Advanced Metaheuristics

Daniele Vigo
D.E.I. - Università di Bologna
daniele.vigo@unibo.it

Main families of Metaheuristics

- Single-solution methods
  - Basic: Tabu Search, Simulated Annealing …
  - Advanced:
    - Iterated Local Search
    - Variable Neighborhood Search
    - Large Neighborhood Search
    - Ruin&Recreate

Multistart Local Search (MLS)

- Repeatedly applies a LS algorithm

  repeat

  - generates a starting solution x
    (randomly or with random parameter);
  - apply Local Search and find the local optimum : x' = LS(x)
  - if z(x') < z(x*) then x* = x'

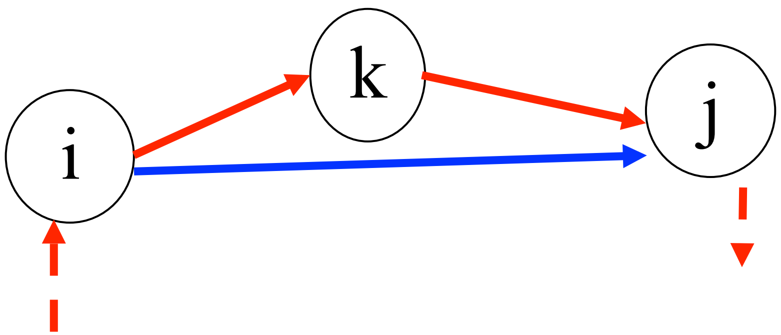
  until stop condition

- easy to implement but not always good
- The solutions are randomly generated thus the local optima are independently distributed
  - in large problems tend to be equal

Example

- Cheapest insertion algorithm for TSP
- Parametric insertion cost

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



Iterated Local Search (ILS)

- Evolution of MultiStart LS
  - uses the local optimum (perturbed) of the previous iteration as a starting point for the current iteration and possibly update it

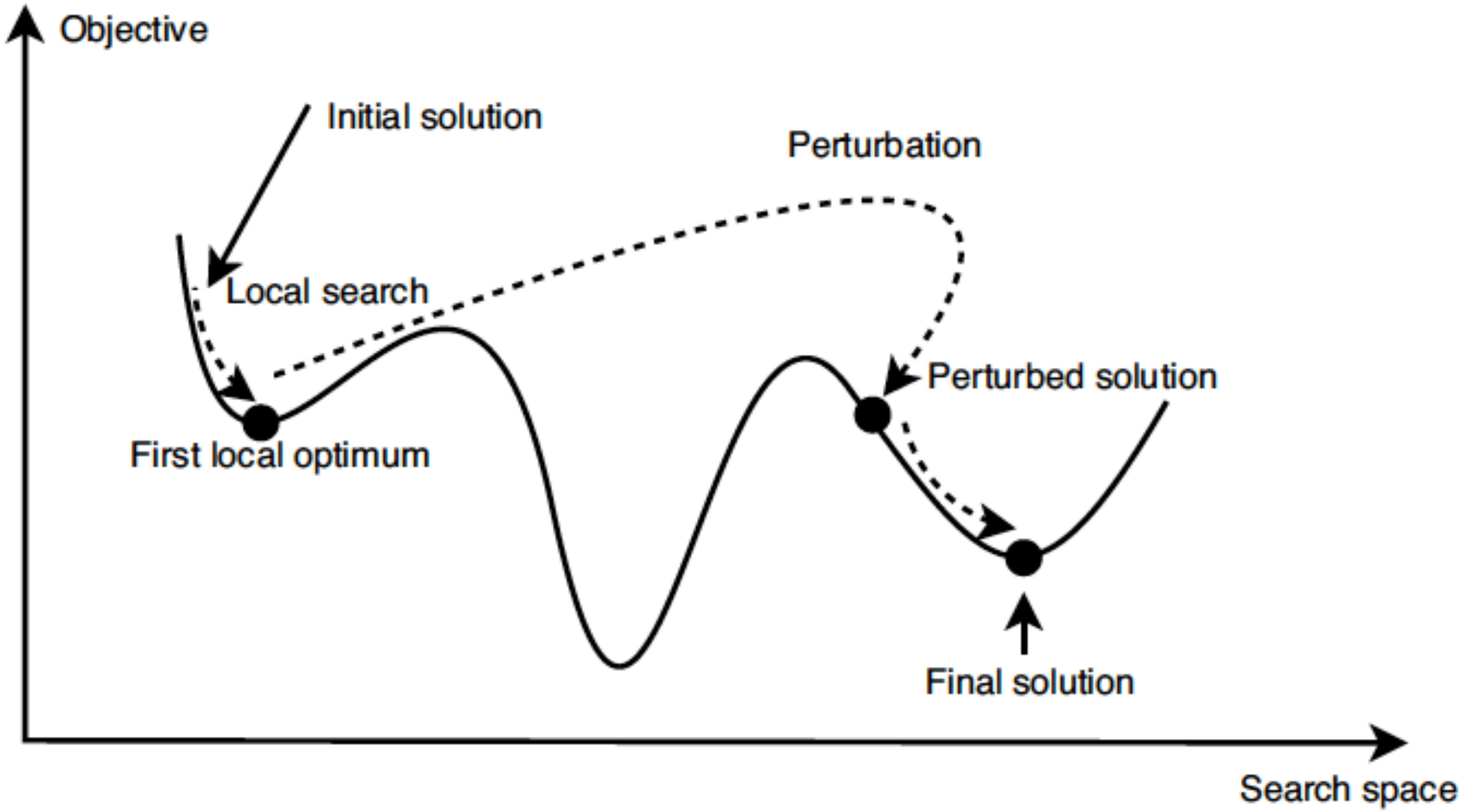
x*= local optimum (apply LS to a random solution)

repeat

- perturb x*;
- x'=LS(x*)
- possibly replace x* with x'

until stop condition

Iterated Local Search (ILS)



Iterated Local Search (ILS)

- Perturbation
  - random modifications
  - sequence of moves (of a different neighborhood)
  - careful choice of the perturbation intensity
    - small: risk of cycling on the local optimum
    - large: loss of information about the optimum ➔ MLS
- Acceptance criteria
  - Probabilistic (es. SA)
  - Deterministic (es. if improving or within a threshold from the best solution)

Iterated LS

Algorithm 4.2: ILS($N_{iter}, N_{rand}, Tour$)

$costCurrent \leftarrow CostOfInitialTour$
$bestTour \leftarrow Tour$
for $it \leftarrow 1$ **to** N_{iter}
    $Tour \leftarrow bestTour$
    for $r \leftarrow 1$ **to** N_{rand}
        $i, j \leftarrow RandomNumber(1, \ldots, |V_c|)$
        if $(i = j)$
            $p \leftarrow RandomNumber(1, \ldots, |V_c|, p \neq i)$
            1OPT($Tour, i, p$)
        else 2OPT($Tour, i, j$)
    while ($improvement$) $costNew \leftarrow$ PERFORM BEST MOVE($Tour$)
    if $(costNew < costCurrent)$
        $costCurrent \leftarrow costNew$
        $bestTour \leftarrow Tour$

Iterated LS

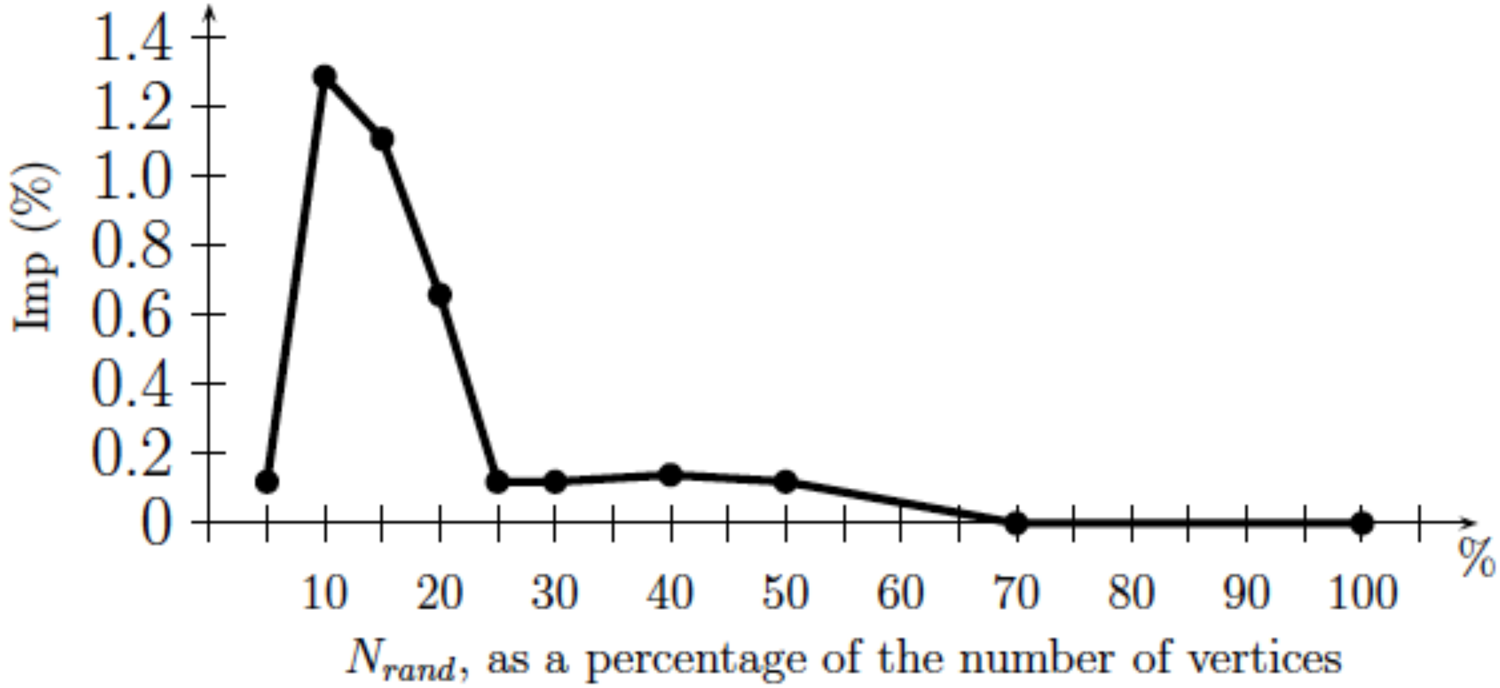
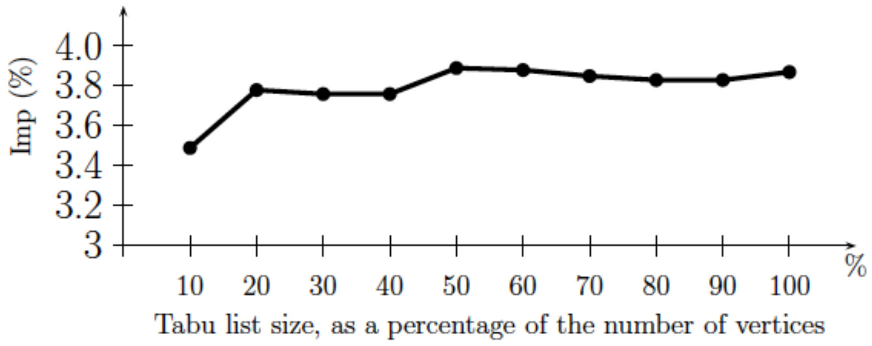


Figure 5: Performance of ILS, as a function of N_{rand}

Basic TS

Algorithm 4.1: TS(N_{iter}, $Tour$)

```
costCurrent ← CostOfInitialTour
for i ← 1 to N_iter
    (i, p, cost1) ← FindBestNotTabu1OptMove(Tour, costCurre
    (i, j, cost2) ← FindBestNotTabu2OptMove(Tour, costCurre
    if (cost1 < cost2)
        Implement1OptMove(Tour, i, p)
        UpdateTabuList(i, i)
    else
        Implement2OptMove(Tour, i, j)
        UpdateTabuList(i, j)
    CostCurrent ← UpdateCostCurrent()
```



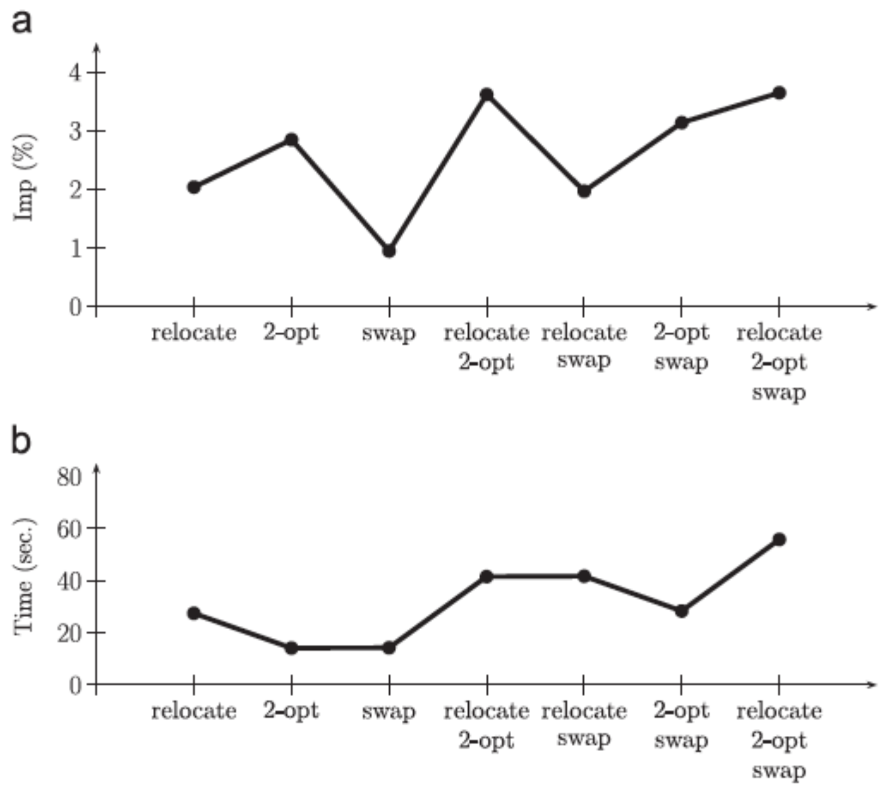
Iterated Tabu Search

Algorithm 4.3: ITS($N^*_{iter}, N_{rand}, Tour$)

$costCurrent \leftarrow CostOfInitialTour$
$bestTour \leftarrow Tour$
for $it \leftarrow 1$ **to** $N_{iter}*$
    $Tour \leftarrow bestTour$
    for $r \leftarrow 1$ **to** N_{rand}
        $i, j \leftarrow RandomNumber(1, \ldots, |V_c|)$
        if $(i = j)$
            $p \leftarrow RandomNumber(1, \ldots, |V_c|, p \neq i)$
            $1\textsc{Opt}(Tour, i, p)$
        else $2\textsc{Opt}(Tour, i, j)$
    $costCurrent \leftarrow \textsc{Tabu Search}(N^*_{iter}, Tour)$
    if $(costNew < costCurrent)$
        $costCurrent \leftarrow costNew$
        $bestTour \leftarrow Tour$

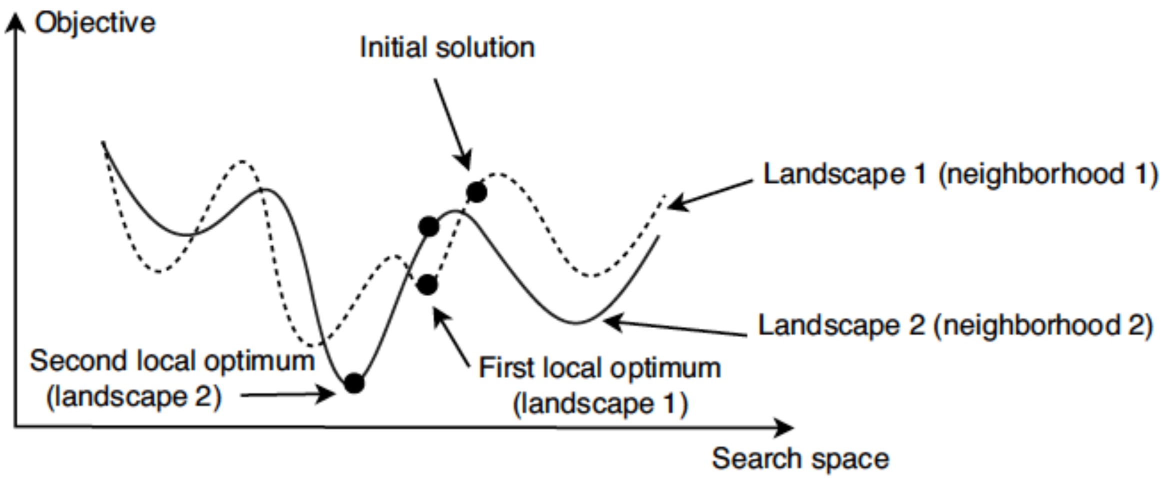
Multiple Neighborhoods

- Often several neighborhoods are available
- Which one use ?
- Combine them to obtain larger ones ?
- Ex. Erdogan et al 2012 for a TSP variant

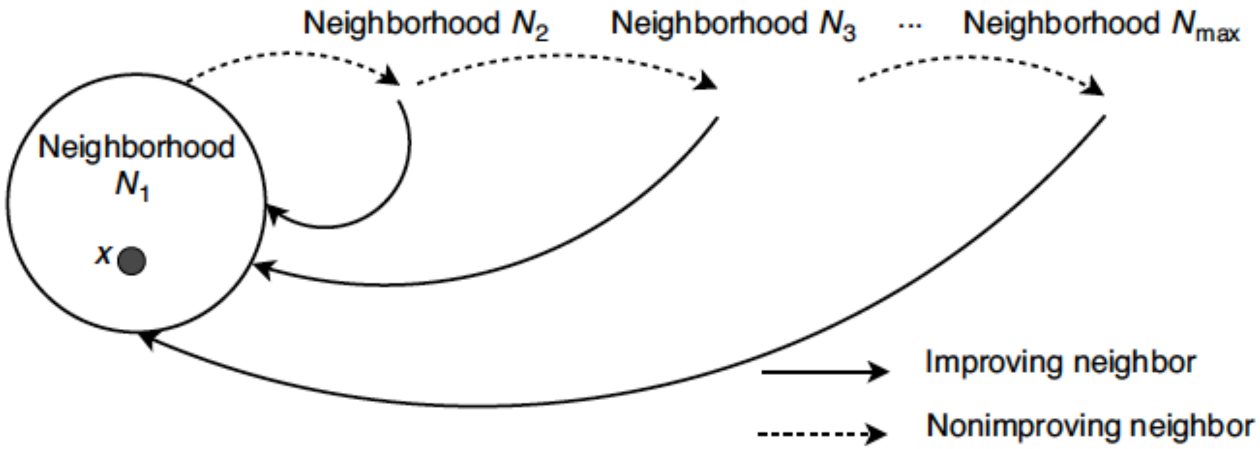


Variable Neighborhood Search

- Proposed by Mladenovich and Hansen (1997)
- Exploits different Neighborhoods N_k ($k=1,\dots, k_{max}$)
  - The Neighborhoods are applied in sequence
    - if the local optimum is not globally improving then $k=k+1$
    - otherwise the move is accepted and $k=1$



Variable Neighborhood Descent



Input: a set of neighborhood structures N_l for $l = 1, \ldots, l_{max}$.
$x = x_0$; /* Generate the initial solution */
$l = 1$;
While $l \leq l_{max}$ **Do**
  Find the best neighbor x' of x in $N_l(x)$;
  If $f(x') < f(x)$ **Then** $x = x'$; $l = 1$;
  Otherwise $l = l + 1$;
Output: Best found solution.

Variable Neighborhood Search

- Stochastic algorithm which uses various Neighborhoods N_k ($k=1,\ldots, k_{max}$)
- Iterative procedure based on 3 phases:
  - Shaking: generates a random move from $N_k(x) \rightarrow x'$
  - Local Search: apply LS to $x' \rightarrow x''$
  - Move: if x'' improving, accept it and restart from N_1, otherwise use N_{k+1}

Basic VNS

Input: a set of neighborhood structures N_k for $k = 1, \ldots, k_{max}$ for shaking.
$x = x_0$; /* Generate the initial solution */
Repeat
  $k = 1$;
  Repeat
    Shaking: pick a random solution $x^{'}$ from the k^{th} neighborhood $N_k(x)$ of x ;
    x'' =local search(x') ;
    If $f(x'') < f(x)$ **Then**
      $x = x^{''}$;
      Continue to search with N_1 ; $k = 1$;
    Otherwise k=k+1 ;
  Until $k = k_{max}$
Until Stopping criteria
Output: Best found solution.

General VNS

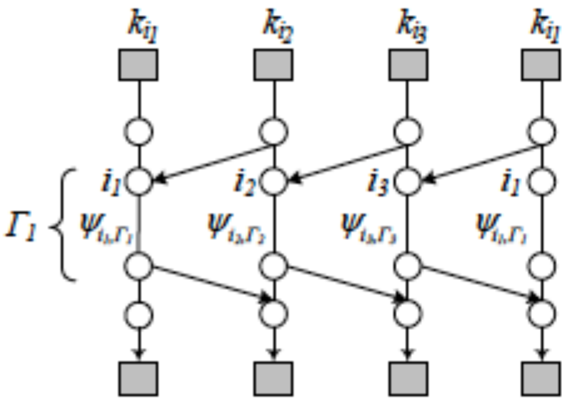
Input: a set of neighborhood structures N_k for $k = 1, \ldots, k_{max}$ for shaking.
a set of neighborhood structures N_l for $k = 1, \ldots, l_{max}$ for local search.
$x = x_0$; /* Generate the initial solution */
Repeat
For k=1 **To** k_{max} **Do**
Shaking: pick a random solution $x^{'}$ from the k^{th} neighborhood $N_k(x)$ of x ;
Local search by VND ;
For l=1 **To** l_{max} **Do**
Find the best neighbor $x^{''}$ of $x^{'}$ in $N_l(x^{'})$;
If $f(x^{''}) < f(x^{'})$ **Then** $x^{'} = x^{''}$; l=1 ;
Otherwise l=l+1 ;
Move or not:
If local optimum is better than x **Then**
$x = x^{''}$;
Continue to search with N_1 $(k = 1)$;
Otherwise k=k+1 ;
Until Stopping criteria
Output: Best found solution.

General VNS

- The critical issue is the choice of the Neighborhoods and their order
- Often parametric families are used

Cyclic-Exchange Neighborhoods (Thompson and Orlin 1989)

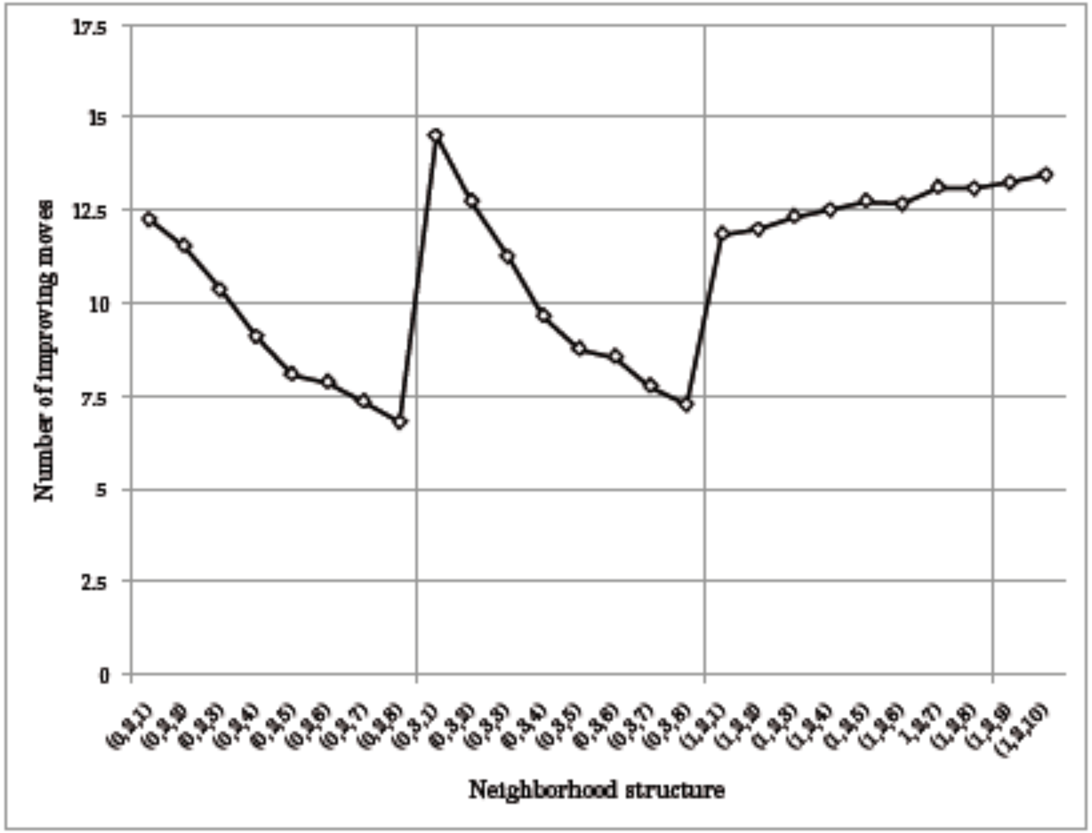
- Parameters
  - Φ: number of depots at which the routes originate
  - Ω: number of routes involved
  - Γ_{max}: maximum sequence length to exchange



No.	Φ	Ω	Γ_{max}	No.	Φ	Ω	Γ_{max}
1	0	2	1	14	0	3	6
2	0	2	2	15	0	3	7
3	0	2	3	16	0	3	8
4	0	2	4	17	1	2	1
5	0	2	5	18	1	2	2
6	0	2	6	19	1	2	3
7	0	2	7	20	1	2	4
8	0	2	8	21	1	2	5
9	0	3	1	22	1	2	6
10	0	3	2	23	1	2	7
11	0	3	3	24	1	2	8
12	0	3	4	25	1	2	9
13	0	3	5	26	1	2	10

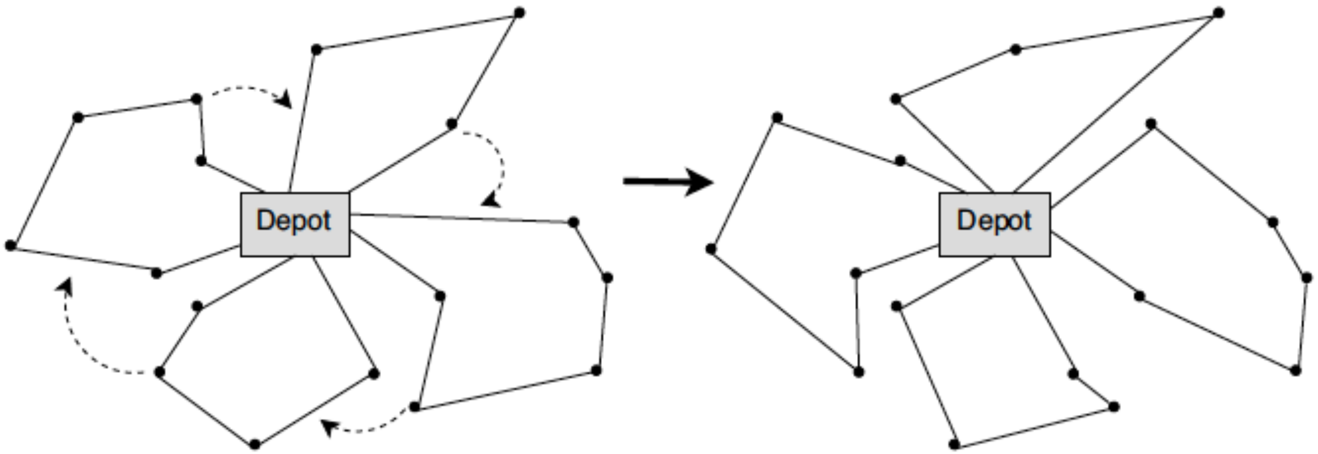
General VNS

- All Neighborhoods may provide a contribution



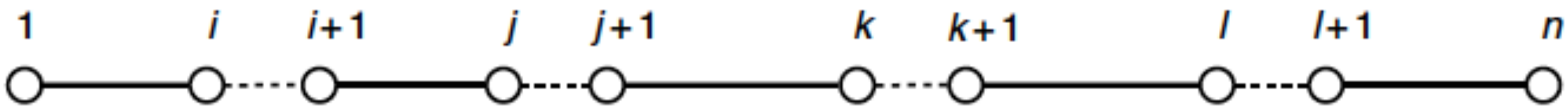
Very Large N. Search

- Local Search using Neighborhoods with very large cardinality (exponential)
- Ex. Ejection chains or cyclic exchanges



Very Large N. Search

- Neighborhood search can be perfromed:
- exactly (in some cases)
  - the best move is determined by solving an optimization problem
  - Ex. Dynasearch for the TSP
    - remove half of the arcs
    - the best recombination is foundby solve a shortest path on a suitably defined graph



Very Large N. Search

- Ex. Assignment Neighborhood for the TSP
  - remove n/2 vertices and form a subtour with the remaining ones
  - define the (square) matrix of reinsertion costs for the vertices in the subtour
  - select the best subset of reinsertions by solving an assignment problem in $O(n^3)$
  - is a restriction in which each reinsertion can be after a different vertex of the subtour

Very Large N. Search

- Heuristic search
  - generate just a heuristic solution belonging to the neighborhood
- Reinsertion LNS (Shaw, 1998)

Algorithm 1 LNS heuristic

```
Function LNS(s ∈ {solutions}, q ∈ ℕ )
  solution s_best=s;
  repeat
    s' = s;
    remove q requests from s'
    reinsert removed requests into s';
    if (f(s') < f(s_best)) then
      s_best = s';
    if accept(s', s) then
      s = s';
  until stop-criterion met
return s_best;
```

Ruin&Recreate

- Ruin&Recreate (Schrimpf et al., 2000)
  - remove q elements and reinsert them (heuristically)
- Removal (Ruin)
  - Random removal: random choice of removed elements
  - Shaw removal: remove “similar” elements (e.g. customers with similar demand) so that the reinsertion will be easier
  - Worst removal: remove elements “badly served” or inefficient portions of the solution
- Reinsertion (Recreate)
  - greedy/construction or regret-based heuristic (complete the partial solution)
  - exact algorithm

Adaptive mechanisms

- Often there are several alternatives for implementing a component of an algorithm
- Some work better than others on some instances but work badly on others
- How “guide” the algorithm to detect the best component “automatically” (i.e. to “adapt” to the specific instance) ?

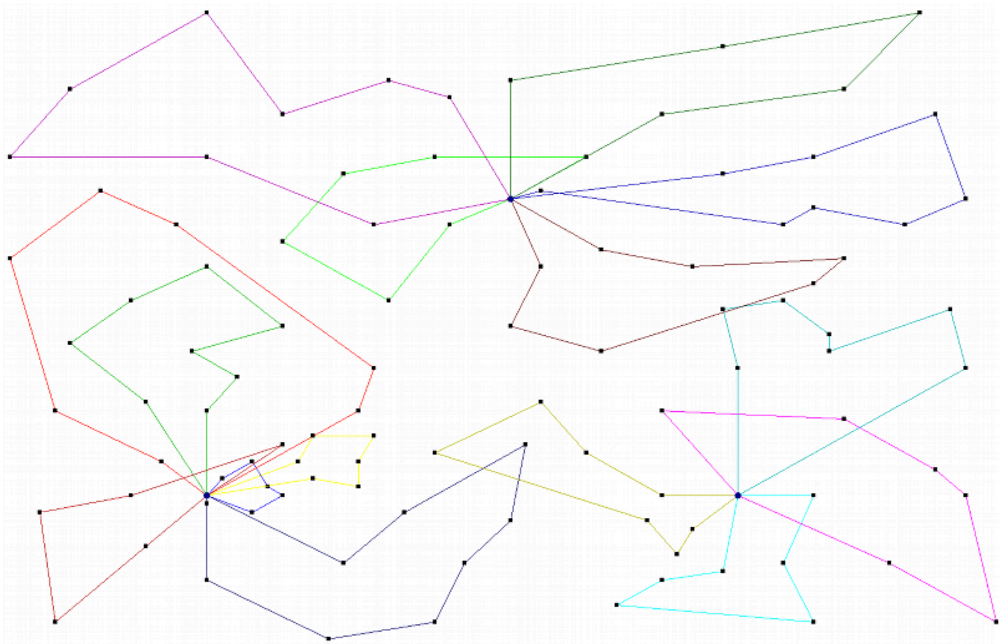
Example: Adaptive LNS

- Adaptive LNS (Pisinger & Ropke, 05)
  - Different alternatives for Removal and Insertion
  - Each may work better on specific instances
  - Initially all methods have same probability
  - At each iteration the method is selected with a probabiltity that is proportional to the effectiveness shown by the method in the previous iterations

Adaptive VNS

- In some problems a totally random shaking can produce very bad solutions
- The local search returns to the initial solution

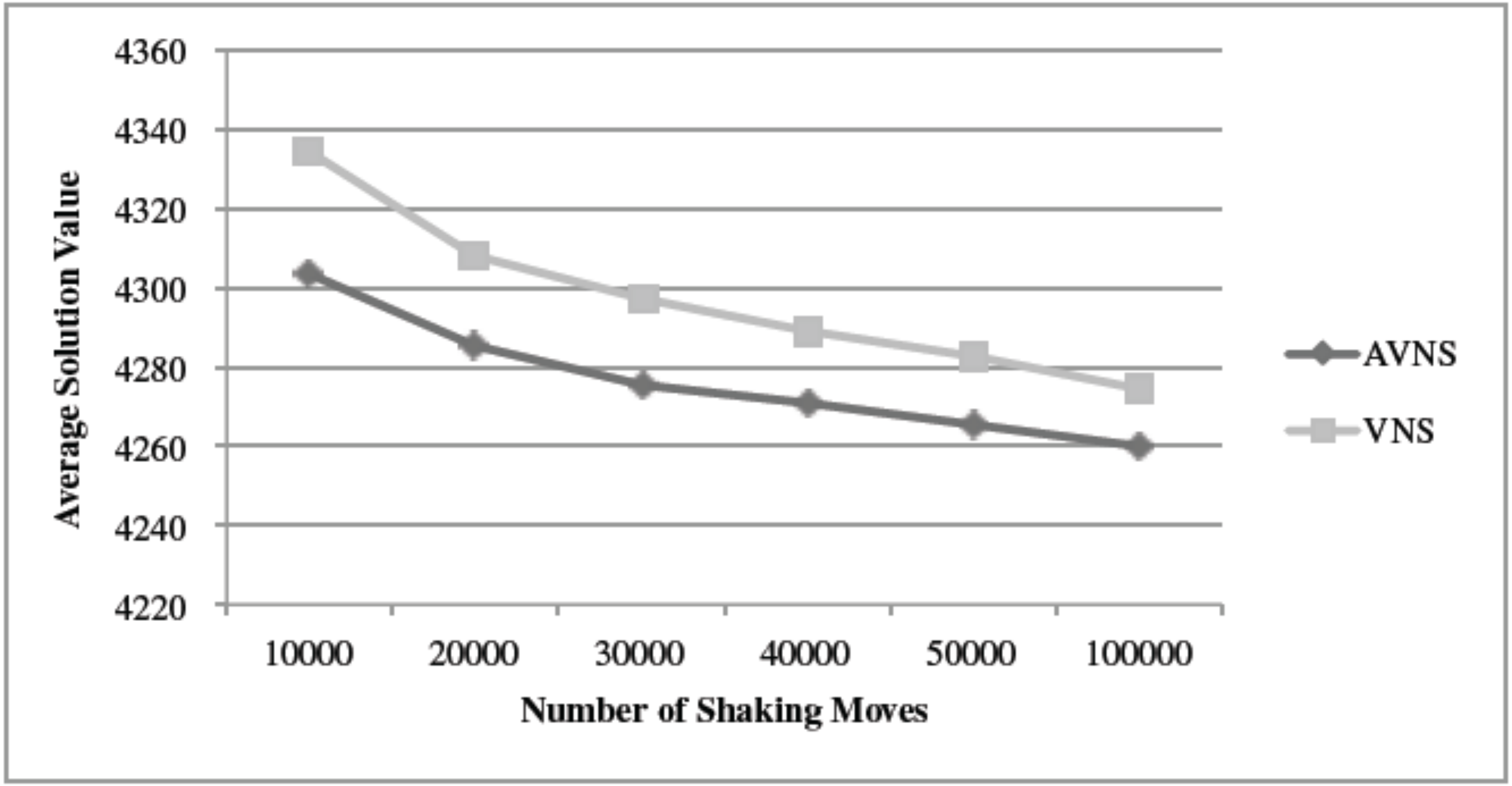
example:
multiple depot
VRP



Adaptive VNS

- Introduce some “bias” in the selection of the elements of the random move (e.g. the involved routes must be “close”)
- (Stenger et al. 2012) Several mechanisms for selecting the routes and the customers involved in the shaking
- Adaptive selection of the “best performing” mechanisms

It works !



4360
4340
4320
4300
4280
4260
4240
4220
Average Solution Value
10000
20000
30000
40000
50000
100000
Number of Shaking Moves
AVNS
VNS

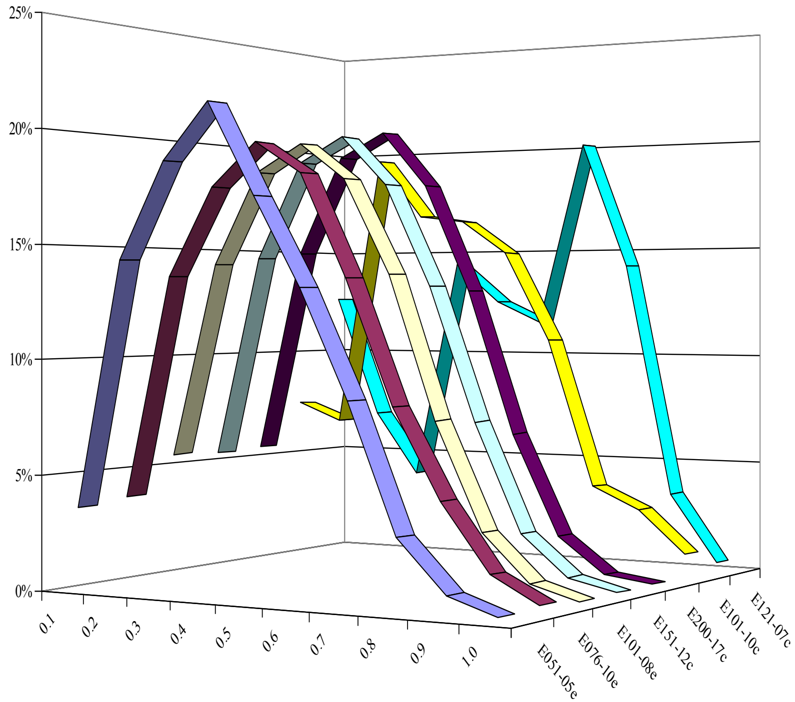
Granular Neighborhoods

- Restriction of standard neighborhoods (Toth, V., INFORMS JC, 03):
  - include and examine only few “promising” moves (e.g. linear cardinality)
  - much faster exploration without degradation in quality
- May be seen as an implementation of Candidate List concept (Glover, Laguna, 97)

Granular Neighborhoods (cont'd)

- How to define promising moves ?
  - CVRP (T&V, 2003): avoid "long" arcs ($c_{ij} > \theta$)

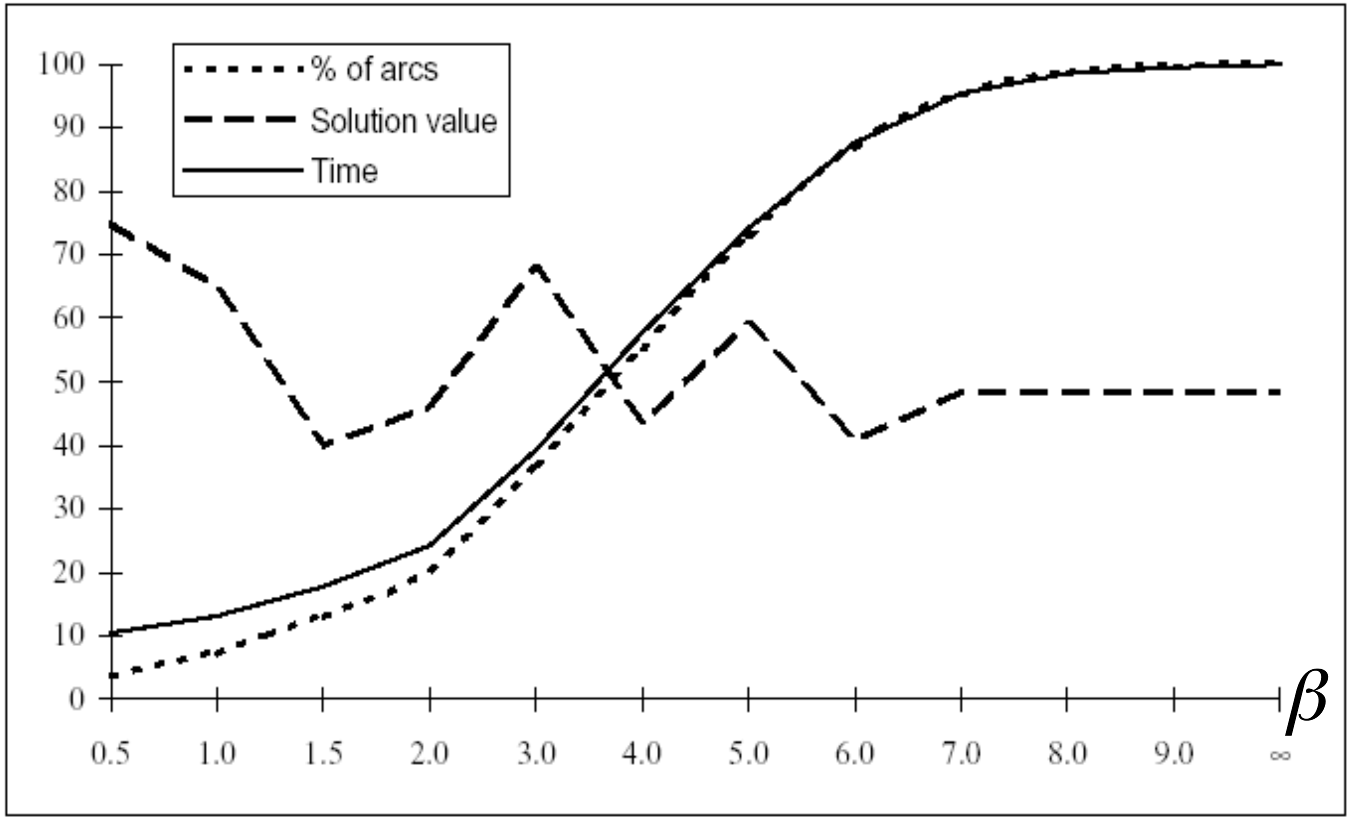
Problem	n	K	z^*	$\bar{z}^*$	$\bar{c}_{ij}$
E051-05e	50	5	524.61	9.54	33.75
E076-10e	75	10	835.26	9.83	34.13
E101-08e	100	8	826.14	7.65	34.64
E151-12c	150	12	1028.42	6.35	33.92
E200-17c	199	17	1291.45	5.98	33.24



moves inserting "long" arcs are avoided

Granular Neighborhoods (cont'd)

$$\theta = \beta \cdot UB/(n+K)$$



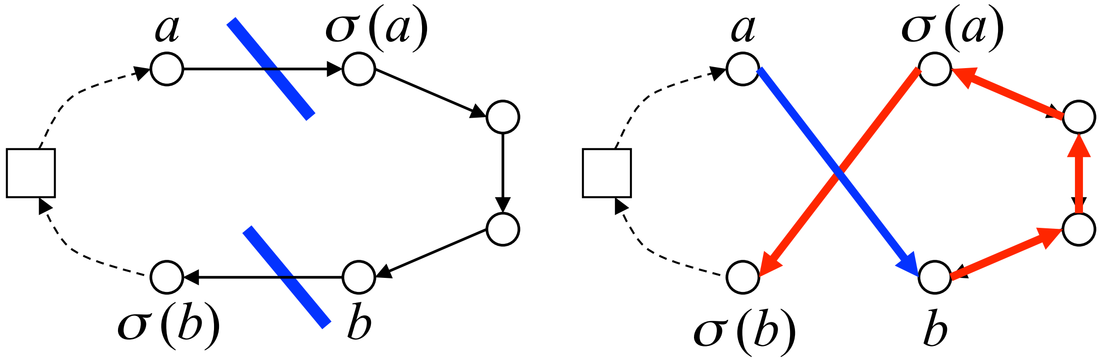
Granular N. for CVRP

- Given θ (Granularity threshold), define:

  $A' = \{(i, j) \in A: c'_{ij} \leq \theta\} \cup L$, with $|A'| = m << n^2$

  where L includes relevant arcs:

  - incident into the Depot, belonging to best solutions,…
- $G' = (V_0, A')$ is stored as a sparse graph
- The G.N. can be examined in $O(m)$ time:
  - each (a,b) $\in A'$ defines a unique move

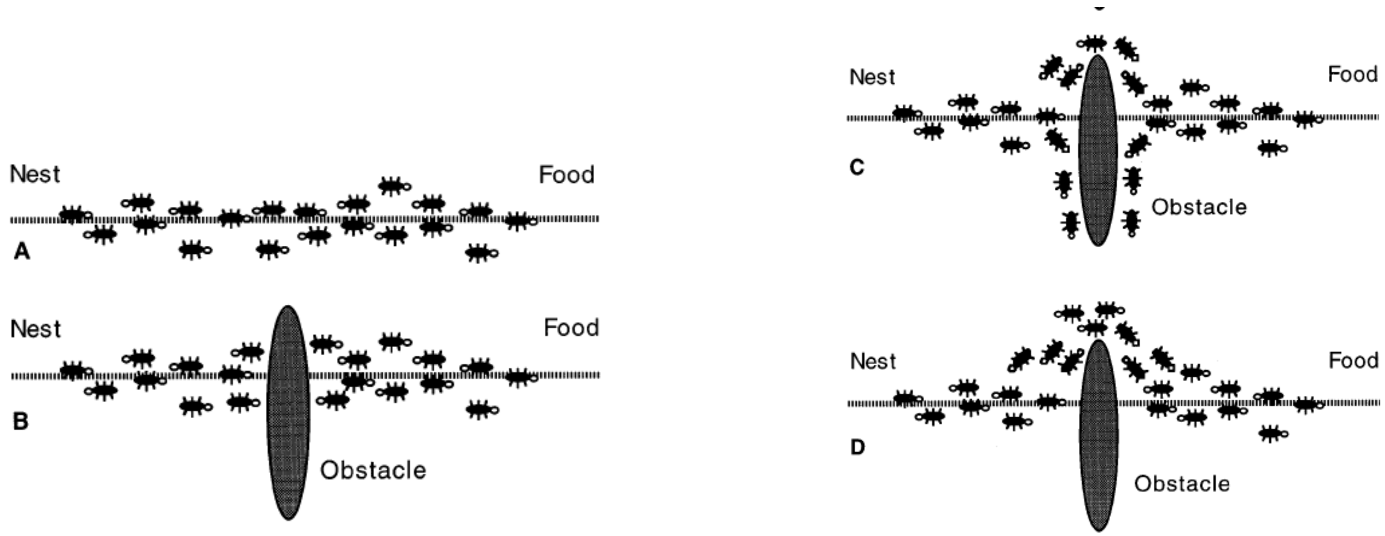


Tabu search methods results

Problem	Osman[7] (BA) f^*	Osman[7] (BA) Time[1]	Taillard[8] f^*	Taburoute[9] standard f^*	Taburoute[9] standard Time[2]	Taburoute[9] best f^*	Rochat and Taillard[10] f^*	Xu and Kelly[4,5] f^*	Xu and Kelly[4,5] Time[3]	Rego and Roucairol[11] f^*	Toth and Vigo[12] f^*	Toth and Vigo[12] Time[6]
E051-05e	**524.61**	*1.12*	**524.61**	**524.61**	*6.0*	**524.61**		**524.61**[4,5]	*29.22*[4,5]	**524.61**	**524.61**	*0.81*
E076-10e	844	*1.18*	**835.26**	835.77	*53.8*	835.32		**835.26**[4,5]	*48.80*[4,5]	835.32	838.60	*2.21*
E101-08e	835	*11.25*	**826.14**	829.45	*18.4*	**826.14**		**826.14**[4,5]	*71.93*[4,5]	827.53	828.56	*2.39*
E101-10c	819.59	*6.79*	**819.56**	**819.56**	*16.0*	**819.56**		**819.56**[4,5]	*56.61*[4,5]	**819.56**	**819.56**	*1.10*
E121-07c	**1042.11**	*23.31*	**1042.11**	1073.47	*22.2*	**1042.11**		**1042.11**[4,5]	*91.23*[4,5]	**1042.11**	1042.87	*3.18*
E151-12c	1052	*51.25*	**1028.42**	1036.16	*58.8*	1031.07		1029.56[4,5]	*149.90*[4,5]	1044.35	1033.21	*4.51*
E200-17c	1354	*32.88*	1298.79	1322.65	*90.9*	1311.35	**1291.45**	1298.58[4,5]	*272.52*[4,5]	1334.55	1318.25	*7.50*
D051-06c	**555.44**	*2.34*	**555.43**	**555.43**	*13.5*	**555.43**		**555.43**[5]	*30.67*[5]	**555.43**	**555.43**	*0.86*
D076-11c	913	*3.38*	**909.68**	913.23	*54.6*	**909.68**		965.62[5]	*102.13*[5]	**909.68**	920.72	*2.75*
D101-09c	866.75	*20.00*	**865.94**	**865.94**	*25.6*	**865.94**		881.38[5]	*98.15*[5]	866.75	869.48	*2.90*
D101-11c	**866.37**	*92.98*	**866.37**	**866.37**	*65.7*	**866.37**		915.24[5]	*152.98*[5]	**866.37**	**866.37**	*1.41*
D121-11c	1547	*22.38*	**1541.14**	1573.81	*59.2*	1545.93		1618.55[5]	*201.75*[5]	1550.17	1545.51	*9.34*
D151-14c	1188	*40.73*	**1162.55**	1177.76	*71.0*	1162.89		No solution	*168.08*[5]	1164.12	1173.12	*5.67*
D200-18c	1422	*55.17*	1397.94	1418.51	*99.8*	1404.75	**1395.85**	1439.29[5]	*368.37*[5]	1420.84	1435.74	*9.11*
E inst.	+1.32%		+0.08%	+0.95%				+0.09%		+0.73%	+0.47%	

Ant systems

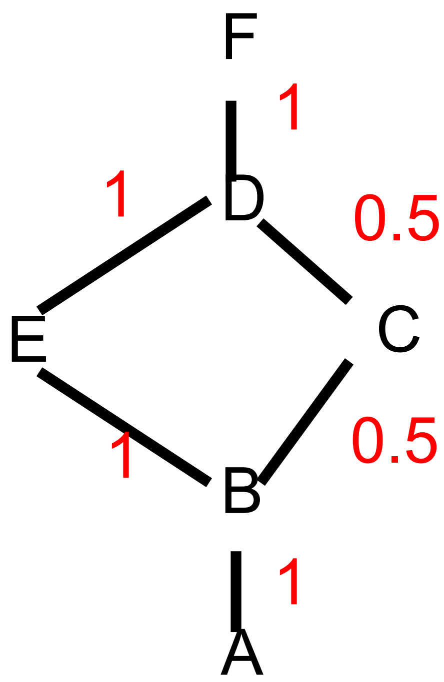
- Inspired by the capacity of ants to optimize collectively the choice of paths to the food
- the path followed by an ant is proportional to the pheromone trace found on the trail



Ant algorithms

- Ant systems are a population based approach (Dorigo, Colorni and Maniezzo), similar to GA
- There is a population of ants, with each ant finding a solution and then communicating with the other ants
  - Time, t, is discrete
  - At each time unit an ant moves a distance, d, of 1
  - Once an ant has moved it lays down 1 unit of pheromone
  - At t=0, there is no pheromone on any edge

Ant Algorithms



16 ants are moving from A - F and another 16 are moving from F - A

At t=1 there will be 16 ants at B and 16 ants at D.

At t=2 there will be 8 ants at D and 8 ants at B. There will be 16 ants at E

The intensities on the edges will be as follows

FD = 16, AB = 16, BE = 8, ED = 8, BC = 16 and CD = 16

Exploration

- We are interested in exploring the search space, rather than simply plotting a route
- We need to allow the ants to explore paths and follow the best paths with some *probability* in proportion to the intensity of the pheromone trail
- We do not want them simply to follow the route with the highest amount of pheromone on it, else our search will quickly settle on a sub-optimal (and probably very sub-optimal) solution
- The probability of an ant following a certain route is a function, not only of the pheromone intensity but also a function of what the ant can see (***visibility***)
- The pheromone trail must not build unbounded. Therefore, we need "evaporation"

Ants for TSP

- At the start of the algorithm one ant is placed in each city
- When an ant decides which town to move to next, it does so with a probability that is based on the distance to that city and the amount of trail intensity on the connecting edge
- The distance to the next town, is known as the *visibility*, n_{ij}, and is defined as $1/d_{ij}$, where, d, is the distance between cities i and j.

Ants for TSP

- In order to stop ants visiting the same city in the same tour a data structure, Tabu, is maintained
- This stops ants visiting cities they have previously visited
- $Tabu_k$ is defined as the list for the k^{th} ant and it holds the cities that have already been visited

Ants for TSP

- After each ant tour the trail intensity on each edge is updated using the following formula

$$T_{ij}\,(t + n) = p\ .\ T_{ij}(t) + sum_k\ \Delta T_{ij}^{k}$$

$$\Delta T_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \begin{array}{c} if\ the\ kth\ ant\ uses\ edge(i,j)\ in\ its\ tour \\ (between\ time\ t\ and\ t+n) \end{array} \\ 0 & otherwise \end{cases}$$

- Q is a constant and L_k is the tour length of the k^{th} ant

Ants for TSP

- Transition Probability

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[T_{ij}(t)]^{\alpha} \cdot [n_{ij}]^{\beta}}{\sum k \in allowed_k \quad [T_{ik}(t)]^{\alpha} \cdot [n_{ik}]^{\beta}} & if\ j \in allowed_k \\ 0 & otherwise \end{cases}$$

- where α and β are control parameters that control the relative importance of trail versus visibility

Ant Algorithms

- If you are interested (and willing to do some work) there is a spreadsheet on the web site that implements some of the above formula

- The spreadsheet was developed by myself simply as means of being able to cross check values whilst I developed an ant algorithm

							Numerator	Denominator
Move A to A	TRUE	1.73	0.00	Visibility A to A	1.00	0.00	1.73	0.00
Move A to A	FALSE	1.73	1.73	Visibility A to A	1.00	1.00	1.73	1.73
Move A to B	FALSE	1.73	1.73	Visibility A to B	0.89	0.89	1.55	1.55
Move A to B	FALSE	1.73	1.73	Visibility A to B	0.89	0.89	1.55	1.55
Move A to C	FALSE	1.73	1.73	Visibility A to C	0.93	0.93	1.62	1.62
Move A to D	FALSE	1.73	1.73	Visibility A to D	0.99	0.99	1.72	1.72
Move A to E	FALSE	1.73	1.73	Visibility A to E	0.77	0.77	1.33	1.33
Move A to F	FALSE	1.73	1.73	Visibility A to F	0.79	0.79	1.37	1.37
Move A to F	FALSE	1.73	1.73	Visibility A to F	0.79	0.79	1.37	1.37
Move A to G	FALSE	1.73	1.73	Visibility A to G	0.88	0.88	1.52	1.52
Move A to H	FALSE	1.73	1.73	Visibility A to H	0.98	0.98	1.70	1.70
Move A to H	FALSE	1.73	1.73	Visibility A to H	0.98	0.98	1.70	1.70
Move A to I	FALSE	1.73	1.73	Visibility A to I	0.97	0.97	1.69	1.69
SUM's		22.52	20.78		11.88	10.88	20.58	18.8514898738

Probability A to A	0.00000
Probability A to A	0.09188
Probability A to B	0.08218
Probability A to B	0.08218
Probability A to C	0.08570
Probability A to D	0.09142
Probability A to E	0.07057
Probability A to F	0.07293
Probability A to F	0.07293
Probability A to G	0.08062
Probability A to H	0.09002
Probability A to H	0.09002
Probability A to I	0.08955

This spreadsheet models the transition probability shown in the paper [ref12]
See notes, if necessary

Distance Table

	A	A	B	B	C	D	E	F
A	1.00							
A	1.00							
B	0.80							
B	0.80							
C	0.8[illegible]							
D	0.99							
E	0.59							
F	0.63							
F	0.63							
G	0.77							
H	0.96							
H	0.96							
I	0.95							

Trail Edge Table

	A	A	B	B	C	D	E	F
A	3.00							
A	3.00							
B	3.00							
B	3.00							
C	3.00							
D	3.00							
E	3.00							
F	3.00							
F	3.00							
G	3.00							
H	3.00							
H	3.00							
I	3.00							

Trail Edge Constant
0.5

Visibility Constant
0.5