Introduction to Metaheuristics

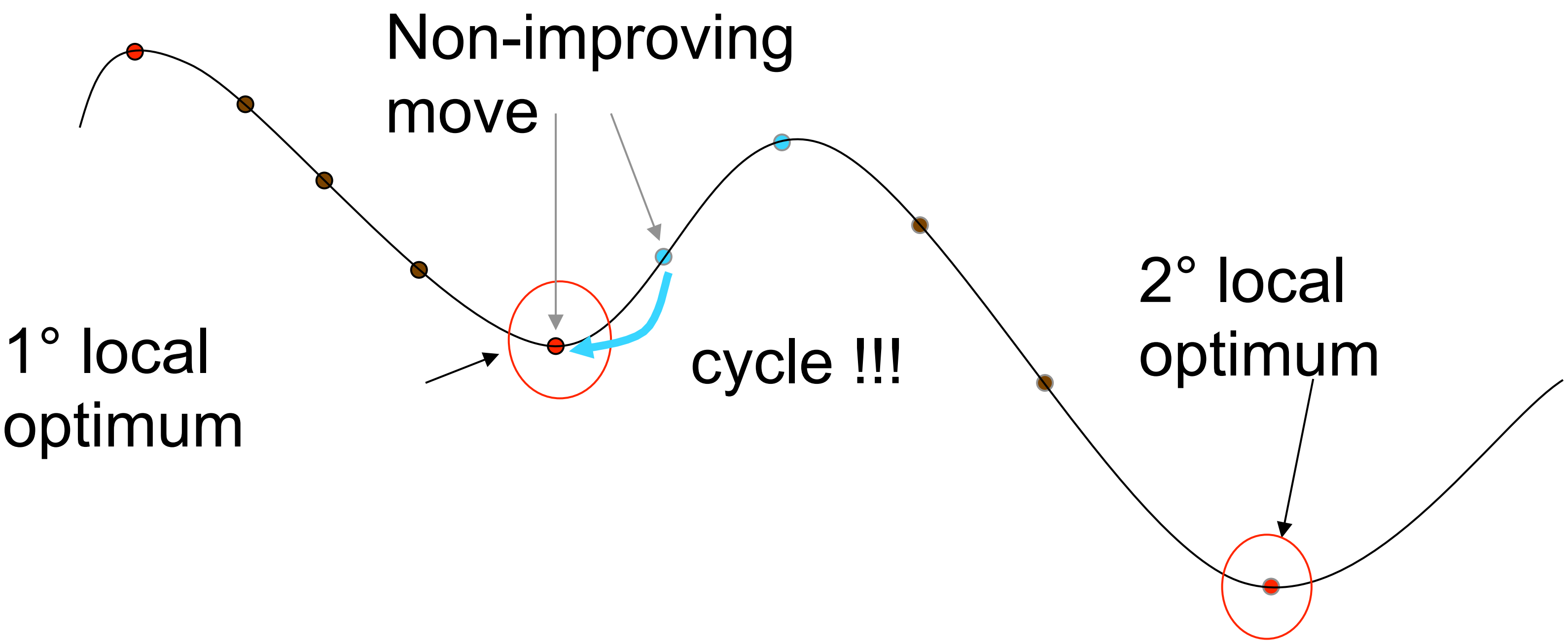
Daniele Vigo

DEI – University of Bologna

daniele.vigo@unibo.it

Metaheuristics

- Local Search algorithms which use special techniques to escape local optima found
- Must avoid cycling



Metaheuristic Algorithms

- Iterative heuristic approaches based on Local Search procedures.
  - They frequently include randomized steps.
  - At any iteration, let x be the current solution and N(x) the corresponding "neighborhood".
- The main goal of the Metaheuristic Algorithms is to "escape" from the current "local optimum" (best solution within the current neighborhood) so as to explore a larger portion of the feasibility region.

Metaheuristics

- From the 80s several Metaheuristics paradigms were proposed:
  - Simulated Annealing
  - Tabu search
  - Genetic Algorithms
  - Neural Networks
  - Ant Systems …
- In general very effective and simple to implement but much more time consuming than construction heuristics

Tabu Search (Glover, 1986)

- Generalization of local search where acceptance of worsening "moves" is allowed
- Use of short term memory (Tabu List) to avoid returning on the last *t* visited solutions
- *T* tabu list := memorize the last *t* visited solutions
- $T = \{x_{k\text{-}1},\ x_{k\text{-}2},\ \ldots,\ x_{k\text{-}t}\}$

Tabu Search: Base algorithm

```
/* for minimization problem */
generate an initial solution s whose value is z(s)
s* = s ;  k := 1 ;      T = {s} ;
while not STOP CRITERION do
    generate neighborhood  N(s)\T   /* non tabu */
    find the best solution s' ∈ N(s)\T wrt z(·)
    if   z(s') < z (s*)   then s* := s' ;   k_best := k
    s := s'
    k := k+1
    insert s'  in T replacing the "oldest" solution
endwhile
```

Tabu Search: termination criteria

- Possible termination criteria :
  - $N(s) \setminus T = \emptyset$
  - $k > k_{max}$
  - time limit reached
  - $k - k_{best} > k_{non\ improving}$
  - s* is optimal (e.g. = to a lower bound)

Tabu list

- T avoids occurrence of cycles with length $\leq |T|$
- store in T complete solutions may be too time consuming
- ex. TSP
  - each solution is a vector with n elements ;
  - compare two solutions costs $O(n)$;
  - verify if a solution is tabu costs $O(n \cdot |T|)$

Tabu list (2)

- Alternative: store "attributes" of the solutions and not entire solutions (for example moves)
- move m: set of the elementary operations to obtain a solution s' from the current one s

  $s' = s \oplus m$ (ex. arc exchanges).
- $N(s) := \{s' : \exists\, m \text{ such that } s' = s \oplus m\}$

Reverse moves

- *Tl* stores the reverse moves of the last executed moves
  - TSP 1: if the last move has moved vertex *i* from the 5° to the 7° position of the circuit then it is forbidden to bring back i to 5° position
  - TSP 2: if the last move has exchanged arcs (i, s(i)) and (*j*, *s*(*j*)) with (*i*, *j*), (*s*(*i*), *s*(*j*)) the moves that involve vertexs *i* or *j* are forbidden
  - KP-01: if the last move has inserted item i in the Knapsack it is forbidden to remove it

Limits of reverse moves

- *TI* much more restrictive than *T*
  - $R := \{x' : \exists\ m \in TI \text{ such that } x' = x \oplus m\}$
  - $|N(x)\setminus R| \leq |N(x)\setminus T\,|$ (often <<)

  - Ex. consider all possible triples of distinct elements of the set {a,b,c,d,e}

Current solution	*move*	*Tabu List*
abc	$c \to d$	$d \to c$
abd	$b \to c$	$d \to c$ $c \to b$
acd	$d \to b$	$d \to c$ $c \to b$ $b \to d$
acb $=$ abc		

Limits of the reverse moves

- *TI* does not guarantee to avoid cycles having period ≤ |*TI*|
- Ex. TSP: *TI* stores the last moved vertexs
  - |*TI*| > 2
    - 1 2 3 4 5 6
    - 1 3 2 4 5 6
  - *TI* = {3}
    - 1 3 2 4 5 6
    - 1 2 3 4 5 6
- cycle with period 1 !

Aspiration criteria

- Technique used to overcome the too restrictive conditions of the reversed moves list
- A move even if tabu can be anyway executed if it produces a “good solution”

Intensification and diversification

- TI is the short term memory
- An effective search requires also a mid/long term memory that allows for:
  - intensification of the search: it is not permitted to move too much from the area of solution space that is currently visited
    - favor moves which have common characteristics with a good solution recently encountered
    - penalty to be added to z(·) for the moves altering such characteristic

Intensification and diversification

- diversification of the search: to move towards unexplored areas of the solutions space
  - penalty to be added to $z(\cdot)$ for the solutions too “close” to the current one
  - penalty to the most frequently performed moves (“long term memory”)

Tabu search: constraints

- Sometimes the problem is too constrained:
  - the cardinality of N(s) can be very small (it is easy that N(x)\T = Ø)
  - some constraints can be relaxed (removed) by adding to z(x) a penalty proportional to theviolation of the constraints in x
  - the search moves also through infeasible solutions
  - Adaptive adjustment of the penalty:
    - increase if recently all infeasible solution are found
    - decrease if recently all feasible solutions are found

Tabu list management

- Tabu tenure t: length of the tabu list
  - constant = ...
  - updated dynamically (Ex. every h iterations)
    - if s* was improved $\Rightarrow$ $t := \max\{t_{min}, \ t-1\}$ (INTENS.)
    - if s* was not changed $\Rightarrow$ $t := \min\{t_{max}, t+1\}$ (DIVERSIFICATION)
  - randomly chosen in $[t_{min}, \ t_{max}]$
  - which values for t, t_{min}, t_{max}, h ???
- How to memorize the tabu list
  - memory vs time

Tabu list for TSP

- It is tabu moving vertex i for t iterations
- Stores the list tabu vertexs
  - T = {i, j, ...} ➔ space O(t),   time O(t)
- Store the iteration in which vertex i is moved
  - $\forall$ vertex i $\Rightarrow$ T(i) := iteration in which i is moved

k = current iteration
if $k \leq T(i) + t$
then *move is tabu*

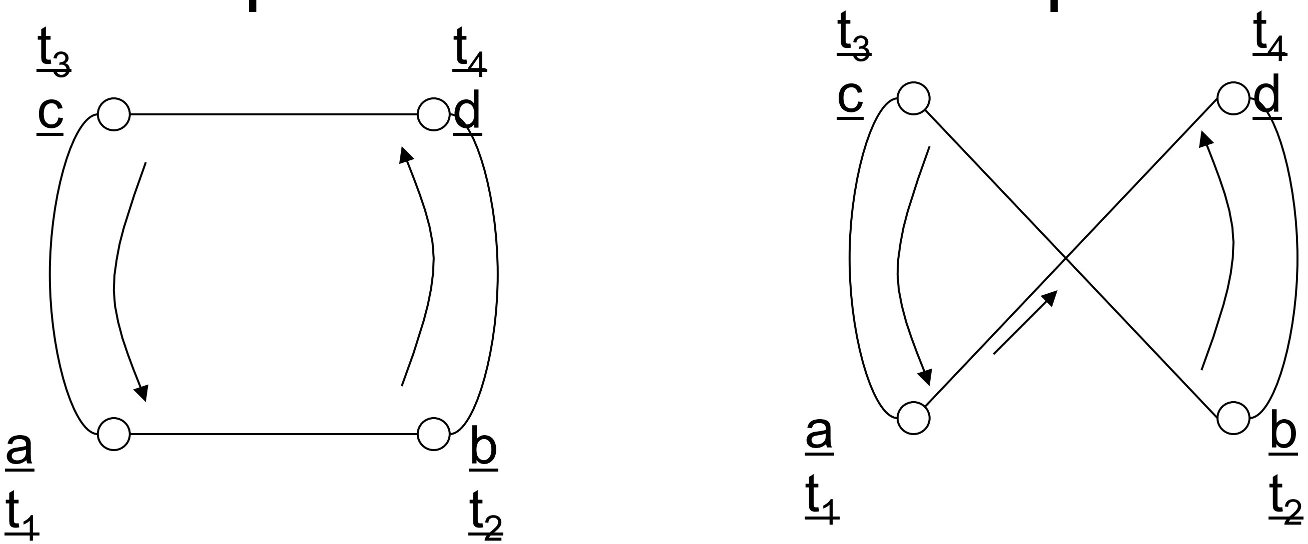
➔ space O(n) but time O(1)

Basic Tabu search for TSP

- Glover 86, Knox and Glover 89, Knox 89-94
- Neighborhood 2-opt exchange $\Rightarrow O(n^2)$
- Tabu list :
  - G 86 : T stores the shortest edge removed by an exchange ➔ cannot be reinserted
    - Aspiration level ➔ improving tabu moves are accepted
  - K 94 : T stores the pairs of longest removed arcs (less restrictive) ➔ cannot be reinserted
    - Aspiration level ➔ tabu moves are accepted when the cost is reduced wrt when the arcs were present

Basic tabu search for TSP

- Efficient implementation of 2/3-opt



- every exchange is represented by a 4-tuple

  $< t_1, \; t_2, \; t_3, \; t_4 >$

  - remove $(t_1, \; t_2), \; (t_4, \; t_3)$
  - insert $(t_2, \; t_3), \; (t_1, \; t_4)$

Basic tabu search for TSP (2)

- Every exchange corresponds to 2 4—tuples
  - $t_1 = a$ $t_2 = b$ < a, b, c, d >
  - $t_1 = d$ $t_2 = c$ < d, c, b, a >
- $\Delta(a,b,c,d) = c_{t1t2} + c_{t4t3} - c_{t2t3} - c_{t1t4}$
  - to have an improving exchange (> 0)
  - (a) $c_{t1t2} > c_{t2t3}$ or (b) $c_{t4t3} > c_{t1t4}$ or both
- if it is improving (a) is verified for at least one of the two representations

Basic tabu search for TSP (3)

- we don't miss an improving exchange if we limit ourself to 4-tuples t1, ..., t4 for which $c_{t1t2} > c_{t2t3}$
- given t1 and t2 it suffices to consider t3 and t4 such that :
  - t3 : $c_{t1t2} > c_{t2t3}$
  - t4 follows t3 in the current solution
- How to implement the algorithm efficiently?
  - $\forall$ i Li = {v1, v2, ...} list of vertices in order of distance from i $\Rightarrow$ time $O(n^2 \log n)$, space $O(n^2)$

Simulated Annealing (SA)

- Search Algorithm based on a randomized neighborhood (Kirkpatrick et al., 1983)
- Simulates the thermodynamic behaviour of re-cooking ("annealing") of solid materials (metal, glass, ...).
- When a solid material is warmed over its melting point and successively cooled, returning to the solid state, its structural properties depend on the cooling process ("cooling schedule").
- The algorithm simulates the changes in the system energy (considered as a set of particles) during the cooling process up to the "convergence" to the solid state

Simulated Annealing (SA)

- given s (current solution), a $s' \in N(s)$ is randomly generated
  - if s' is improving, then execute it
  - otherwise non-improving moves are executed with decreasing probability
  - The probability depends on a parameter T ("temperature") decreasing during execution according to a "cooling schedule"
  - intial value T_0, final value T_f
  - e.g., T is decreased by α every k iterations

Basic SA Algorithm

```
generate an initial solution s with value z(s)
s* := s
determine the starting (and ending) temperatures T=T0 (Tf)
while T > Tf do
    choose randomly a move that transforms s in s'
    Δ = z(s') – z(s)
    if Δ ≤ 0 then            /* downhill */
        s := s'
        if z(s) < z(s*) then s* = s
    else                          /* uphill */
        generates a random value r in [0,1]
        if r < e^(-Δ/T) then s := s'
    decrease T according to a cooling schedule
endwhile
```

Considerations on SA

- Non homogeneous version:
  - The temperature decreases at every (k) move
- Homogeneous version:
  - Keep the same temperature until an equilibrium status is reached, then decrease it
- final temperature: in theory T_f
  - in practice stop when
    - the optimal solution is not improved since P iterations
    - a move is not accepted since Q iterations, or time limit
- Converges to global optimum

Simulated Anealing: variants

- Reannealing
  - first run: store the termperature T° for which the best solution is found
  - second run: more accurate search with T=T°
- Restricted neighborhood
  - moves not leading to good solutions are not considered
    - Ex. TSP only moves connecting close vertices

Simulated Anealing: variants

- Record-to-Record travel
  - deterministic variant of SA (Dueck, '93)
  - let s* be the best solution so far
  - the current solution is accepted if it is within a prescribed maximum deviation Δs from s*
  - For example $\Delta s = 0.01\ s^*$

Simulated Annealing vs TS

- Differences
  - Tabu search
    - non improving moves only when a local optimum is found
    - no randomization
  - Simulated annealing
    - non improving moves always possible
    - strongly based on randomization
    - random examination of the neighborhood moving either to the first improving one or to one the pass a randomized test

Genetic algorithms

- (Holland 1975; Goldberg 1987)
- Based on the analogy with the evolution of a population of elementary individuals
  - individual ⇔ solution
  - fitness of an individual ⇔ cost of the solution
- Individuals combine to generate new ones in successive generations
  - mutation and recombination operators
- Evoution process selects the best individuals

Basic Genetic Algorithms

- INITIALIZATION: build an initial population with n individuals S={S1, ...,Sn}
- repeat /* generations */
  - MUTATION: chose m individuals in S and apply a randomized mutation to obtain m new individuals
  - RECOMBINATION (CROSSOVER): chosei r pairs of individuals and combine them in a random way to generate r new individuals that reflects the characteristics of the parents
  - SELECTION: use a selection criterion to reduce the population again to n individuals chosen from n + m + r individuals of S
- until STOP CRITERION

Inconvenients of basic GA

- The scheme is not suited for te solution of constrained problems
- Given an intial population of feasibel solutions mutation and crossover operators produce new solutions that are generally infeasible:
  - Use of specialized operators that keep feasibility
  - Use of a local search to regain feasibility and bring solutions to local optima ("send individuals to school before they start reproducing")

Genetic Local Search

- INITIALIZATION: build an initial population with n individuals S={S1, ...,Sn}
- IMPROVEMENT: determine the local optima associated with the n individuals with a Local Search Algorithm
- repeat /* generations */
  - MUTATION: chose m individuals in S and apply a randomized mutation to obtain m new individuals
  - RECOMBINATION (CROSSOVER): chosei r pairs of individuals and combine them in a random way to generate r new individuals that reflects the characteristics of the parents
  - IMPROVEMENT: apply a Local Search Algorithm to each m+r ndividuals to obtain a new set of solutions S'
  - SELECTION: use a selection criterion to reduce the population again to n individuals chosen from n + m + r individuals of S U S'
- until STOP CRITERION

Operators for GA

- In a GA the solution is represented by a string of values which are the "genes"
- Ex. in KP-01 string of binary variables associated to items
- MUTATION
  - chose one or more genes randomly (generally with uniform probability P = 1/n)
  - change the value of selected genes

Operators for GA (2)

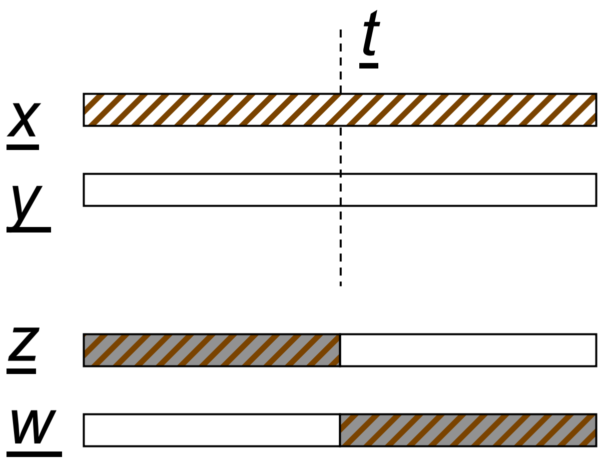
- CROSSOVER
  - Chose 2 individuals (“parents”)

    x = {x1, ..., xn} ed y = {y1, ..., yn}
  - combine them so as to create one or more new individuals (“offspring”)
  - z = {z1, ..., zn} (and w = {w1, ..., wn})
  - whose genes are a combination of the genes of the parents

CROSSOVER operators

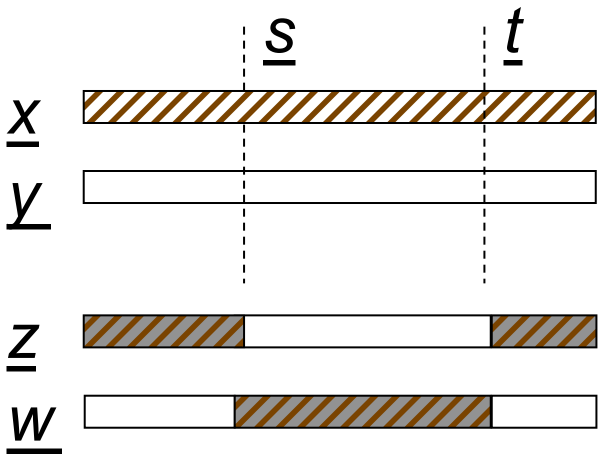
- ONE–POINT CROSSOVER (1 or 2 offspring)
  - Chose a "cutting" point t uniform in [1,n]

$$z_i = \begin{cases} x_i & i < t \\ y_i & i \geq t \end{cases} \qquad w_i = \begin{cases} y_i & i < t \\ x_i & i \geq t \end{cases}$$



CROSSOVER operators

- TWO–POINT CROSSOVER (1 or 2 offspring)
  - Chose two “cutting” point s and t uniform in [1,n]



Ex. GA for KP-01

- MUTATION: randomly change the value of one or more genes x_j of the solution

1	0	0	1	1	1	0

$\Longrightarrow$

1	1	0	1	0	1	1

- ONE-POINT CROSSOVER:

1	0	0	1	1	1	0
1	1	0	1	0	1	1

$\Longrightarrow$

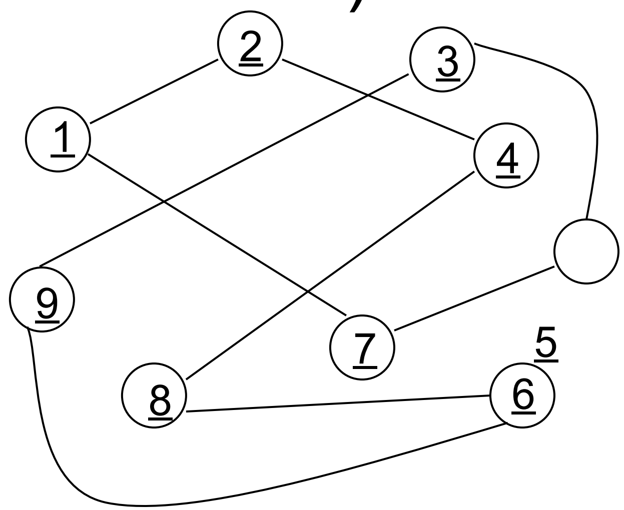
1	0	0	1	0	1	1
1	1	0	1	1	1	0

- It must be checked the feasibility of the new solutions

GA for TSP

- Representation of the solution:
  - visit sequence of the vertices (permutation)

3	5	7	1	2	4	8	6	9



  - mutation and crossover may produce infeasible solutions

- Ex. MUTATION random change of a gene

3	5	7	1	2	4	8	6	9

$\Longrightarrow$

3	5	7	1	2	6	8	6	9

Mutation variant for TSP

- New mutation operator that generates a new feasible solution
- Chose two cutting point and revert the subsequence between the two points

3	5	7	1	2	4	8	6	9

$\Longrightarrow$

3	5	4	2	1	7	8	6	9

Crossover for TSP

- if a TWO-POINT CROSSOVER is used

			s				t		
x	3	5	7	1	2	4	8	6	9
y	1	9	2	3	4	6	8	7	5

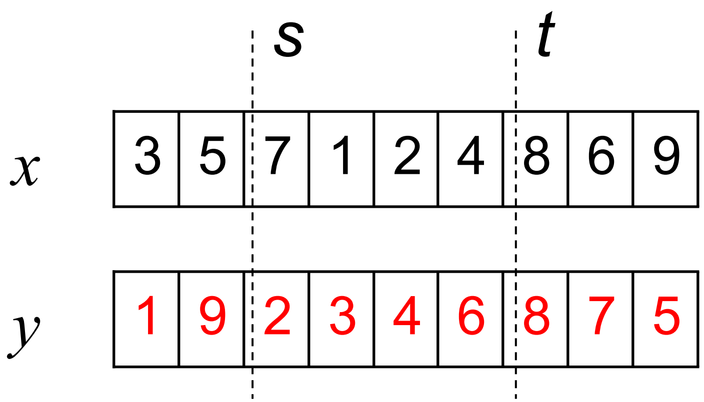
$\Rightarrow$

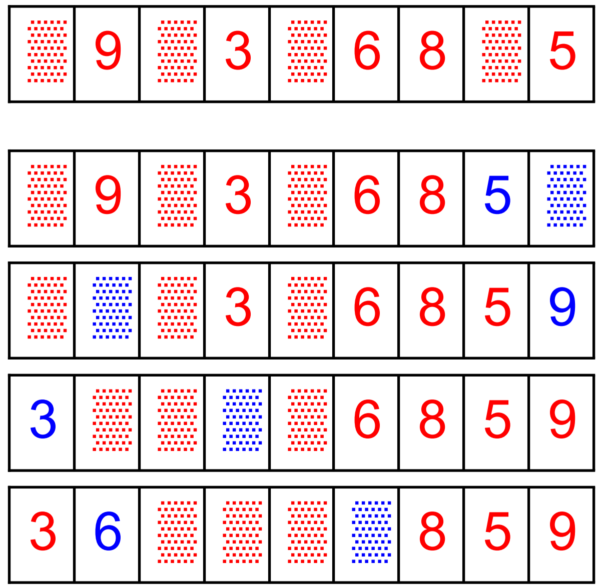
			s				t		
x'	3	5	2	3	4	6	8	6	9
y'	1	9	7	1	2	4	8	7	5

- *x*' and *y*' are NOT tours !
- solution: ORDER CROSSOVER

Order Crossover for TSP

- Chose s and t and in y replace the elements of the substrin in x with “holes”





- Move the holes left or right until they reach the central position by minimizing the pertrubation in the solution
- replace the holes with the substring of x

3	6	7	1	2	4	8	5	9

Considerations on Metaheuristics

- General algorithmic paradigms that must be specialized for specific problems
- Often non compatitive with ad hoc heuristics
- Development time much smaller than for ad hoc heuristics
- Generally much better than basic local search but with much larger computing times
- Often become complex and dependant from several parameters that are diffcult to tune

References

- VJ RaywardSmith, IH Osman, CR Reaves, GD Smith eds
  Modern Heuristics Search Methods, Wiley Chichester
- IH Osman, JP Kelly eds *MetaHeuristics Theory and Applications*
  Kluwer Academic Publishers Boston MA
- F Glover, M Laguna eds
  Tabu Search Kluwer Academic Publishers Boston MA
- E Aarts, JK Lenstra eds
  Local Search in Combinatorial Optimization Wiley Chichester
- S Voss, S Martello, I Osman, C Roucairol eds
  MetaHeuristics Advances and Trends in Local Search Algorithms for Optimization
  Kluwer Academic Publishers Boston MA
- CC Ribeiro, P Hansen eds *Essays and Surveys in Metaheuristics*
  Kluwer Academic Publishers Boston MA