on ATSP problems taken from the First International Contest on Evolutionary Optimization, IEEE-EC 96, May 20-22, 1996, Nagoya, Japan

Comparison of ACS-3-opt and GA GA+local search on ATSPs

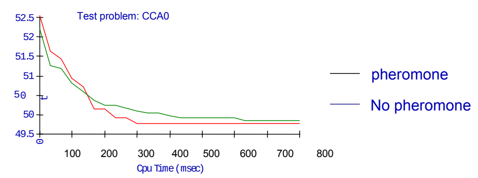
Problem name	ACS-3-opt average (length) (1)	ACS-3-opt average (sec)	ACS-3-opt % error (1)-(3) ------ (3)	ATSP-GA average (length) (2)	ATSP-GA average (sec)	ATSP-GA % error (2)-(3) ------ (3)
p43 (43-city problem)	2,810	2	0.00 %	2,810	10	0.00 %
ry48p (48-city problem)	14,422	19	0.00 %	14,440	30	0.12 %
ft70 (70-city problem)	38,679.8	6	0.02 %	38,683.8	639	0.03 %
kro124p (100-city problem)	36,230	25	0.00 %	36,235.3	115	0.01 %
ftv170 (170-city problem)	2,755	68	0.00 %	2,766.1	211	0.40 %

Results obtained by ACS-3-opt and by ATSP-GA on ATSP problems taken from the First International Contest on Evolutionary Optimization, IEEE-EC 96, May 20-22, 1996, Nagoya, Japan

Pheromone trail and heuristic function: are they useful?

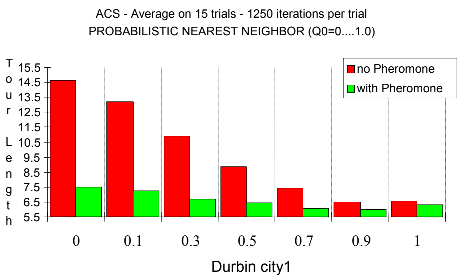


Effectiveness of distributed pheromone learning



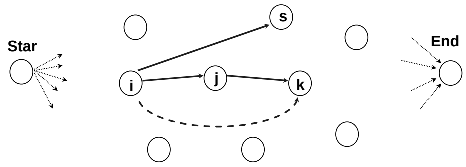
Best tour length as a function of elapsed CPU time (avg on 100 runs)

ACS vs Probabilistic Nearest Neighborhood



Sequential Ordering Problem

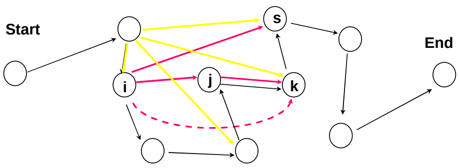
It consists of finding a minimum weight Hamiltonian path on a directed graph subject to multiple precedence constraints among nodes.



SOP models real-world problems like production planning, single-vehicle pick-up and delivery and transportation problems

Sequential Ordering Problem

Find the minimal sequence from node **Start** to node **End** that visits all the nodes and do not violate precedence constraints (in red)



Sequential Ordering Problem

- Escudero (1988)
- *General ATSP Problem*
  - Precedence Constrained ATSP Polytope (Balas, Fischetti, Pulleyblank, 1995).
  - Branch and Cut (Ascheuer, 1996)
  - Maximum Partial Order/Arbitrary Insertion GA (Chen and Smith, 1996)
  - HAS-SOP: ACO based algorithm (Gambardella L.M, Dorigo M., 2000)
- *Pick-Up and Delivery*
  - Lexicographic search with labeling Procedure (Savelsbergh, 1990).

Sequence-based crossover operators

Partially Mapped Crossover (PMX) [Goldberg and R. Lingle., 1985] has the form of two-point crossover.

```
Parent 1:    i b d | e f g | a c h j
Parent 2:    h g a | c b j | i e d f

Offspring:   i b d | c b j | a e h j
               f   |       |   e   g
```

2: Example of PMX. The mappings are c-e, b-f, and j-g.

The offspring takes the cities from Parent 2 between the cut-points, and it takes the cities in the first and last sections from Parent 1. However, if a city in these outer sections has already been taken from Parent 2, its "mapped" city is taken instead. The mappings are defined between the cut-points--the city of Parent 2 is mapped to the corresponding city of Parent 1.

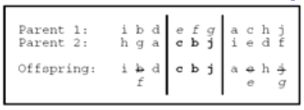
Sequence-based crossover operators

Order Crossover (OX) [Davis 85] has the form of two-point crossover.

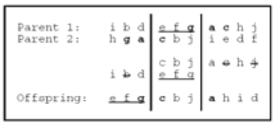
```
Parent 1:    i b d | e f g | a c h j
Parent 2:    h g a | c b j | i e d f

                   | c b j | a e h j
             i b d | e f g |

Offspring:   e f g | c b j | a h i d
```

The offspring starts by taking the cities of Parent 2 between the cut-points. Then, starting from the second cut-point, the offspring takes the cities of Parent 1 ("wrapping around" from the last segment to the first segment). When a city that has been taken from Parent 2 is encountered, it is skipped--the remaining cities are appended in the order they have in Parent 1.

Sequence-based crossover operators



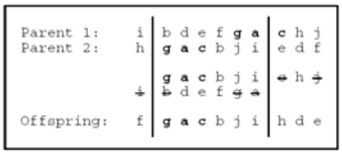
2: Example of PMX. The mappings are c-e, b-f, and j-g.



Comparison between PMX and OX. OX can be less disruptive to sub-tours. For example, the sub-tour e-f-g in Parent 1 is now transmitted to the offspring. However, the common sub-tour g-a-c is (still) disrupted.

Sequence-based crossover operators

Maximal Sub-Tour (MST)--the longest (undirected) sub-tour that is common to both parents. Thus, OX is modified to preserve the MST.

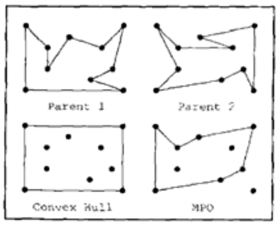


7: Example of MST-OX. The Maximal Sub-Tour g-a-c is now preserved.

Scanning both parents to identify the Maximal Sub-Tour, the first cut-point occurs to the immediate left of the MST in Parent 2. The second cut-point is then made a random distance to the right of the MST1. After OX is applied

Sequence-based crossover operators

The longest common partial order is the Maximum Partial Order (MPO). Using Arbitrary Insertion to complete this partial solution, the overall process defines the Maximum Partial Order/Arbitrary Insertion heuristic operator.



Maximum Partial Order (MPO).

Sequence-based crossover operators

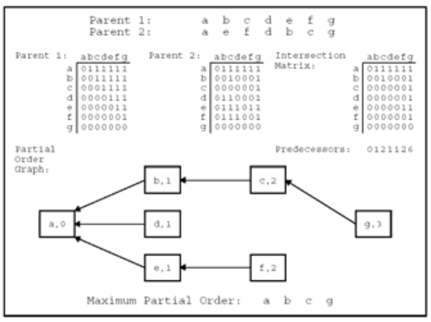
Maximum Partial Order (MPO).

- Convert parent sequences to Boolean matrices
- Intersect matrices
- Sum columns to get each city's predecessors
- Build partial order graph
  - Find city with fewest number of predecessors
  - Attach city to the predecessor with the most *ordered* predecessors
- Find longest path in graph

3: Pseudo-code to find the Maximum Partial Order of two parent sequences.

Sequence-based crossover operators

Maximum Partial Order (MPO).



City g is preceded by all of the cities in the graph, but it can only be attached to cities c and f because those cities have the most ordered predecessors (2).

MPO/AI performance

Table 6.3: Results for MPO/AI in GENIE. Population size is 400, and the initial population is 400 AI solutions started from the convex hull. Values are percent surplus from known optimum for average of 5 runs (50 generations each).

TSPLIB Instance	Avg. Best Convex Hull/AI Start Tour	Avg. Best MPO/AI Tour
d198	+ 3.05 %	+ 1.24 %
lin318	+ 6.04 %	+ 1.75 %
fl417	+ 1.91 %	+ 0.58 %
pcb442	+ 8.97 %	+ 3.48 %
u574	+ 8.45 %	+ 2.59 %
average	+ 5.68 %	+ 1.93 %

Is the Common Good? A New Perspective Developed in Genetic Algorithms, PHD Thesis, Stephen Chen, 1999

MPO/AI for SOP

SOP Instance	Size	Con-straints	Average Best AI Start	Average Best MPO/AI	Overall Best MPO/AI	Original Branch and Cut Bounds [Asc96]	Time to Bound (s)	Runtime (s)
ft70.1	71	17	41809	39615	39545	**39313**	NA	76
ft70.2	71	35	43485	40435	**40422**	[39739, 41778]	5.4	73
ft70.3	71	68	46731	42558	* **42535**	[41305, 44732]	2.8	48
ft70.4	71	86	55982	53583	**53562**	[52269, 53882]	2.4	47
kro124p.1	101	25	45758	40996	**40186**	[37722, 42845]	5.4	136
kro124p.2	101	49	49056	42576	**41677**	[38534, 45848]	5.2	94
kro124p.3	101	97	63768	51085	**50876**	[40967, 55649]	3.8	103
kro124p.4	101	131	87975	76103	* **76103**	[64858, 80753]	2.0	76
rbg323a	325	2412	3466	3161	**3157**	[3136, 3221]	207.6	1566
rbg341a	343	2542	3184	2603	**2597**	[2543, 2854]	70.6	2205
rbg358a	360	3239	3165	2636	**2599**	[2518, 2758]	533.8	4491
rbg378a	380	3069	3420	2843	**2833**	[2761, 3142]	67.6	5354
ry48p.1	49	11	16602	15813	* **15805**	[15220, 15935]	2.4	22
ry48p.2	49	23	18071	16676	* **16666**	[15524, 17071]	1.6	32
ry48p.3	49	42	22074	19905	* **19894**	[18156, 20051]	7.6	29
ry48p.4	49	58	32591	31446	31446	[29967, 31446]	5.0	19

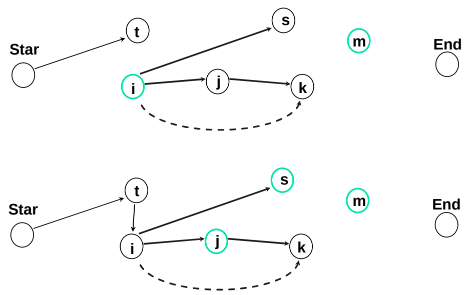
HAS-SOP: Hybrid Ant System for SOP

- Dorigo, Gambardella 2000
- Constructive phase based on ACS
- Trail updating as ACS
- New local search SOP-3_exchange strategy based on a combination between lexicographic search and a new labeling procedure.
- New data structure to drive the search
- First in literature that uses a local search edge-exchange strategy to directly handle multiple constraints without any increase in computational time.

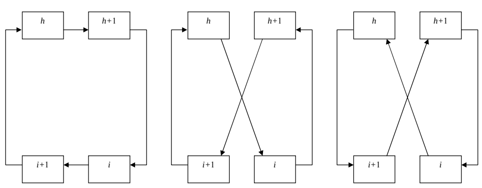
Ants for SOP

- Each ant iteratively starts from node 0 and adds new nodes until all nodes have been visited and node *n* is reached.
- When in node *i*, an ant chooses probabilistically the next node *j* from the set *F*(*i*) of feasible nodes.
- *F*(*i*) contains all the nodes *j* still to be visited and such that all nodes that have to precede *j*, according to precedence constraints, have already been inserted in the sequence

Feasable Ant Sets

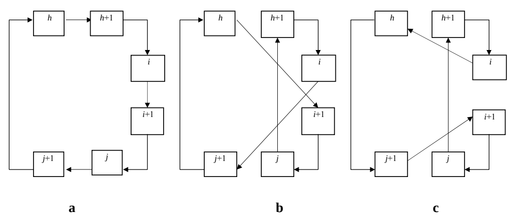


Local Search



A 2-exchange always inverts a path.

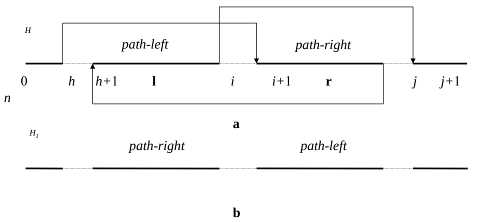
Local Search



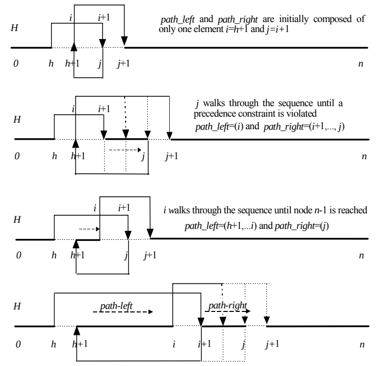
A 3-exchange without (b) and with (c) path inversion

SOP-3-Exchange

It exchanges 2 sequences. The procedure follows a lexicographic search and it is able to check in constant time if a precedence constraint is violated



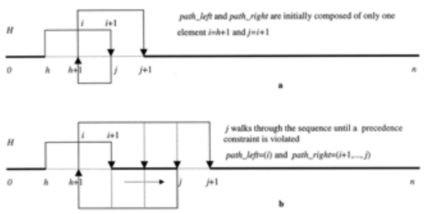
SOP-3-Exchange: lexicographic search



SOP-3-Exchange Labeling procedure

We start by fixing h, $i=h+1$, and *path_left*=(i).
for all nodes s □*successor*[i] we set *f_mark*(s)=*count_h*.

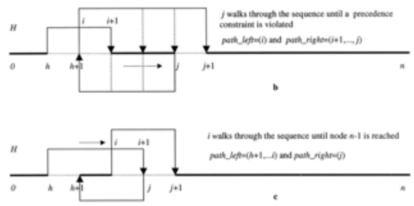
We repeat this operation each time *path_left* is expanded with a new node i. The labeling procedure marks with the value *count_h* all the nodes in the sequence that must follow one of the nodes belonging to *path_left*.



SOP-3-Exchange Labeling procedure

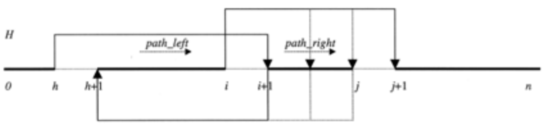
When *path_right* is expanded moving j in ‹i+2, ..., n› if *f_mark*(j) =*count_h* we stop the search because the label indicates that j must follow a node in *path_left*.

At this point, if no other search-termination condition is met, the procedure restarts expanding again *path_left*. In this situation all the previous defined labels remain valid and the search continues by labeling all the successors of the new node i.

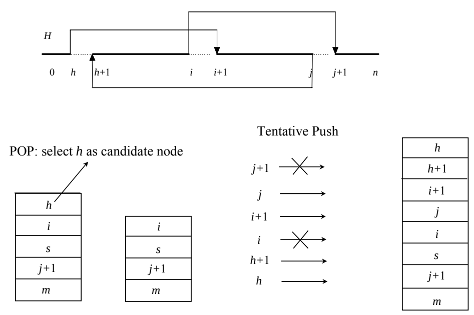


SOP-3-Exchange Labeling procedure

When we move h forward into the sequence we invalidate all previously set labels by setting *count_h*=*count_h*+1.



Selection of node h with the Don't Push Stack

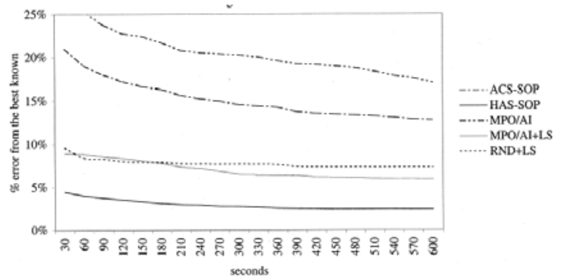


Local search contribution

	RND	MPO/AI	ACS-SOP	RND+LS	MPO/AI+LS	HAS-SOP
prob.100	1440.17%	134.66%	40.62%	50.07%	47.58%	**17.46%**
rbg109a	64.57%	0.33%	1.93%	0.08%	0.06%	**0.00%**
rbg150a	37.85%	0.19%	2.54%	0.08%	0.13%	**0.00%**
rbg174a	40.86%	0.01%	2.16%	0.15%	**0.00%**	0.08%
rbg253a	45.85%	0.03%	2.68%	0.21%	**0.00%**	**0.00%**
rbg323a	80.14%	1.08%	9.60%	1.27%	**0.08%**	0.21%
rbg341a	125.46%	3.02%	12.64%	4.41%	**0.96%**	1.54%
rbg358a	151.92%	7.83%	20.20%	4.98%	2.51%	**1.37%**
rbg378a	131.58%	5.95%	22.02%	4.17%	1.40%	**0.88%**
avg	235.38%	17.01%	12.71%	7.27%	5.86%	**2.39%**

253

Sequential Ordering Problems



254

HAS-SOP

PROB	TSPLIB Bounds	MPO/AI Best	MPO/AI Avg	MPO/AI Time (sec)	HAS-SOP Best	HAS-SOP Avg	HAS-SOP Time (sec)
ft70.1.sop	39313	39545	39615	120	39313	39313.0	29.8
ft70.2.sop	[39739,40422]	40422	40435	120	40419	40433.5	114.1
ft70.3.sop	[41305,42535]	42535	42558	120	42535	42535.0	64.4
ft70.4.sop	[52269,53562]	53562	53583	120	53530	53566.5	38.2
kro124p.1.sop	[37722,40186]	40186	40996	240	39420	39420.0	115.2
kro124p.2.sop	[38534,41677]	41667	42576	240	41336	41336.0	119.3
kro124p.3.sop	[40967,50876]	50876	51085	240	49499	49648.8	262.8
kro124p.4.sop	[64858,76103]	76103	76103	240	76103	76103.0	57.4
rbg323a.sop	[3136,3157]	3157	3161	2760	3141	3146.0	1685.5
rbg341a.sop	[2543,2597]	2597	2603	3840	2580	2591.9	2149.6
rbg358a.sop	[2518,2599]	2599	2636	6120	2555	2561.2	2169.3
rbg378a.sop	[2761,2833]	2833	2843	8820	2817	2834.3	2640.3

We tested and compare our algorithms on a set of problems in TSPLIB using a SUN Ultra SPARC 1 (167Mhz)

255

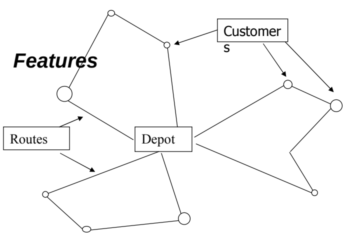
PROB	TSPLIB Bounds	NEW Lower Bounds	NEW Upper Bounds	HAS-SOP All Best	Avg Result	Std.Dev.	Avg Time (sec)
ESC63.sop	62			62	62.0	0	0.1
ESC78.sop	18230			18230	18230.0	0	6.9
ft53.1.sop	[7438,7570]		7531	7531	7531.0	0	9.9
ft53.2.sop	[7630,8335]		8026	8026	8026.0	0	18.4
ft53.3.sop	[9473,10935]		10262	10262	10262.0	0	2.9
ft53.4.sop	14425			14425	14425.0	0	0.4
ft70.1.sop	39313			39313	39313.0	0	29.8
ft70.2.sop	[39739,40422]	39803	40419	40419	40433.5	24.6	114.1
ft70.3.sop	[41305,42535]	41305		42535	42535.0	0	64.4
ft70.4.sop	[52269,53562]	53072	53530	53530	53566.5	7.6	38.2
kro124p.1.sop	[37722,40186]	37761	39420	39420	39420.0	0	115.2
kro124p.2.sop	[38534,41677]	38719	41336	41336	41336.0	0	119.3
kro124p.3.sop	[40967,50876]	41578	49499	49499	49648.8	249.7	262.8
kro124p.4.sop	[64858,76103]			76103	76103.0	0	57.4
prob.100.sop	[1024,1385]	1027	1190	1190	1302.4	39.4	1918.7
rbg109a.sop	1038			1038	1038.0	0	14.6
rbg150a.sop	[1748,1750]			1750	1750.0	0	159.1
rbg174a.sop	2033			2033	2034.7	1.4	99.3
rbg253a.sop	[2928,2987]	2940	2950	2950	2950.0	0	81.5
rbg323a.sop	[3136,3157]	3137	3141	3141	3146.0	1.4	1685.5
rbg341a.sop	[2543,2597]	2543	2574	2574	2591.9	11.8	2149.6
rbg358a.sop	[2518,2599]	2529	2545	2545	2561.2	5.2	2169.3
rbg378a.sop	[2761,2833]	2817	2817	2817	2834.3	10.7	2640.3

Norbert Ascheuer (1997) has run his branch&cut SOP program starting from our best solutions. He could not improve them within 24-CPU hours on a SUN SPARC Station 4 (110Mhz) but he proves optimality for rbg378a and computes the new reported lower bounds.

256

Vehicle Routing Problems (VRPs)

1. VRP is a generic name given to a class of problems in which customers are visited by vehicles (first formulated by Dantzig and Ramser in 1950)
2. The problem is to design routes for the vehicles so as to meet the given constraints and objectives minimising a given objective function.



257

VRP- Characteristics and Components

Freight transportation provided by vehicles through a route network

Main components:
- Road network
- Customers
- Vehicles
- Depots
- Drivers
- Operational constraints:
  - global
  - for single routes
- Optimization objectives

258

VRP- Characteristics and Components

The road network

A graph $G=(V, A)$, $G=(V, E)$ or $G=(V, A \cup E)$
Directed, undirected or mixed
Sparse vs Dense
Sparse ⇒ $|A|=O(|V|)$
Dense ⇒ $|A|=O(|V|^2)$

Directed graph:
small scale road network (cities)

Undirected graph:
large scale road network (countries, regions)

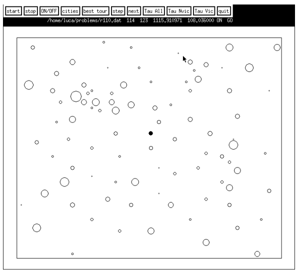
VRP- Characteristics and Components

The road network

Vertices
- depots, customers, road intersections
- $V=\{0, 1, \ldots, n\}$

Arcs
- roads
- directed $(i, j) \in A$ or undirected $e \in E$
- length, or travel cost $c_{ij} \forall (i, j) \in A$
- travel time $t_{ij} \forall (i, j) \in A$

VRP- Characteristics and Components



Customers

Associated to vertex or arcs
Requested quantity

Characteristics

- Service time (load/unload)
- Delivery Time Windows
- Pick-up and delivery
- Access Limitation
- Allow to split delivery

VRP- Characteristics and Components

Fleet/Vehicles

- Fleet size (fixed or variable)
- Company or outsourced fleet (fixed vehicle cost)
- Depot of reference (single, multiple, possibility of change)
- Vehicle capacity (maximum load allowed; weight, volume)
- Freight compatibility (perishable goods, dangerous materials)
- Compatibility with streets
- Load/unload procedure
- Costs (associated with mileage, time, fuel, journey, load)

VRP- Characteristics and Components

Drivers
- Employee workers or vehicle owners
- Union and contract conditions
- Working periods, shifts and breaks
- Availability and possibility of overtime

Depots
- Single or multiple
- Number and type of available vehicles
- Set of a priori assigned customers
  ⇒ decomposable problems

VRP- Characteristics and Components

Operational constraints

Relevant to:
the nature of transport
the quality of service
the driver working contract

Two classes of constraints:
Local constraints (single route)
Global constraints (the whole set of routes)

VRP- Characteristics and Components

Operational constraints

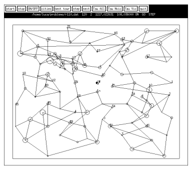
Local constraints (single route)
- vehicle capacity
- maximum allowed route distance/duration
- time constraints (arrival, departure, time windows)
- kind of service (pickup, delivery or both)
- precedence among customers:
  - pickup and delivery
  - linehaul/backhaul

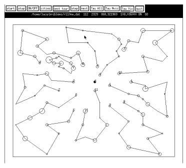
Global constraints (whole set of routes)
- maximum number of vehicles
- maximum number of routes (for vehicle or depot)
- workload balancing
- working periods and shifts (minimum time between routes)

VRP- Characteristics and Components

Objectives: (multiples)
- Minimize: the global transportation cost + drivers and vehicles fixed costs
- Minimize: the number of vehicles and/or drivers
- Balancing of the routes
- Minimize: penalties for not/partially served customers

  ⇒ Conflicting objectives



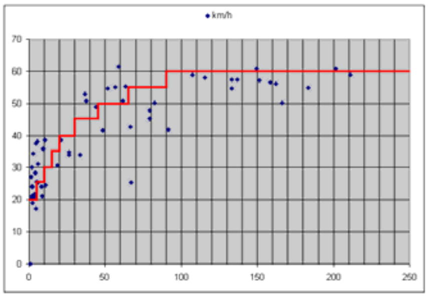


VRP- Characteristics and Components

Other characteristics

- Service split on several days
- More routes for vehicles in a day
- More requests for a customer
- Demand partially or not a priori known (dynamic, on-line problems)
- Stochastic and/or time dependent arc costs/travel times

Speed Model



Speed - Distances

0 – 5	km	=>	20 km/h
5 – 10	km	=>	25 km/h
10 – 15	km	=>	30 km/h
15 – 20	km	=>	35 km/h
20 – 30	km	=>	40 km/h
30 – 45	km	=>	45 km/h
45 – 65	km	=>	50 km/h
65 – 90	km	=>	55 km/h
90 ...	km	=>	60 km/h

The General Vehicle Routing Problems

Problem formulation

Given a generic graph G=(V, A∪ E) determine a minimum cost set of M cycles (routes) that serve the required vertices U⊆V and the required edges R⊆A∪ E satisfying a set of operational constraints

Cost of a route = the sum of the cost of the edges belonging to the route

Applications:
- collection and delivery of goods
- waste collection
- street cleaning
- school-bus routing
- dial-a-ride systems
- transportation of people with handicap
- routing of salespeople

The General Vehicle Routing Problems

Two main classes of problems

Node Routing Problem (NRP)

Customers/demand concentrated in sites associated with vertices
- ☐ only required vertices U, R=∅
- ☐ frequently denoted as VRP or Vehicle Scheduling Problem

Arc Routing Problem (ARP)

Customers/demand evenly distributed along the edges
- ☐ only required edges R, U=∅

The problems without operational constraints:
NRP reduces to Traveling Salesman Problem (TSP)
ARP reduces to Rural Chinese Postman Problem (RCPP)
ARP with R=A reduces to Chinese Postman Problem (CPP)

VRP-Node Routing Problems

It concerns the distribution/collection of goods

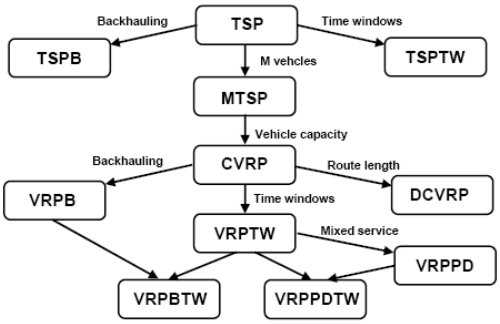
Main components:
- Vehicles
- Depots
- Drivers
- Road Network

Solution:

A set of routes performed by a fleet of vehicles such that:
- each route starts and ends at vehicles' depots
- the customers' requirements are satisfied
- the operational constraints are fulfilled
- the global transportation cost is minimized

271

- Relationships among the basic VRP (NRP) problems



272

VRP-Node Routing Problems

- Traveling Salesman Problem (TSP)
- Traveling Salesman Problem with Backhauls (TSPB)
- Traveling Salesman Problem with Time Windows (TSPTW)
- Multiple Traveling Salesman Problem (MTSP)
- Capacitated Vehicle Routing Problem (CVRP)
- Distance Constrained Vehicle Routing Problem (DCVRP)
- Vehicle Routing Problem with Backhauls (VRPB)
- Vehicle Routing Problem with Time Windows (VRPTW)
- Vehicle Routing Problem with Pickup and Delivery (VRPPD)

273

VRP-Node Routing Problems

Road graph

G=(V,A) (strongly) connected → G'=(V, A') complete

V ={0,1,...,n} , |V| = n+1, set of vertices
0 the depot
1,..., n customers' locations (cities)

∀(i, j)∈A' cij≥0 is a positive minimum cost (distance) of traveling from city i to city j

274

TSP-Traveling Salesman Problem

TSP (node routing problems without operational constraints). A traveling salesman must visit his customers located in different cities and come back home

- Single vehicle, Road graph G=(V, A) → G'=(V, A') complete
- V = customers' cities and TS home (depot); |V|=n
- ∀(i, j)∈A' cij≥0 the minimum cost (distance) of traveling from city i to city j
- Solution: a minimum cost route which starts and ends at depot and reaches each customer
- Complexity: Strongly NP-hard optimization problem

275

MTSP- Multiple Traveling Salesman

MTSP (multiple traveling salesmen)

M vehicles (no size limitation)

A single common depot

Each vehicle (salesman) must visit at least one customer

Solution: M minimum cost routes (tours) which start and end at depot so that each customer is visited exactly once

276

CVRP-Capacitated VRP

- K identical vehicles
- C vehicle capacity
- A single common depot

∀i∈V\{0} customer a demand di≥0 is defined (d0=0) such that di≤C
- ∀S⊆V define d(S)=Σdi,i∈S
- Each vehicle performs at most one route
- K≥Kmin where Kmin is the minimum number of vehicles to serve all the customers
- Kmin may be determined solving a Bin Packing Problem (BPP) (NPhard problem but with fast approximation algorithms)

277

CVRP-Capacitated VRP

- ∀S⊆V\{0} define r(S) the minimum number of vehicles to serve the customers in S
- trivial bound $r(S) = \left\lceil \frac{d(S)}{C} \right\rceil$
- Solution: a set of exactly K routes (circuits) with minimum cost such that:
  - (a) each circuit visits the depot
  - (b) each customer is visited by exactly a single route
  - (c) the sum of the customer demands visited by a route does not exceed C

278

CVRP-Capacitated VRP

- Simple variants:
  - If K>Kmin some vehicle may not be used
    - find at most K routes
    - fixed costs for using vehicles
    - find the minimum number of routes
  - Different vehicle capacities Ck k=1,...,K
- Complexity
  - the CVRP is a Strongly NP-hard optimization problem
  - it generalizes the TSP

279

CVRP-Capacitated VRP

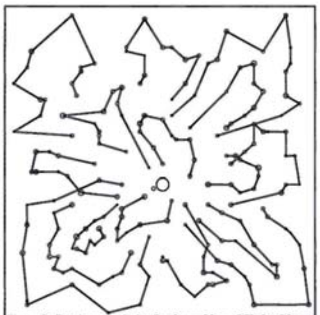


Figure 2 : Best known solution for the problem of Christofides et al. (1979) with 199 customers.

280

DCVRP-Distance Constrained VRP

- A variant of CVRP:
- the capacity constraints is replaced by a maximum route length (time) constraint
- ∀(i, j)∈A′ tij≥0 the length (time) to travel from i to j
- T = maximum route length (time)
- Tk k=1,..,K if the vehicles are different
- ∀i∈V customer, a service time si may be defined explicitly or added to the travel times (t′ij= tij + si/2 + sj/2)
- The cost usually coincides with length (time)
- Solution: the minimum total length (time) solution as for CVRP
- DC-CVRP: both distance and capacity of vehicles are constrained

281

VRPTW-VRP with Time Windows

- variant of CVRP:
- ∀i∈V customer, a time window (TW) is defined as the time interval [ai, bi]
- ∀i∈V\{0} customer, a service time si is given
- ∀(i, j)∈A′ tij≥0 a travel time is given
- the service for each customer must start within his TW
- in case of early arrival the vehicle must wait time instant ai before starting the service
- the routes starts at time 0
- Travel times usually coincide with costs
- TWs induce an implicit orientation (an asymmetric model can be used)

282

VRPTW-VRP with Time Windows

- Solution: a set of exactly K routes (circuits) with minimum cost
- such that:
  - (a) each circuit visits the depot
  - (b) each customer is visited by exactly a single route
  - (c) the sum of the customer demands visited by a route does not exceed C
  - (e) for each customer the service starts within the TW [ai, bi] and the vehicle stops si time instants

VRPTW-VRP with Time Windows

Variants

With soft time windows the violation is a cost in the objective function

Goals (2): first minimize the number of vehicles and second the total distance. Initial solution NP-Hard.

Complexity: Strongly NP-hard generalizes CVRP (ai=0 bi=∞)

TSPTW is the special case for K=1 and C≥d(V)

VRPB - VRP with Backhauls

- An extension of CVRP:
- the set of customers is partitioned into Linehaul Customers (LC) and Backhaul Customers (BC)
- V=L∪B |L|=n |B|=m
- LC require a quantity of goods to be delivered
- BC require a quantity of goods to be picked up
- Precedence constraint among the LC and BC served by the same route:
  - all LC must be served before any BC

VRPPD - VRP with Pickup and Delivery

- An extension of CVRP:
- each customer is associated with two quantities:
  - di demand of commodities to be delivered
  - pi demand of commodities to be picked up
- the commodities (goods) are assumed homogeneous (sometimes only di= di - pi is specified)
- for each customer is defined:
- Oi vertices that are origin of the delivery demand
- Di vertices that are destination of the picked up demand
- at each customer location the delivery is performed before the pickup

VRPPD - VRP with Pickup and Delivery

- Solution: a set of exactly K routes (circuits) with minimum cost
- such that:
- (a) each circuit visits the depot
- (b) each customer is visited by exactly a single route
- (c) the current load of a vehicle along the circuit is non negative never exceed the vehicle capacity C
- (d) ∀ customer i, the customer in Oi (different from the depot) are served in the same circuit before Di
- (e) ∀ customer i, the customer in Di (different from the depot) are served in the same circuit after Oi

VRPPD - VRP with Pickup and Delivery

- If the origin and destination of demands are common (e.g., the depot) they may be not explicitly considered ⇒ VRP with simultaneous P & D (VRPSPD)
- Complexity: Strongly NP-hard generalizes CVRP (Oi= Di={0}, pi=0 ∀i)
- TSPPD specializes the VRPPD for K=1

VRP approaches

Exact Approach (up to 100 nodes)
- Branch and bound (Fisher 1994)

Approximation
- Clark and Wright (1964)
- Hierarchical Approach (split + TSP)
  - Fisher & Jaikumur (1981)
  - Taillard (1993)
- Multi-route Improvement Heuristics
  - Kinderwater and Savelsbergh (1997)

MetaHeuristics
- Tabu search, Rochat and Taillard (1995)
- Constraint Programming, Shown (1998)
- Tabu search Kelly and Xu (1999)
- Granular Tabu, Toth & Vigo (1998)
- Ant System, Gambardella & al. (1999)

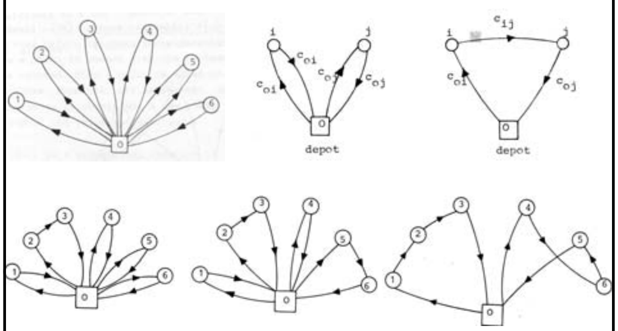
Clarke-Wright Saving Heuristic (1964). A constructive procedure proposed for VRP

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

Calculate saving $s_{ij}=c_{0i}+c_{0j}-c_{ij}$ and order the saving in increasing order

At each step find the largest saving s_{ij} where:
1. *i* and *j* are not in the same tour
2. neither *i* and *j* are interior to an existing route
3. vehicle and time capacity are not exceeded
4. link *i* and *j* together to form a new tour (replacing to other routes)

Clarke-Wright Saving Heuristic



(Fiala 1978)

Clarke-Wright Saving Heuristic

s(4, 6) = d(1, 4) + d(1, 6) - d(6, 4) = 57 + 61 - 71 = 47

TABLE 6-4 Distances (below diagonal) and savings (above the diagonal)

	1	2	3	4	5	6	7	8	9	10
1		—	—	—	—	—	—	—	—	—
2	25		39	48	25	18	5	0	1	5
3	43	29		48	14	8	0	3	2	23
4	57	34	52		55	47	15	3	6	20
5	43	43	72	45		77	36	19	26	49
6	61	68	96	71	27		50	36	47	86
7	29	49	72	71	36	40		39	46	57
8	41	66	81	95	65	66	31		78	66
9	48	72	89	99	65	62	31	11		83
10	71	91	114	108	65	46	43	46	36	

TABLE 6-5 Quantities of refuse to be collected at each point for Example 13.

Node	2	3	4	5	6	7	8	9	10
Quantity	4	6	5	4	7	3	5	4	4

TABLE 6-6 Savings list for Example 13.

Link	*Savings*	*Link*	*Savings*	*Link*	*Savings*
(6, 10)	86	(4, 6)	47	(4, 7)	15
(9, 10)	83	(7, 9)	46	(3, 5)	14
(8, 9)	78	(7, 8)	39	(3, 6)	8
(5, 6)	77	(2, 3)	39	(4, 9)	6
(8, 10)	66	(5, 7)	36	(2, 7)	5
(7, 10)	57	(6, 8)	36	(2, 10)	5
(4, 5)	55	(5, 9)	26	(3, 8)	3
(6, 7)	50	(2, 5)	25	(4, 8)	3
(5, 10)	49	(3, 10)	23	(3, 9)	2
(3, 4)	48	(4, 10)	20	(2, 9)	1
(2, 4)	48	(5, 8)	19	(2, 8)	0
(6, 9)	47	(2, 6)	18	(3, 7)	0

http://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.html

Clarke-Wright Saving Heuristic

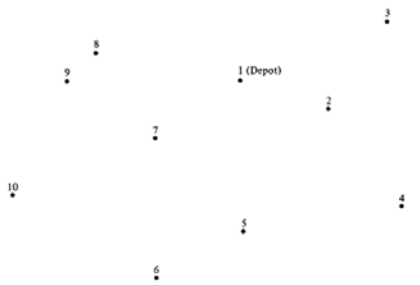


FIGURE 6.21 Depot and nine points to be visited.

Clarke-Wright Saving Heuristic

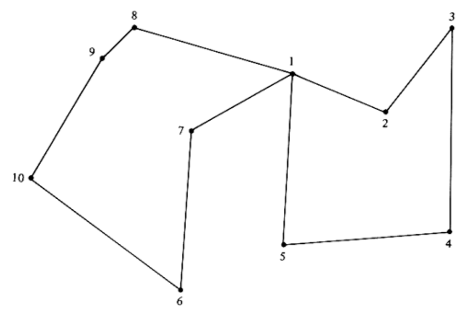


FIGURE 6.31 Final solution to the refuse-collection VRP, using Algorithm 6.7, when vehicle capacity is 23 units.

Clarke-Wright Saving Heuristic

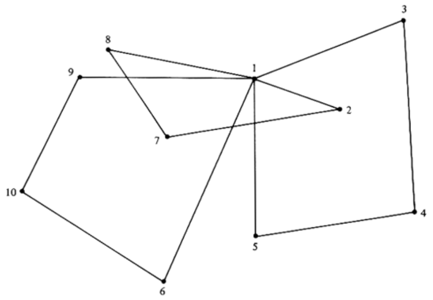


FIGURE 6.32 Final solution to refuse-collection VRP using Algorithm 6.7, when vehicle capacity is 16 units.

295

TWO PHASE METHODS:
Cluster First ruote second

Phase 1: Clustering

A clustering problem is solved to assign each customer to a single vehicle

Phase 2: Routing

Find the route for each vehicle (solving a TSP problem)

296

Cluster First – Route Second

Methods:

Elementary clustering methods

- Sweep algorithm
- Fisher-Jaikumar Generalized Assignment (GA) based algorithm
- Location-based heuristic

Truncated Branch-and-Bound approaches

- Levels of the exploration tree ⇒ vehicle routes
- Each level contains a set of (partial) feasible routes generated by one or more criteria (e.g., savings)
- Branching ⇒ route selection

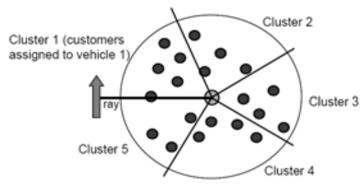
Petal Algorithm

297

Cluster First - Route Second

Sweep algorithm:

- Planar VRP
- Feasible cluster initially obtained by rotating a ray centered at the depot
- A vehicle route is found by solving a TSP problem for each cluster

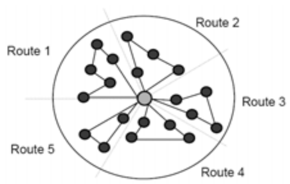


298

Cluster First - Route Second

Sweep algorithm:

- Planar VRP
- Feasible cluster initially obtained by rotating a ray centered at the depot
- A vehicle route is found by solving a TSP problem for each cluster



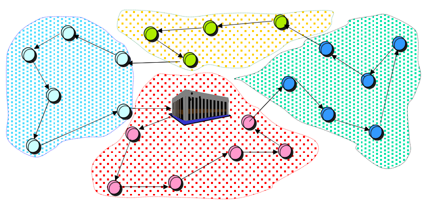
299

TWO PHASES APPROACH

Routing □ Clusterin

1. **Routing**:

2. **Clustering**:



300

Multi-Route Improvement Heuristic

Based on LS: exploration of a neighbourhood N(x) of solutions

N(x) is built using “moves”

Possible moves:

- Insert a customer in a different position in the sequence of visit
- Swap the positions of a pair of customers
- k-Opt

Two classes of methods:

Single route improvement

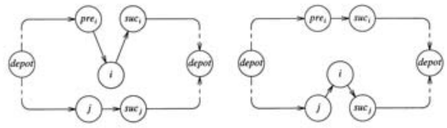
the assignment of customers to routes (vehicles) not change (analogous to TSP improvement heuristic)

Multi route improvement

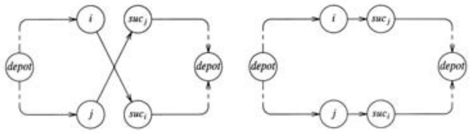
the moves may also change the customer-route assignment

Multi-Route Improvement Heuristic

Kinderwater and Savelsbergh (1997)



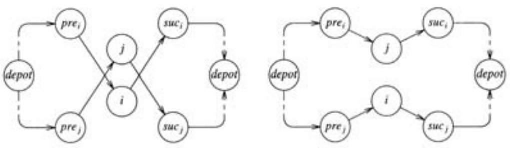
Customers relocation



Crossover

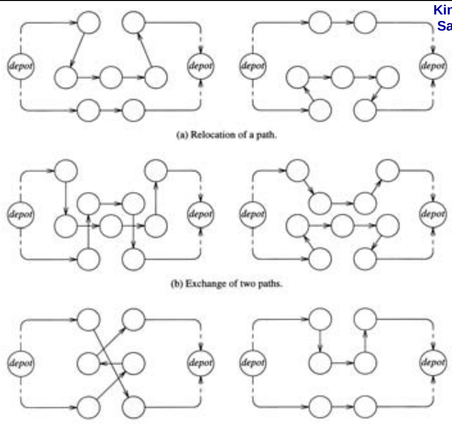
Multi-Route Improvement Heuristic

Kinderwater and Savelsbergh (1997)



Customers exchange

Kinderwater and Savelsbergh (1997)



(a) Relocation of a path.

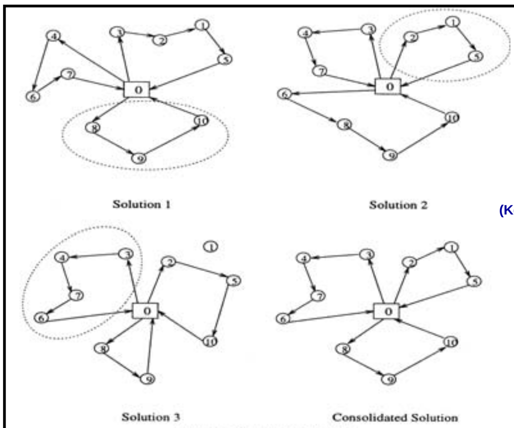
(b) Exchange of two paths.

(c) A crossover plus 2-exchange.

Tabu Search with set-partition based Heuristic

(Rochat & Taillard 1995, Kelly & Xu 1999)

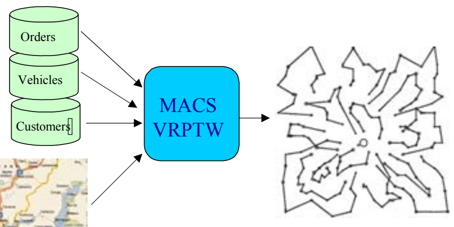
1. Keep an adaptive memory as a pool of good solutions
2. Some element (single tour) of these solutions are combined together to form new solution (more weight is given to best solutions)
3. partial solutions are completed by an insertion procedure.
4. Tabu search is applied at the tour level



(Kelly & Xu1999)

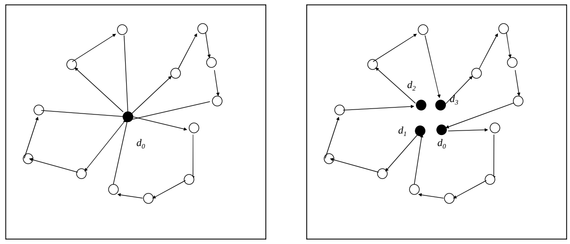
Figure 1. Two-phase procedure.

VRPTW: MACS-VRPTW: Vehicle Routing Problem with TWindows



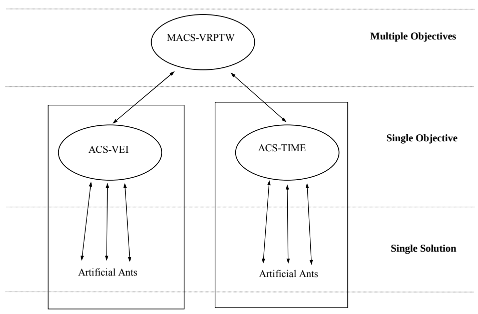
MACS-VRPTW: Vehicle Routing Problem with Time Windows Gambardella, Taillard, Agazzi, *New Ideas in Optimization*, 1999

MACS-VRPTW: Vehicle Routing Problem with Time Windows, Gambardella, Taillard, Agazzi, 1999



VRPTW is transformed into a TSP by adding m-1 new depots

MACS-VRPTW (Gambardella et al.1999)



VRP-TW: in case of vehicles and distance minimization two ant colonies are working in parallel on the two objective functions

MACS-VRPTW

```
/* MACS-VRPTW: Multiple Ant Colony System for Vehicle Routing Problems with Time Windows */
Procedure MACS-VRPTW()
1. /* Initialization */
    /* ψ^gb is the best feasible solution: lowest number of vehicles and shortest travel time
     #active_vehicles(ψ) computes the number of active vehicles in the feasible solution ψ */
    ψ^gb ← feasible initial solution with unlimited number of vehicles produced
          with a nearest neighbor heuristic
2. /* Main loop */
   Repeat
         v  ← #active_vehicles(ψ^gb)
         Activate ACS-VEI(v - 1)
         Activate ACS-TIME(v)
         While ACS-VEI and ACS-TIME are active
             Wait an improved solution ψ from ACS-VEI or ACS-TIME
              ψ^gb ← ψ
              if #active_vehicles(ψ^gb) < v then
                         kill ACS-TIME and ACS-VEI
         End While
    until a stopping criterion is met
```


ACS-TIME

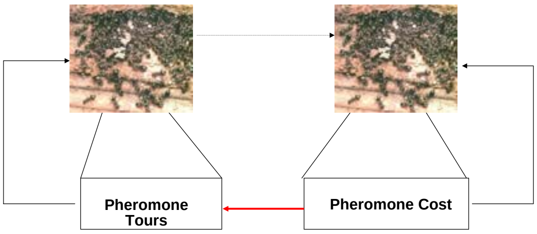
```
/* ACS-TIME: Travel time minimization. */
Procedure ACS-TIME(v)
/* Parameter v is the smallest number of vehicles for which a feasible solution has been computed */
1. /* Initialization */
   initialize pheromone and data structures using v
2. /* Cycle */
   Repeat
         for each ant k
             /* construct a solution ψ^k */
            new_active_ant(k, local_search=TRUE, 0)
         end for each
         /* update the best solution if it is improved */
         if ∃ k : ψ^k is feasible and J_ψ^k < J_ψ^gb then
                send ψ^k to MACS-VRPTW
          /* perform global updating according to Equation 2*/
         τ_ij = (1 − ρ)·τ_ij + ρ/J_ψ^gb          ∀(i, j) ∈ ψ^gb
  until a stopping criterion is met
```


ACS-VEI

```
/* ACS-VEI: Number of vehicles minimization. */
Procedure ACS-VEI(s)
 /* Parameter s is set to v-1, that is, one vehicle less than the smallest number of vehicles for which a feasible solution
    has been computed
   #visited_customers(ψ) computes the number of customers that have been visited in solution ψ */
1. /* Initialization */
   initialize pheromone and data structures using s
   ψ^ACS-VEI ← initial solution with s vehicles produced with a nearest neighbor
          heuristic. /* ψ^ACS-VEI is not necessary feasible */
2. /* Cycle */
    Repeat
         for each ant k
             /* construct a solution ψ^k */
            new_active_ant(k,local_search=FALSE,IN)
            ∀ customer j ∈ ψ^k: IN_j ← IN_j + 1
         end for each
         /* update the best solution if it is improved */
         if ∃ k : #visited_customers(ψ^k) > #visited_customers(ψ^ACS-VEI) then
                ψ^ACS-VEI ← ψ^k
                ∀ j: IN_j ← 0 /* reset IN */
                if ψ^ACS-VEI is feasible then
                   send ψ^ACS-VEI to MACS-VRPTW
         /* perform global updating according to Equation 2 using both ψ^ACS-VEI and ψ^gb */
         τ_ij = (1 − ρ)·τ_ij + ρ/J_ψ^ACS-VEI          ∀(i, j) ∈ ψ^ACS-VEI
         τ_ij = (1 − ρ)·τ_ij + ρ/J_ψ^gb          ∀(i, j) ∈ ψ^gb
  until a stopping criterion is met
```

MACS-VRPTW

Tours minimization | Cost minimization



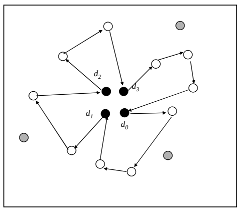
Search for a new feasible solution with one vehicle less

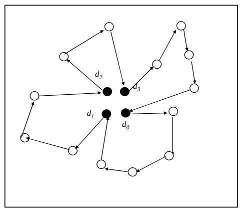
Minimize the cost of a feasible solution

313

MACS-VRPTW (Gambardella et al.1999)

Unfeasible solutions are repaired by insertion procedures





Feasible solutions are improved with local search procedures

314

Benchmark problems

With Time Windows (TSPLIB)

56 problems (Solomon, 1987) of six different types (C1,C2,R1,R2,RC1,RC2).

Each data set contains between eight to twelve 100-node problems.

- •C = clustered customers with easy TW.
- •R = customers location generated uniformly randomly over a square.
- •RC = a combination of randomly placed and clustered customers.
- •Sets of type 1 have narrow time windows and small vehicle capacity.
- •Sets of type 2 have large time windows and large vehicle capacity.

315

	R1		C1		RC1		R2		C2		RC2	
	VEI	DIST	VEI	DIST	VEI	DIST	VEI	DIST	VEI	DIST	VEI	DIST
MACS-VRPTW	**12.00**	1217.73	**10.00**	**828.38**	11.63	1382.42	**2.73**	967.75	3.00	**589.86**	**3.25**	1129.19
RT	12.25	1208.50	**10.00**	**828.38**	11.88	1377.39	2.91	961.72	3.00	**589.86**	3.38	1119.59
TB	12.17	1209.35	**10.00**	**828.38**	**11.50**	1389.22	2.82	980.27	3.00	**589.86**	3.38	1117.44
CR	12.42	1289.95	10.00	885.86	12.38	1455.82	2.91	1135.14	3.00	658.88	3.38	1361.14
PB	12.58	1296.80	10.00	838.01	12.13	1446.20	3.00	1117.70	3.00	589.93	3.38	1360.57
TH	12.33	1238.00	10.00	832.00	12.00	1284.00	3.00	1005.00	3.00	650.00	3.38	1229.00

Average of the best solutions computed by different VRPTW algorithms. Best results are in boldface. RT=Rochat and Taillard (1995), TB= Taillard et al. (1997), CR=Chiang and Russel (1993), PB=Potvin and Bengio (1996), TH= Thangiah et al. (1994)

316

	Old Best			New Best	
Problem	**source**	**vehicles**	**length**	**vehicles**	**length**
r112.dat	RT	10	953.63	**9**	982.140
r201.dat	S	4	1254.09	4	**1253.234**
r202.dat	TB	3	1214.28	3	**1202.529**
r204.dat	S	2	867.33	2	**856.364**
r207.dat	RT	3	814.78	**2**	894.889
r208.dat	RT	2	738.6	2	**726.823**
r209.dat	S	3	923.96	3	**921.659**
r210.dat	S	3	963.37	3	**958.241**
rc202.dat	S	4	1162.8	**3**	1377.089
rc203.dat	S	3	1068.07	3	**1062.301**
rc204.dat	S	3	803.9	3	**798.464**
rc207.dat	S	3	1075.25	3	**1068.855**
rc208.dat	RT	3	833.97	3	**833.401**
tai100a.dat	RT	11	2047.90	11	**2041.336**
tai100c.dat	RT	11	1406.86	11	**1406.202**
tai100d.dat	RT	11	1581.25	11	**1581.244**
tai150b.dat	RT	14	2727.77	14	**2656.474**

New best solution values computed by MACS-VRPTW.
RT=Rochat and Taillard (1995), S = Shaw (1998) TB= Taillard et al. (1997)

317

KTI/CTI

AntRoute: Fleet optimization for fuel distribution, Pina Petroli SA,CH

Fuel distribution

Multiple time windows

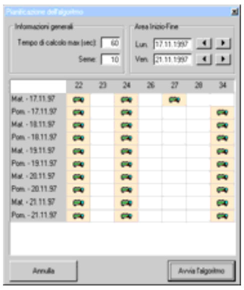
Stochastic quantity

Accessibility restrictions

318

Non homogeneous Vehicle characteristics

- Lorries must return to the depot for the lunch break
- Lorries are equipped with:
  - tanks of different capacities (7500, 11500, 23500 litres)
  - hoses of different diameters (1 ¼ and 2 inches) and length (30, 50, 120 metres)



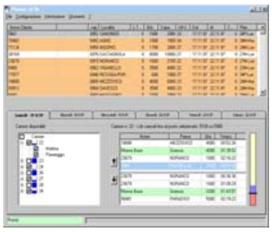
Half day availability
Half day planning with one week visibility

Service times assumptions

- Travel time between two nodes is computed according to distance, road type and weather conditions
- Customer lookup time: it is a parameter of the customer
- Set-up time: it is computed according to the length of the hose
- Oil delivery time: it is computed according to an approximate equation of the valve installed on the lorry

Minimise the total time required to serve all orders

Dynamic fleet optimization for fuel distribution, Pina Petroli SA, Grancia, CH



Improvement certified at

Pina Petroli SA, Grancia, CH

Computational time PC: 3 minutes
Average improvement: 20%

IDSIA

Der Spiegel

COMPUTER

Duft der Daten

Auf kürzestem Weg tragen Ameisen das Futter in ihren Bau. Programmierer nutzen jetzt die Tricks der Insekten, um die Abläufe in Fabriken zu optimieren.

Tag für Tag kämpfen sich die Lieferwagen der Firma Pina Petroli die steilen Hänge am Ufer des Luganer Sees empor, um selbst die entlegensten Gehöfte noch mit Brennstoff zu versorgen. Vor allem im Winter aber bleibt zuweilen die Küche kalt: Eine Straße ist verschneit, ein Wagen kaputt.

Nun soll das Fahrten-Management auf ungewöhnliche Weise optimiert werden: mit Hilfe der Ameisen. Das findet zumindest Luca Maria Gambardella, Berater von Pina Petroli und Direktor des Idsia-Instituts für künstliche Intelligenz im schweizerischen Manno: „Ameisen

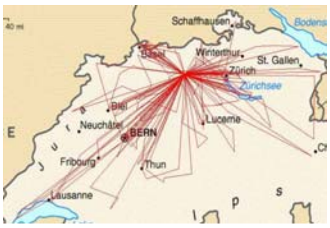
Verkehrsfunk der Ameisen...

Liegt einem Ameisenvolk ein Stein im Weg, so geht zunächst die Hälfte links und die andere Hälfte rechts herum. Beide Trupps hinterlassen eine Duftspur aus Pheromonen, die mit der Zeit verdunstet und schwächer wird.

Auf der kürzeren Route drängeln sich die Tiere stärker. Der dadurch intensivere Geruch

The Suhr distribution problem

- Central Depot
- Non-homogeneous fleet
- Customers accessibility restriction
- Customers time windows



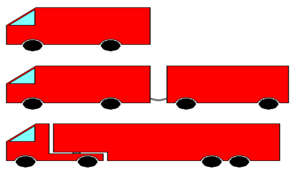
Objectives

1. N. of tours minimization
2. Cost minimization

Cost function = total_km * km_cost + total_time_violation * tv_cost

Non-homogeneous (infinite) fleet of vehicles

- Truck
- Truck + trailer
- Tractor unit + semi-trailer

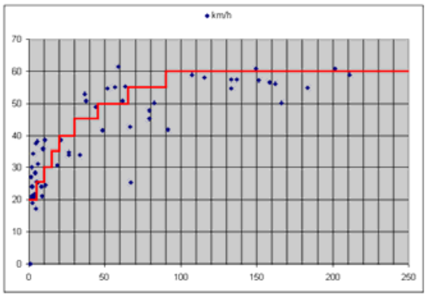


Remark: All capacities are expressed in pallets.

Other features:

- Embedded lift
- Refrigerated container
- Parking time (constant)
- Service time (variable)
- Trailer hook / unhook time (constant)

Speed Model



Speed - Distances

0 – 5 km => 20 km/h
5 – 10 km => 25 km/h
10 – 15 km => 30 km/h
15 – 20 km => 35 km/h
20 – 30 km => 40 km/h
30 – 45 km => 45 km/h
45 – 65 km => 50 km/h
65 – 90 km => 55 km/h
90 ... km => 60 km/h

325

Ant-Route
The first prototype test

Total number of orders	228
Total number of pallets	1736
Type of trucks available	MW+ANH, SS
MW+ANH capacity	35 pallets
SS capacity	33 pallets
MW capacity	17 pallets
Kilometric cost	3.10 fr. / km
Time windows violation cost	75.00 fr. / hour
Time windows width	60 min
Suhr opening - closing time	05:00 - 22:00
Trailer unhooking time	10 min
Trailer hooking time	10 min
Parking time	10 min
Unloading time	90 sec / pallet

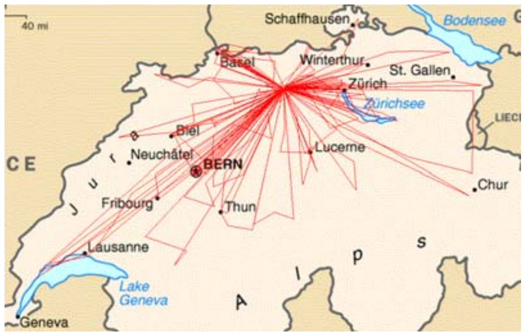
326

Ant-Route results

	1st disp.	2nd disp.	Ants	Gap (1st disp.)
Total number of tours	59	60	58	-1
Number of tours with AHZ	47	43	53	+ 6
Number of tours with SS	11	15	0	- 11
Number of tours with MW SOLO	1	2	5	+ 4
Average truck filling percentage	85.75%	85.50%	88.91%	+ 3.16%
Average number of orders per tour	3.881	3.816	3.931	+ 0.05
Average number of pallets per tour	29.424	28.933	29.931	+ 0.507
Total time windows violation time	159h 07m	131h 19m	12h 24m	- 146h 43m
Total waiting time	6h 30m	39h 14m	12h 24m	+ 5h 54m
Total delay time	152h 37m	92h 5m	0h 0m	- 152h 37m
Total km	10397 km	10793 km	10579 km	+ 182 km
Total cost	44'165 fr.	43'307 fr.	33'725 fr.	- 10'440 fr. 23.6%
Solution generation time	4 hours	4 hours	5 min	- 3h 55m

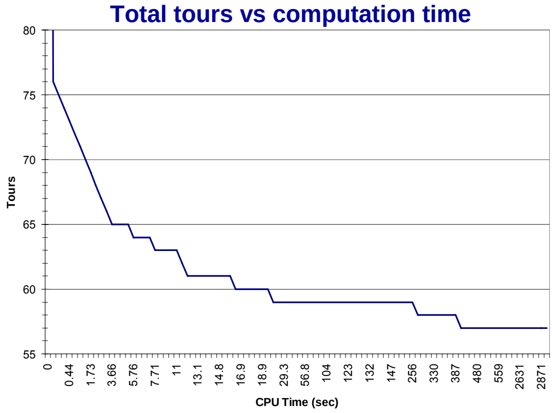
327

Ant-Route result after 5 minutes



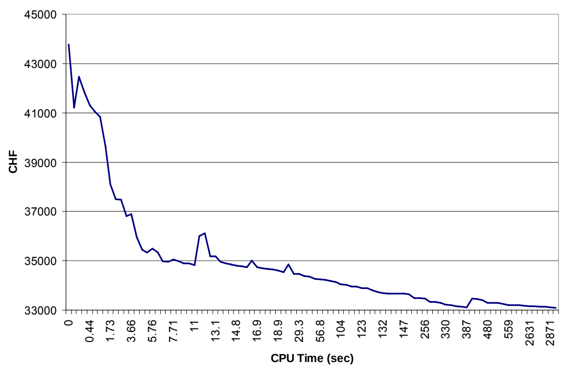
328

Total tours vs computation time



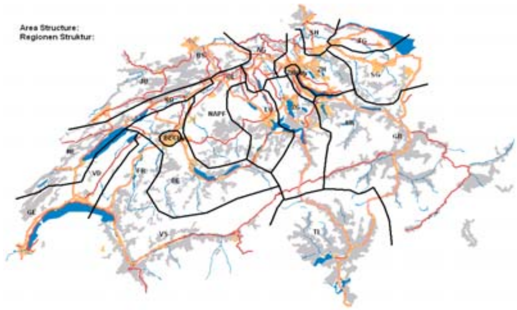
329

Total solution cost vs computation time



330

Regionalization

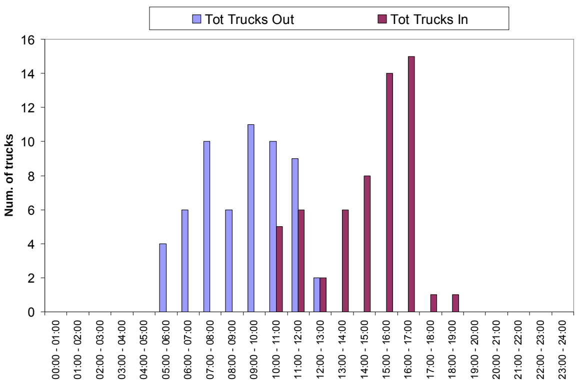


Ant Route Results

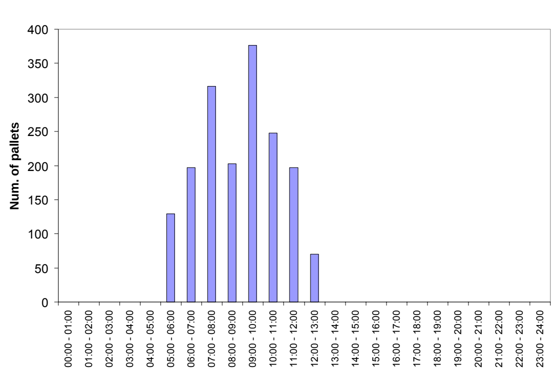
Main Objective: number of tours minimization

	Number of tours	Tot KM	Km x tour	Loading %	Computation Time
Time windows +- 30 minutes With Distribution areas	-12%	-2,2%	+12%	+13,5%	5 min Vs 4 persons 4 hours
Free Optimization No Time Windows No areas	-21,5%	-14,2%	+10%	+27%	5 min

Trucks distribution during the day

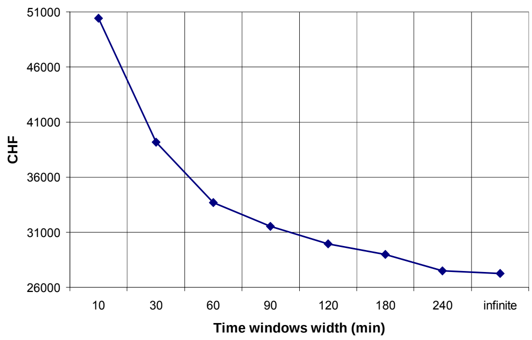


Pallets distribution during the day

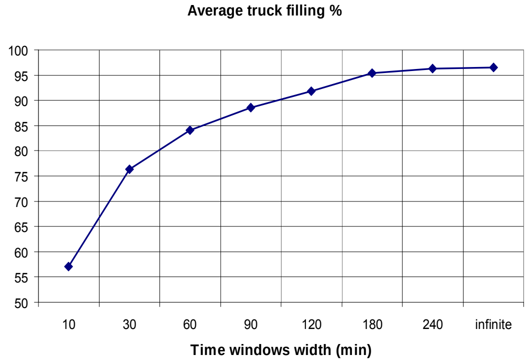


Ant-Route as a Strategic tool

Ant-Route as a Strategic tool



Ant-Route as a Strategic tool



337

Ant-Route: about the algorithm…

Foundations

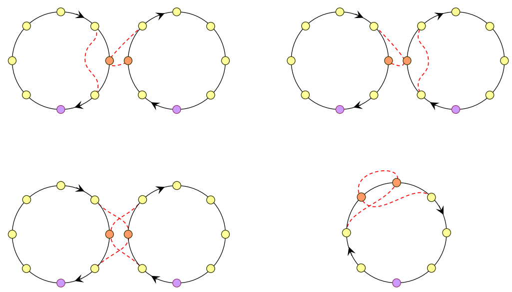
- MACS-VRPTW (1999, Gambardella, Taillard, Agazzi)
- Two colonies of ants
- Constructive phase (exploration & exploitation) + Local search

Extensions & Adaptations

- Vehicle choice at the start of each tour (pheromone based)
- Trailer hooking / unhooking management
- Constrained tours shape (to cope with dispatchers tastes…)
- Area structure management
- Starting time of each tour
- Vehicles usage optimization

338

Local search exchanges

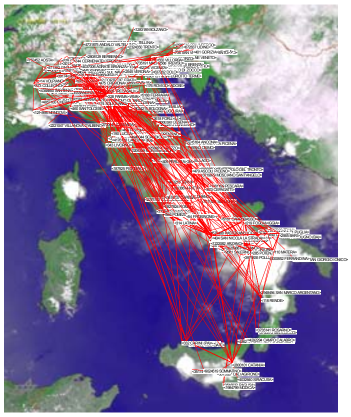


339

AntOptima
we speed up your business

NUMBER 1 LOGISTICS GROUP

AntRoute



340

Number1 Logistics Group Italia

Number1 is the largest Italian logistic operator (Barilla group)

Moves goods from factories to stores

700/1000 vehicles x day

No own fleet but all external trucks

Multiple starting points

Pick-up and delivery along Italy

341

The distribution problem of Number1

- Pick-up & Delivery: there is not a central depot
- Every order has a source point and a destination point
- Every point of the distribution network has a time window
- Every point of the network has a constant service time
- Heterogeneous point typology: providers, depots, clients
- Homogeneous fleet of vehicles

Objective:

Maximization of the average tours efficiency.

This should implicitly have as a side effect the minimization of the number of tours and of the total km.

342

Homogeneous (infinite) fleet of vehicles

Tractor unit + semi-trailer 

- Each vehicle has two capacities: Nominal and Maximum
- Each capacity has three dimensions: pallets, kg, m^3

Unit of measurement	Nominal capacity	Maximum capacity
pallets	33	34
kg	27000	28350
m^3	76	76

Tours efficiency

$$\eta_i = \frac{\sum_{j=1}^{M_i} q_j l_j}{Q_i L_i} \qquad f = \frac{\sum_{i=1}^{N} \eta_i}{N}$$

η_i = Efficiency of the *i-th* tour
M_i = Amount of orders in the *i-th* tour
Q_i = Nominal capacity of the vehicle associated with the *i-th* tour
L_i = Total length (km) of the *i-th* tour
q_j = Pallets of the *j-th* order
l_j = Distance (km) between source and destination points of the j-th order
N = Total amount of tours
f = Average tours efficiency (= the objective function)

Constraints

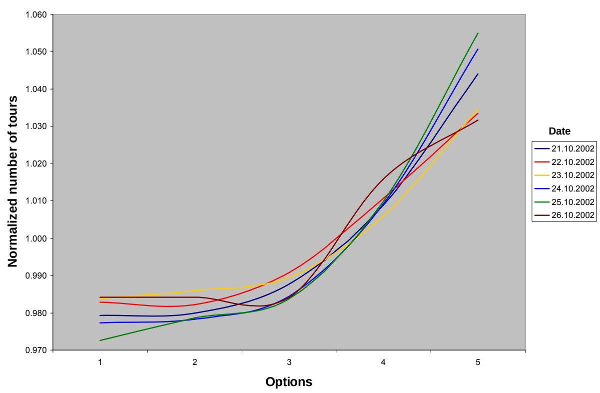
1. Respect of the time windows at each distribution point
2. Respect of the max. capacities of the vehicles
3. At most 4 points per tour
4. At most 2 clients per tour
5. All pick-ups of a tour must have the same date
6. All deliveries of a tour must have the same date
7. At most 200 km between two consecutive pick-ups
8. At most 200 km between two consecutive deliveries
9. At most 9 hours of travel per day
10. Pick-ups & deliveries cannot be interleaved
11. Order groups cannot be split into different tours

Feasibility study

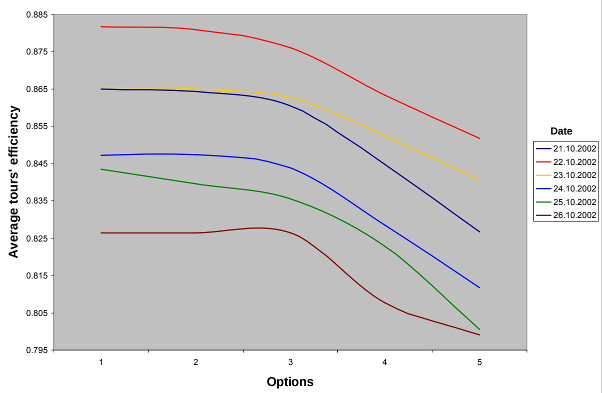
Different constraints scenarios have been evaluated

Option	Max points per tour	Max clients per tour	Use pick-up regions	Use delivery regions
1	-	-	NO	NO
2	-	2	NO	NO
3	4	2	NO	NO
4	4	2	YES	NO
5	4	2	YES	YES

Feasibility study



Feasibility study

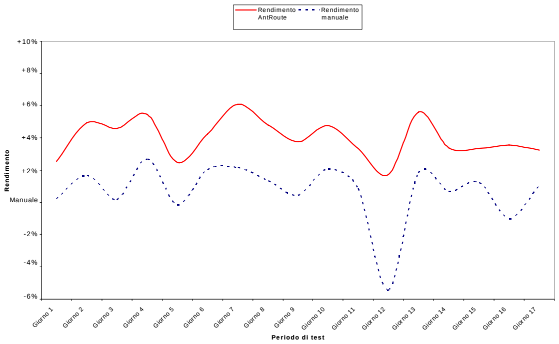


Final version acceptance test

- Integration of further constraints and requests
- Refinement of the algorithm
- The challenge: Number1 vs ANT-Route over one month
- Number1 tours penalization

	Number1	ANT-Route	Absolute difference	Relative difference
Tours				**-2.63%**
Total km				**-1.39%**
Efficiency without penalty			**+3.17%**	-
Efficiency with penalty			**+4.19%**	-

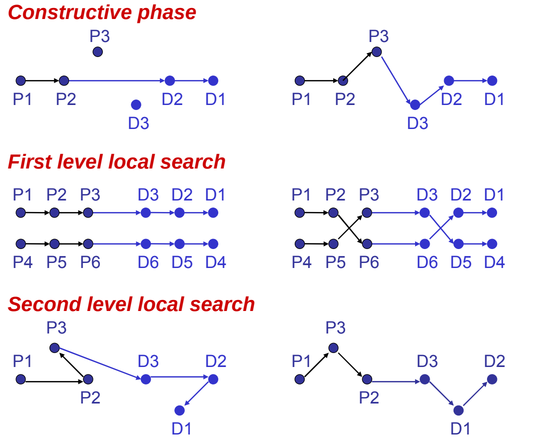
Performances Planner VS AntRoute



ANT-Route: Algorithm description

Same philosophy as in MACS-VRPTW but…

- Only one colony of ants (efficiency maximization)
- Each order involves two physical points (source and destination): this heavily increases the search space.
- The algorithm consists of:
  - a constructive phase using a LIFO policy;
  - a first level local search exchanging orders between different tours and preserving the LIFO structure;
  - a second level local search exchanging points within each tour individually.



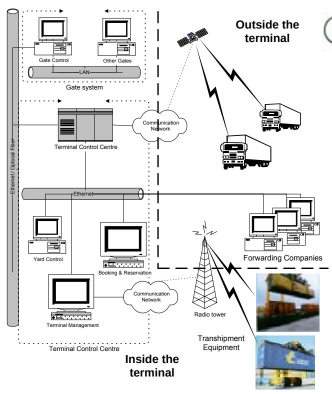
Number1 Logistics Group Italia

AntRoute is fully integrated in the operative process

Continuous optimization of new orders

Performance improvement from 2 to 4-5%.

Performance Parma-Veneto from 86.5% to 89.9%.

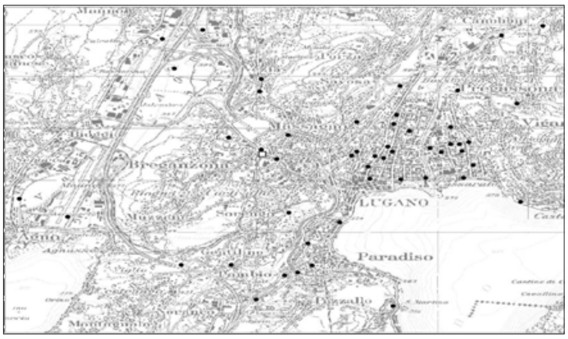


MOSCA EU 5 (2001-2003)

On line urban distribution with dynamic traffic information

IT, DE, CH, UK

From the real problem



355

The Dynamic Vehicle Routing Problem, Montemanni et. All 2003

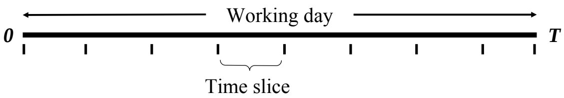
- **New orders** arrive when the **working day** has **already started**
- New orders have to be assigned to **vehicles** which may have **already left the depot**
- Vehicles **do not need to go back to the depot** when they are assigned new orders
- A **communication system** must exist between vehicles and the depot

Problems covered:

- Parcel collection
- Feeder systems
- Fuel distribution
- ...

356

Strategy for DVPRs



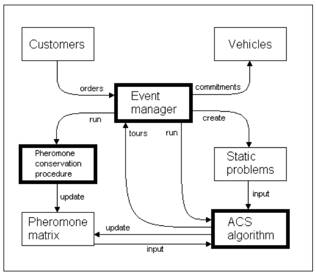
- The working day is **divided** into $\boldsymbol{n_{ts}}$ **time slices**
- For each **time slice** a **Static Vehicle Routing like problem** is solved by an **Ant Colony System** (ACS) algorithm

357

The ACS-DVRP algorithm. Elements

1. Event manager
- Collects new orders
- Keeps trace of the already served orders
- Keeps trace of the current position of vehicles.
- Creates a sequence of SVRPs
- Assign orders to vehicles

2. Ant Colony algorithm
- Solves SVRPs



3. Pheromone conservation procedure
- Passes information about good solutions from a SVRP to the following one of the sequence

358

ACS-DVRP - Event Manager (2)

At the **end of a time slice** the following **operations** are carried out:

- **Orders starting** within the next $T_{ts} + T_{ac}$ (in the solution of the last SVPR) are **committed** to the respective vehicles
- A **new SVR-like problem** is created, where
  - New **starting positions** and **residual capacities** are calculated for the vehicles
  - **New orders** received during the last time slice are inserted and **committed orders** are deleted
- A **pheromone conservation strategy** is **run**
- The **ACS algorithm** is **run** for T_{ts} seconds

359

ACS-DVRP - Pheromone conservation

When the ACS algorithm finishes working on a SVRP:
- **Pheromone matrix** contains **encrypted information** about good solutions
- The **next SVRP** of the sequence is potentially **very similar** to the **SVRP just considered**

These considerations are used to **prevent optimization to restart** each time **from scratch**. The new pheromone matrix is then as follows:

$\tau_{ij} = (1-\gamma_r)\,\tau_{ij}^{old} + \gamma_r \tau_0$ for each pair of customers contained in both the new and the old SVRP

$\tau_{ij} = \tau_0$ for pairs involving new customers

γ_r is a new parameter which regulates pheromone conservation

360

Benchmarks description (1)

The problems have been originally **presented** in

- **Kilby et al.** "Dynamic VRPs: a study of scenarios". Technical report APES-06-1998, University of Strathclyde, 1998

The problems are derived from **well-known SVRPs**:

- 12 are from **Taillard**, "Parallel iterative search methods for vehicle-routing problems". *Networks*, 23(8):661-673, 1994
- 7 are from **Christofides and Beasley**, "The period routing problem". *Networks*, 14:237-256, 1984
- 2 are from **Fisher et al.** "A generalized assignment heuristic for vehicle routing". *Networks*, 11:109-124, 1981

361

Benchmarks description (2)

Information **added** (specified by Kilby et al.):

- The **length** of the working day (***T***).
- An **appearance time** for each customer
- A **service time** for each customer
- The **number of vehicles**, set at **50** for each problem

Extra **parameters** to be set (not specified by Kilby et al.):

- The **time of cutoff**. $T_{co} = 0.5\ T$
- The **advance commitment time**. $T_{ac} = 0.01\ T$

362

Number of time slices nts

n_{ts}		c100	f71	tai75a
10	Min	1004.58	311.95	1880.11
	Max	1145.20	399.26	2105.14
	Avg	1083.64	362.93	1963.19
25	Min	973.26	311.18	1843.08
	Max	1100.61	420.14	2043.82
	Avg	1066.16	348.69	1945.20
50	Min	1131.95	333.25	1966.92
	Max	1228.97	452.73	2133.87
	Avg	1185.25	417.74	2019.82

Travel times. 5 runs for each problem ($\gamma_r = 0.3$)

$n_{ts} = 25$ is the best choice

363

Computational results

- No pheromone = multi-start local search algorithm
- ACS-DVRP = the method we propose

ACS leads to the following **improvements**:

- 4.86% for Min
- 2.40% for Max
- 4.37% for Avg

ACS has always the **best values** for **Min** and **Avg**

Problem	No pheromone			ACS-DVRP		
	Min	Max	Avg	Min	Max	Avg
c100	1080,33	1169,67	1124,04	973,26	1100,61	1066,16
c100b	978,39	1173,01	1040,99	944,23	1123,52	1023,60
c120	1546,50	1875,35	1752,31	1416,45	1622,12	1525,15
c150	1468,36	1541,54	1493,06	1345,73	1522,45	1455,50
c199	1774,33	1956,76	1898,20	1771,04	1998,87	1844,82
c50	693,82	756,89	722,15	631,30	756,17	681,86
c75	1066,59	1142,32	1098,85	1009,38	1086,65	1042,39
f134	16072,97	17325,73	16866,79	15135,51	17305,69	16083,56
f71	369,26	437,15	390,48	311,18	420,14	348,69
tai100a	2427,07	2583,02	2510,29	2375,92	2575,70	2428,38
tai100b	2302,95	2532,57	2406,91	2283,97	2455,55	2347,90
tai100c	1599,19	1800.85	1704,40	1562,30	1804,20	1655,91
tai100d	2026,82	2165,39	2109,54	2008,13	2141,67	2060,72
tai150a	3787,53	4165,42	3982,24	3644,78	4214,00	3840,18
tai150b	3313,03	3655,63	3485,79	3166,88	3451,69	3327,47
tai150c	3090,47	3635,17	3253,08	2811,48	3226,73	3016,14
tai150d	3159,21	3541,27	3323,57	3058,87	3382,73	3203,75
tai75a	1911,48	2140,57	2012,13	1843,08	2043,82	1945,20
tai75b	1634,83	1934,35	1782,46	1535,43	1923,64	1704,06
tai75c	1606,20	1886,24	1695,50	1574,98	1842,42	1653,58
tai75d	1545,21	1641,91	1588,73	1472,35	1647,15	1529,00
Total	**53454,54**	**59060,81**	**56241,50**	**50855,94**	**57645,52**	**53784,02**

Travel times. 5 runs for each problem (Intel P4 1.5 GHz)

364

END

365

References

Aarts, E.H., J.K. Lenstra. 1997. Introduction. E. H. Aarts, J. K. Lenstra, eds. Local Search in Combinatorial Optimization. John Wiley & Sons, Chichester, UK. 1–17.

Ascheuer, N., L. F. Escudero, M. Grotschel, M. Stoer. 1993. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization* 3 25–42.

P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, É. D. Taillard, A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows, *Transportation Research-C* 5, 1997, 109-122.

Bentley J.L., "Fast algorithms for geometric traveling salesman problem," *ORSA Journal on Computing*, vol. 4, pp. 387–411, 1992.

Chen, S., S. Smith. 1996. Commonality and genetic algorithms. Technical Report CMU-RI-TR-96-27, The Robotic Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

W. C. Chiang, R. Russel, Hybrid Heuristics for the Vehicle Routing Problem with Time Windows, Working Paper, Department of Quantitative Methods, University of Tulsa, OK, USA, 1993.

Clark G, Wright, J.W., Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research*, 12, 568-581, 1964.

Dantzig G. B., Fulkerson R., and Johnson S. M. , Solution of a large-scale traveling salesman problem, *Operations Research* 2 (1954), 393-410.

Dorigo M., Di Caro G., Gambardella L.M., "Ant Algorithms for Distributed Discrete Optimization", *Artificial Life*, Vol. 5, N. 2, 1999.

Dorigo M., Gambardella L.M, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation* 1,1, pp. 53-66, 1997

Ercoli C., Re B., Progetto TSP per Algoritmi e Complessità Computazionale del Corso di Laurea in Informatica, Università di Camerino, Italy, a.a. 2003-2004

Fiala F., Vehicle Routing Problems, GMD-Mitteilungen, 46, Boon, 1978.

Fisher M. L., Optimal Solution of Vehicle Routing Problems Using Minimum K-trees, *Operations Research* 42, 1994, 626-642.

Fisher M. L., Jaikumur R., A generalized assignment heuristic for vehicle routing, *Network* 11, 109-124, 1981.

Flood M. M., The Traveling Salesman Problem, *Operations Research* 4, 1956, 61-75.

366

References

Gambardella L.M, Dorigo M, An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, *INFORMS Journal on Computing*, vol.12(3), pp. 237-255, 2000.

Gambardella L.M, Taillard E., Agazzi G., MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows , In D. Corne, M. Dorigo and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, London, UK, pp. 63-76, 1999.

Gomory R. E. , Outline of an algorithm for integer solutions to linear programs, *Bulletin of the American Mathematical Society* 64 (1958), 275-278.

Gomory R. E. , An algorithm for integer solutions to linear programs, in: Recent Advances in Mathematical Programming (R. L. Graves and P. Wolfe, eds.), McGraw-Hill, New York, 1963, pp. 269-302.

Helsgaun K., An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, DATALOGISKE SKRIFTER (Writings on Computer Science), Roskilde University, No. 81, 1998.

Johnson, D.S., L.A. McGeoch. 1997. The traveling salesman problem: a case study, E. H. Aarts, J. K. Lenstra, eds. Local Search in Combinatorial Optimization. John Wiley & Sons, Chichester, UK. 215–310.

P. Kilby, P. Prosser, P. Shaw, Guided Local Search for the Vehicle Routing Problems With Time Windows, in *Meta-heuristics: Advances and Trends in Local Search for Optimization*, S.Voss, S. Martello, I.H. Osman and C.Roucairol (eds.), Kluwer Academic Publishers, Boston, 1999, 473-486.

Kindervater, G.A.P., M.W.P. Savelsbergh. 1997. Vehicle routing: handling edge exchanges, E. H. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK. 311–336.

Montemanni R., Gambardella L.M., Rizzoli A.E., Donati A.V.. A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System. ODYSSEUS 2003: Second International Workshop on Freight Transportation and Logistics, Palermo, Italy, 27-30 May 2003

Laporte G. Gendreau M., Potvin J-Y., Semet F., Classical amd Modern Heuristics for the vehicle routing problem, *International Transaction in Operational Research*, vol. 7, pp. 285-300, 2000.

O. Martin, S.W. Otto, and E.W. Felten, “Large-step Markov chains for the TSP incorporating local search heuristics,” *Operations Research Letters*, vol. 11, pp. 219-224, 1992.

Padberg M. W. and Grötschel M. , Polyhedral computations, in: The Traveling Salesman Problem (E. L. Lawler et al., eds.), Wiley, Chichester, 1995, pp.307-360.

367

References

Padberg M. W. and Rinaldi G. , A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, *SIAM Review* 33 (1991), 60-100.

Papadimitriou C., Steiglitz K. *Combinatorial optimization : algorithms and complexity* by, Prentice-Hall, New Jersey, 1982

J.-Y. Potvin, S. Bengio, The Vehicle Routing Problem with Time Windows - Part II: Genetic Search, *INFORMS Journal of Computing* 8, 1996, 165-172.

G. Reinelt, The traveling salesman: computational solutions for TSP applications. Springer-Verlag, 1994.

C. Rego, C. Roucairol, A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem, in *Meta-heuristics: Theory and applications*, I.H. Osman, J. Kelly (eds.), Kluwer Academic Publishers, Boston, 1996, 661-675.

Y. Rochat, É. D. Taillard, Probabilistic Diversification and Intensification in Local Search for Vehicle Routing, *Journal of Heuristics* 1, 1995, 147-167.

M. Solomon, Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints, *Operations Research* 35, 1987, 254-365.

P. Shaw, Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming* (CP '98), M. Maher and J.-F. Puget (eds.), Springer-Verlag, 1998, 417-431.

É. D. Taillard, Parallel Iterative Search Methods for Vehicle Routing Problems, *Networks* 23, 1993, 661-673.

É. D. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows, *Transportation Science* 31, 1997, 170-186.

S. R. Thangiah, I. H. Osman, T. Sun, Hybrid Genetic Algorithm Simulated Annealing and Tabu Search Methods for Vehicle Routing Problem with Time Windows, Technical Report 27, Computer Science Department, Slippery Rock University, 1994.

P. Toth, D. Vigo, eds. The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

J. Xu, J. Kelly, A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem, *Transportation Science* 30, 1996, 379-393.

368