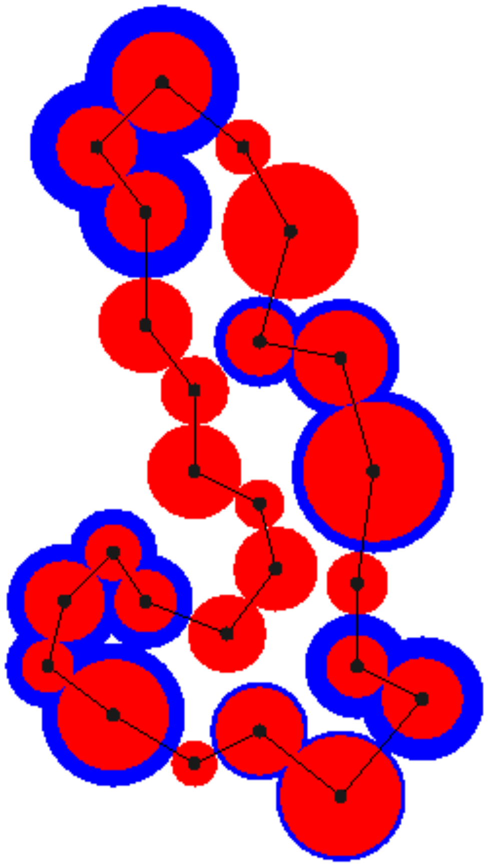
POLYTECH TOURS
Département Informatique

Multi-start local search for the traveling salesman problem

Jorge E. Mendoza
Department of Computer Science
Polytech Tours
France

The traveling salesman problem (TSP)



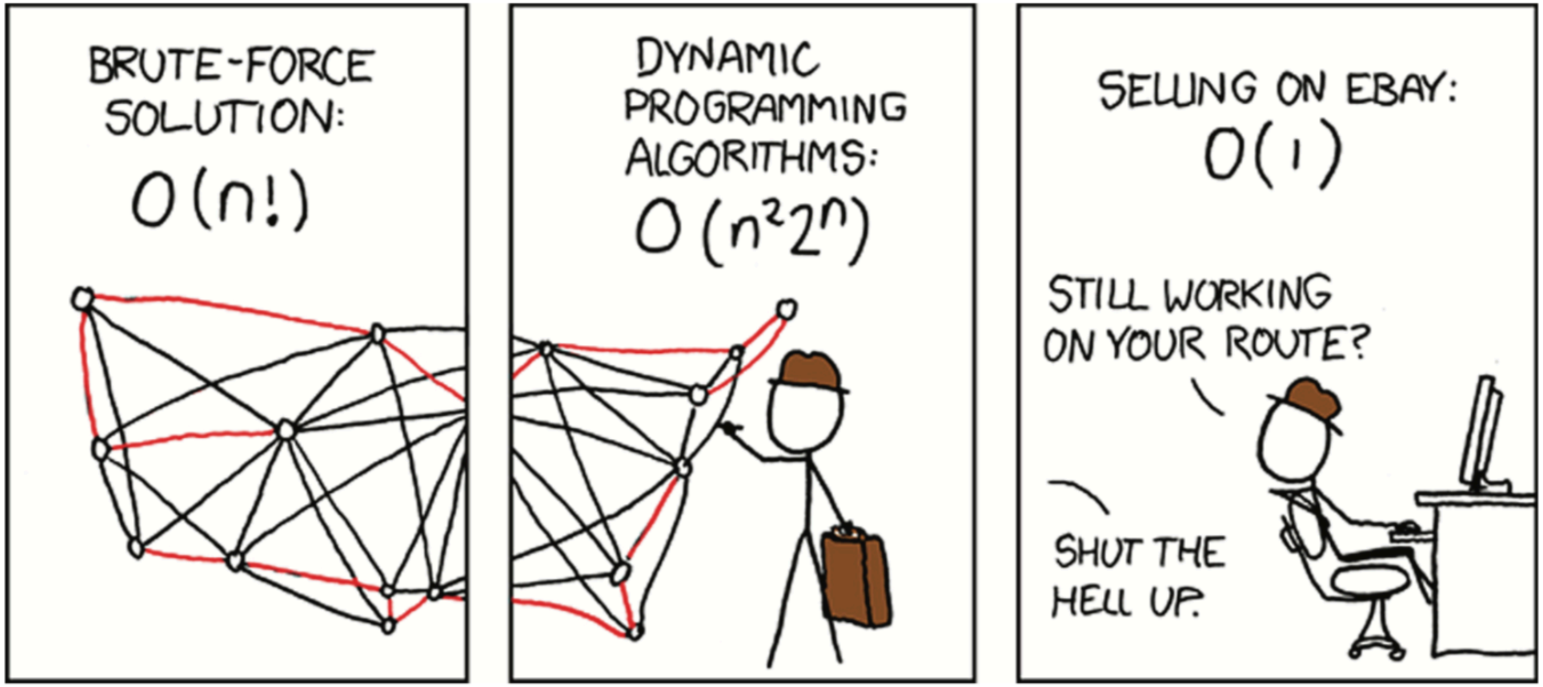
Definition

"Given a collection of cities and the cost of travel between each pair of them, the **traveling salesman problem**, or **TSP** for short, is to find the cheapest way of visiting all of the cities and returning to your starting point."

Taken from: http://www.tsp.gatech.edu/problem/index.html

Solving the TSP

- The TSP is an NP-complete problem (i.e., we do not know a "good" algorithm to solve it)

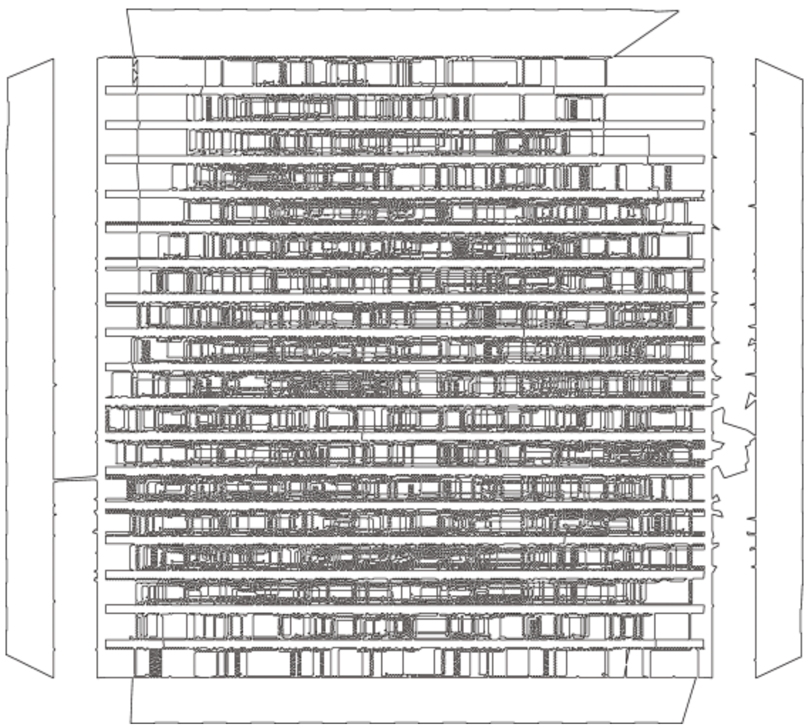


Source: xkcd.com

POLYTECH
TOURS
Département Informatique

Solving the TSP

- The TSP is an NP-complete problem (i.e., we do not know a "good" algorithm to solve it)



Largest TSP solved to optimality

85,900 cities

Solved with the Concorde algorithm in 2006

Solving the TSP

- The TSP is an NP-complete problem (i.e., we do not know a "good" algorithm to solve it)
- Solution approaches
  - Dynamic programming
  - Constraint programming
  - Constructive heuristics
  - Metaheuristics

Solving the TSP

- The TSP is an NP-complete problem (i.e., we do not know a "good" algorithm to solve it)
- Solution approaches
  - Dynamic programming
  - Constraint programming
  - Constructive heuristics
  - Metaheuristics
    - Local Search-based
    - Genetic Algorithms
    - Large Neighborhood Search

Solving the TSP

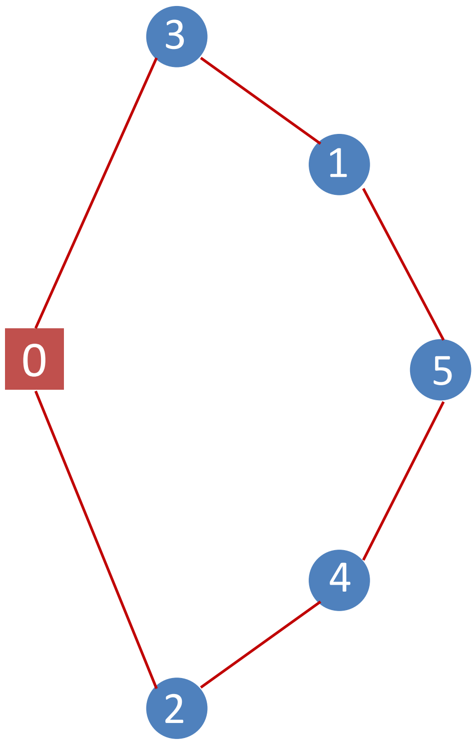
- The TSP is an NP-complete problem (i.e., we do not know a “good” algorithm to solve it)
- Solution approaches
  - Dynamic programming
  - Constraint programming
  - Constructive heuristics
  - Metaheuristics
    - Local Search-based
    - Genetic Algorithms
    - Large Neighborhood Search

Local Search

- Start from an initial solution
- Apply small changes to the solution
- Check if the solution improves
- Repeat until some stopping criterion is met
- Main “ingredients”
  - Solution representation
  - Neighborhood Scheme
  - Stopping criterion
  - Initial solution generator

Solution representation: better with an example

Solution representation for the TSP

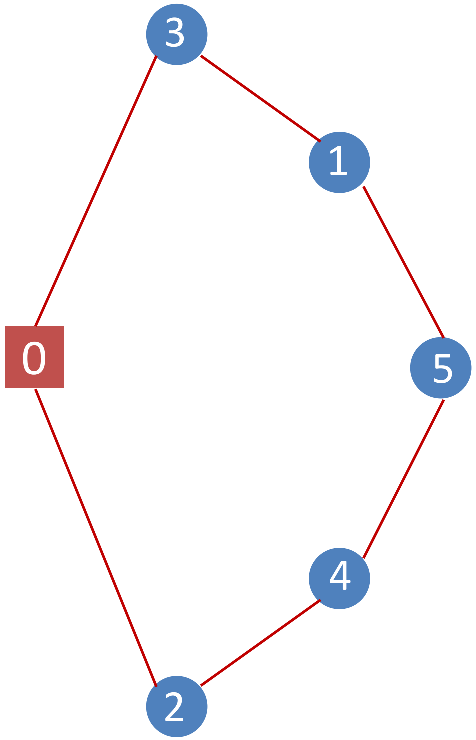


Alternative 1: the index of the positions of an array of integers indicates the visiting order of the city represented by the integer inside the position

0	1	2	3	4	5
0	3	1	5	4	2

Solution representation: better with an example

Solution representation for the TSP



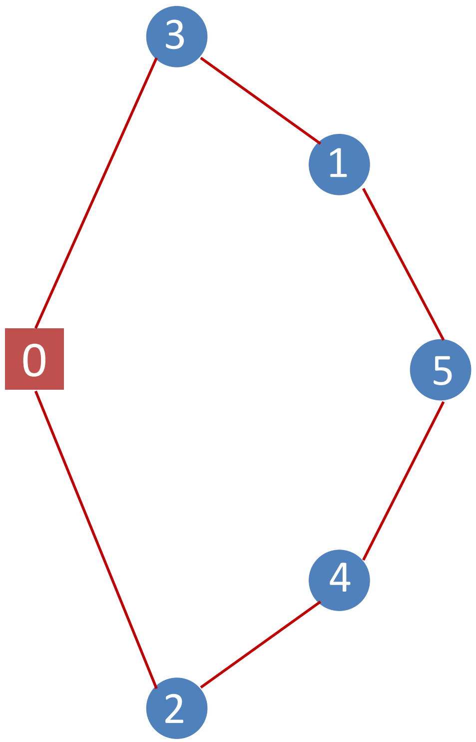
Alternative 1: the index of the positions of an array of integers indicates the visiting order of the city represented by the integer inside the position

0	1	2	3	4	5
0	3	1	5	4	2

Alternative 2: ?

Solution representation: better with an example

Solution representation for the TSP



Alternative 1: the index of the positions of an array of integers indicates the visiting order of the city represented by the integer inside the position

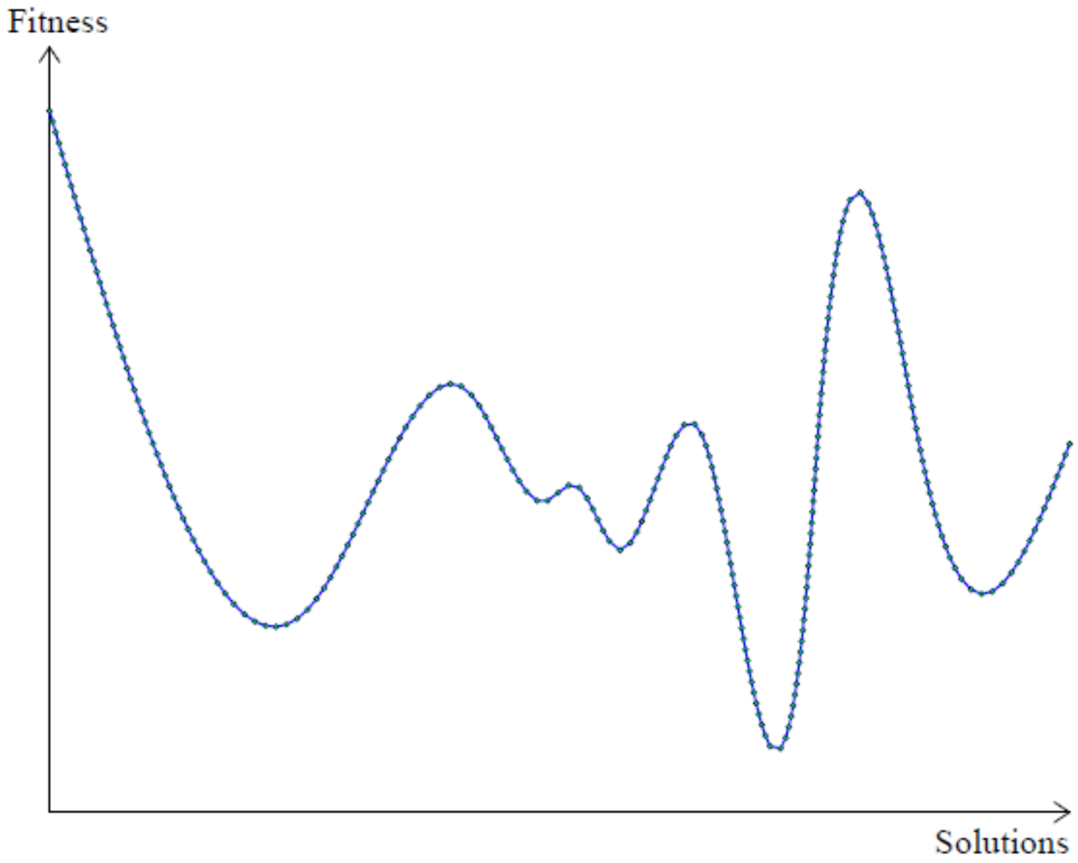
0	1	2	3	4	5
0	3	1	5	4	2

Alternative 2: the positions of the array represent the cities and the integer inside each position indicates what city comes next in the tour

0	1	2	3	4	5
3	5	0	1	2	4

Solution representation

The solution/search space



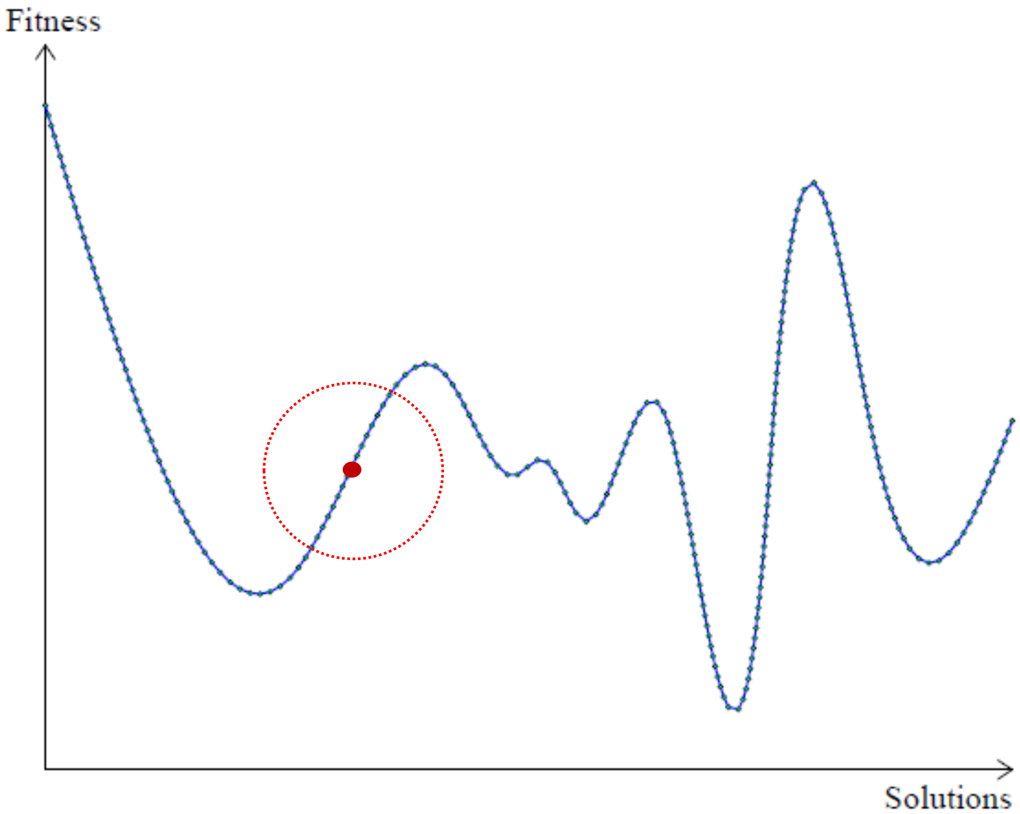
POLYTECH TOURS
Département Informatique

Solution representation

Key aspects

- **Completeness:** all solutions associated with the problem must be represented
- **Connexity:** a search path must exist between any two solutions of the search space. Any solution of the search space, especially the global optimum solution, can be attained
- **Efficiency:** the representation must be easy to manipulate by search operators. The time and space complexities of the operators dealing with the representation should be as low as possible

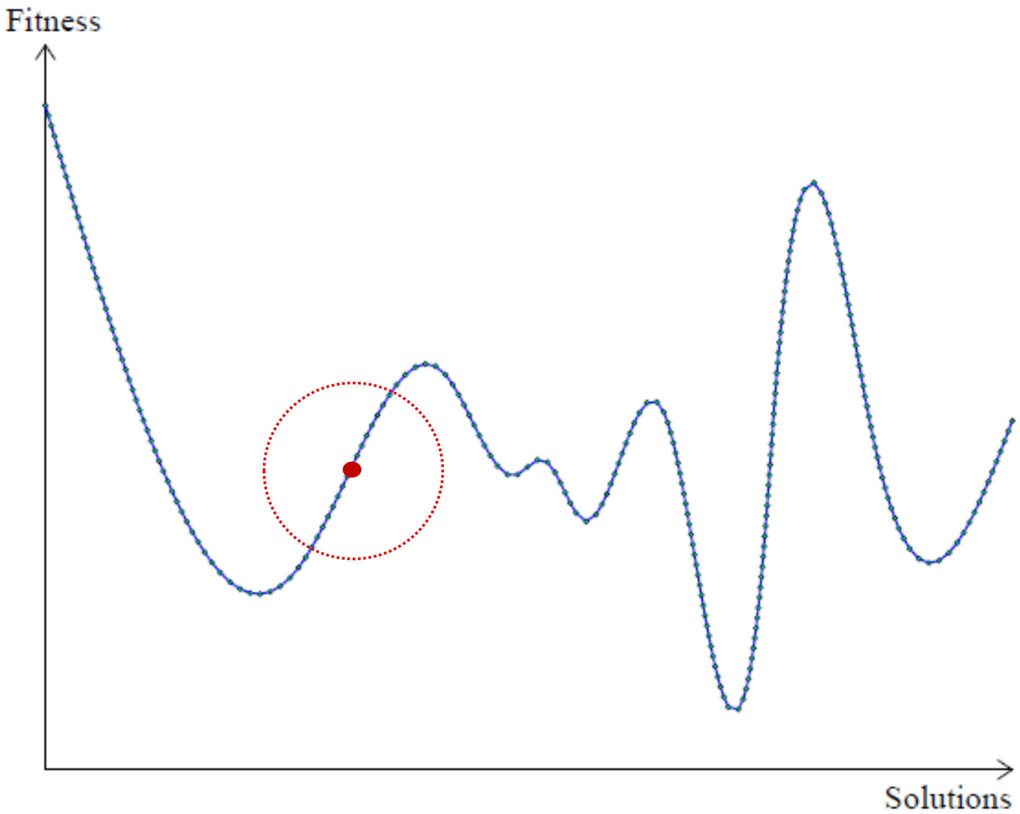
Neighborhood and neighbor solutions



Neighborhood:

Let S be the set of all solutions that form the solution space. A neighborhood function N is a mapping $N : S \rightarrow 2^S$ that assigns to each solution s of S a set of solutions $N(s) \subset S$.

Neighborhood and neighbor solutions



Neighborhood:

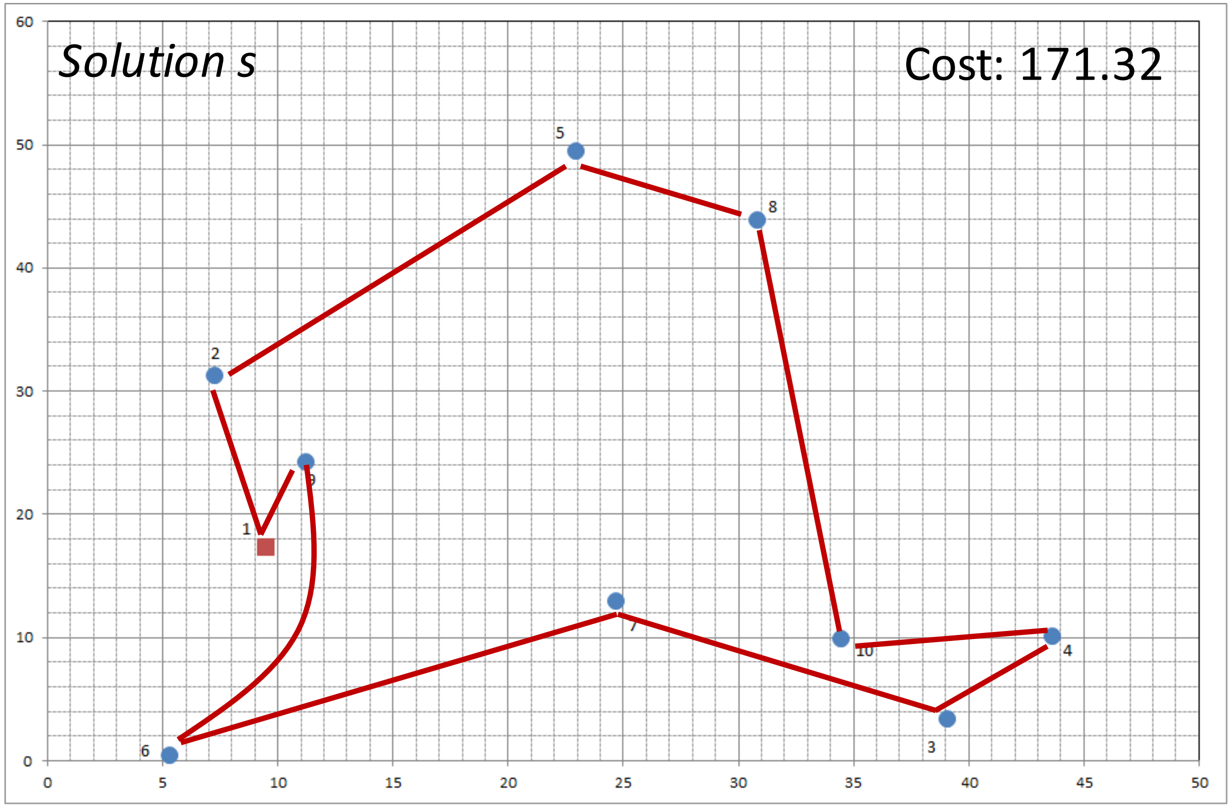
Let S be the set of all solutions that form the solution space. A neighborhood function N is a mapping $N : S \rightarrow 2^S$ that assigns to each solution s of S a set of solutions $N(s) \subset S$.

Neighbor:

A neighbor is a solution s′ in the neighborhood of s ($s' \in N(s)$).
A neighbor is generated by the application of a *move* operator m that performs a small perturbation to the solution s

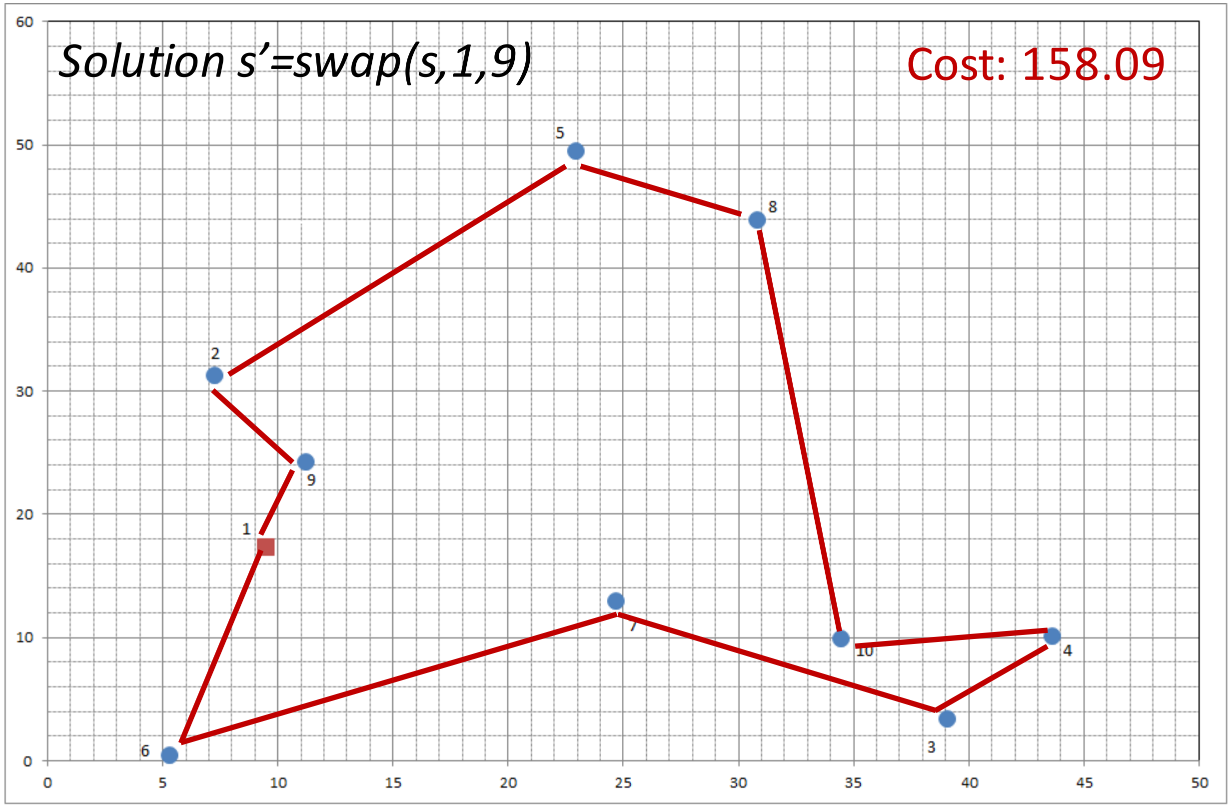
Example: moves and neighborhoods for the TSP

Swap move: given two cities, exchange their positions in the tour



Example: moves and neighborhoods for the TSP

Swap move: given two cities, exchange their positions in the tour



---

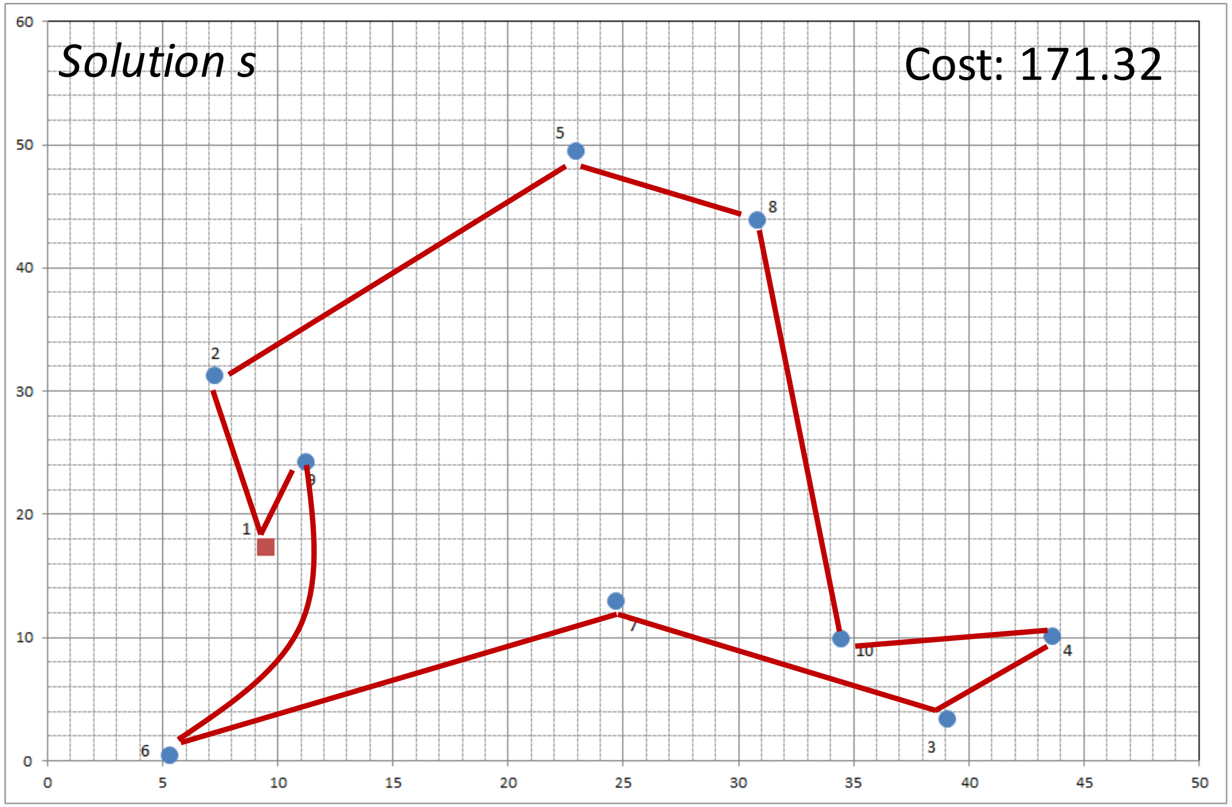
swapNeighborhood(s)

$N = \{\}$
for $i = 0$ to $n-1$ **do**
  for $j = i+1$ to $n-1$ **do**
    $s' = swap(s, i, j)$
    $N = N \cup s'$
  end for
end for
return N

POLYTECH
TOURS
Département Informatique

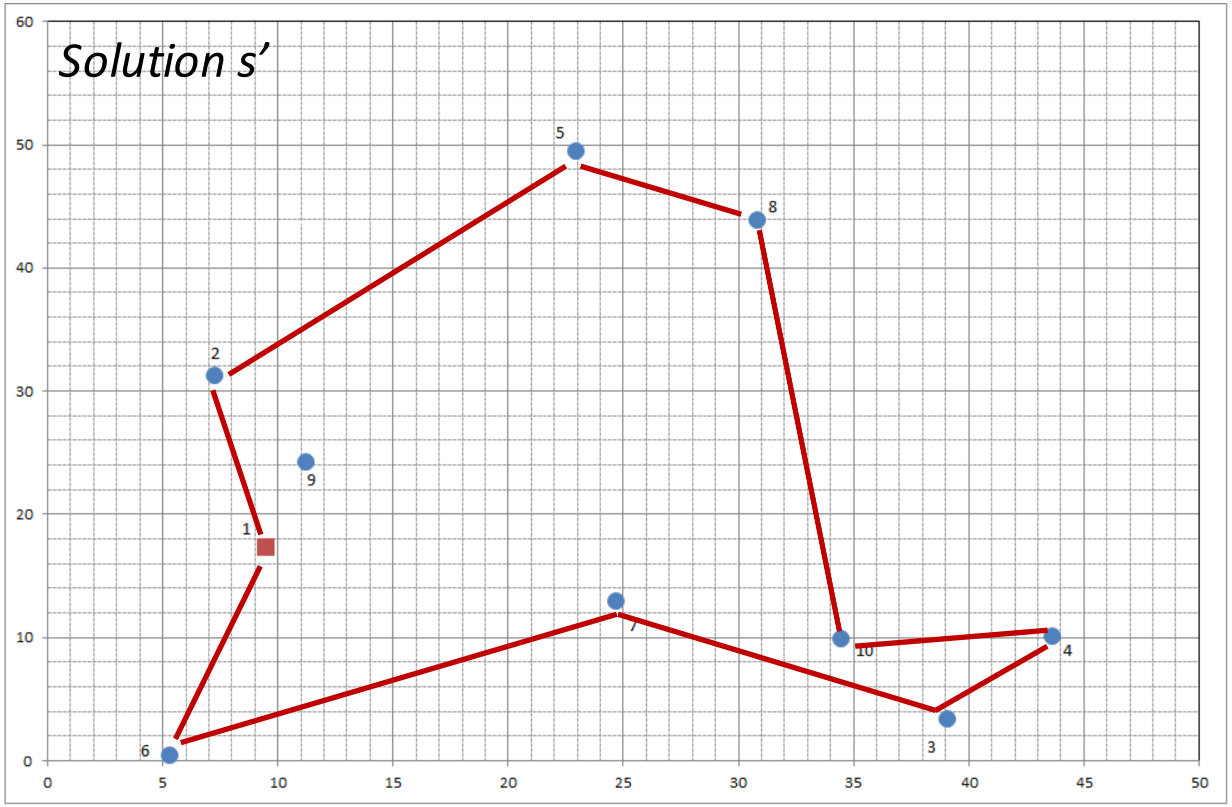
Example: moves and neighborhoods for the TSP

Re-locate move: extract a city from the tour and re-insert it on a different position



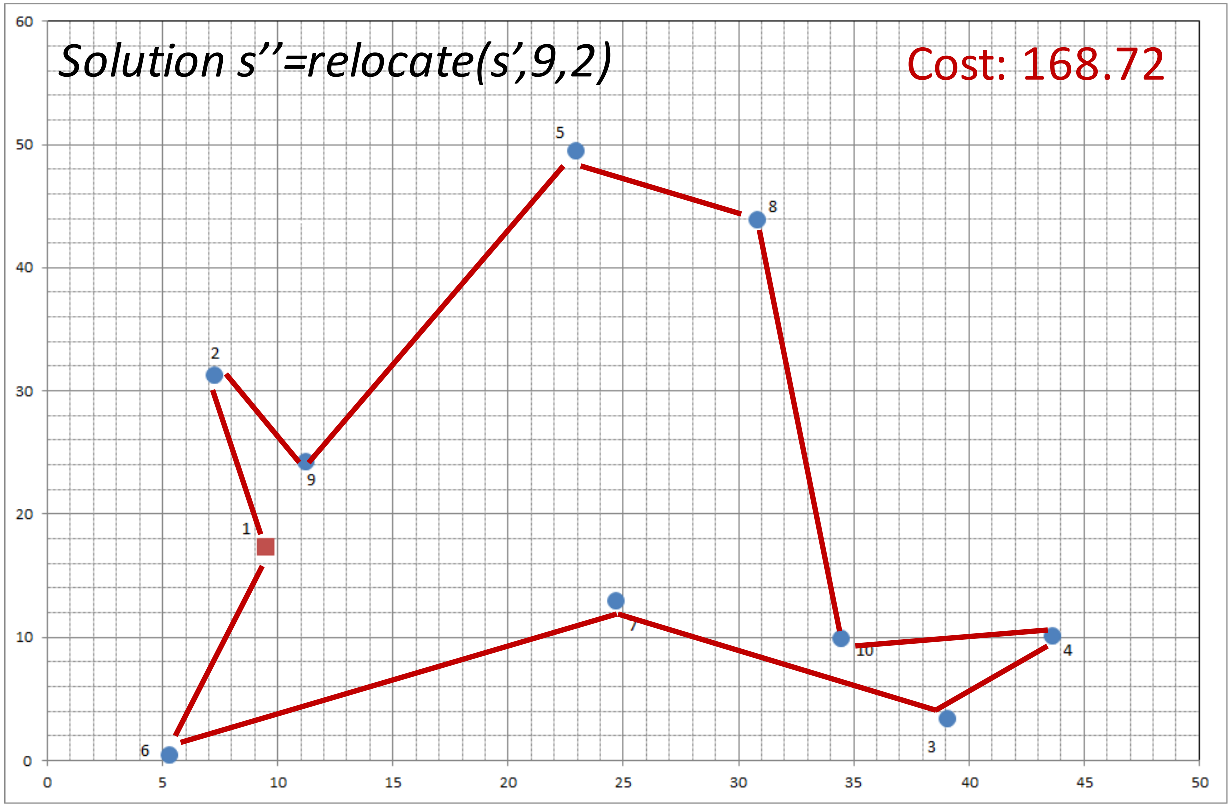
Example: moves and neighborhoods for the TSP

Re-locate move: extract a city from the tour and re-insert it on a different position



Example: moves and neighborhoods for the TSP

Re-locate move: extract a city from the tour and re-insert it on a different position



---

relocateNeighborhood(s)

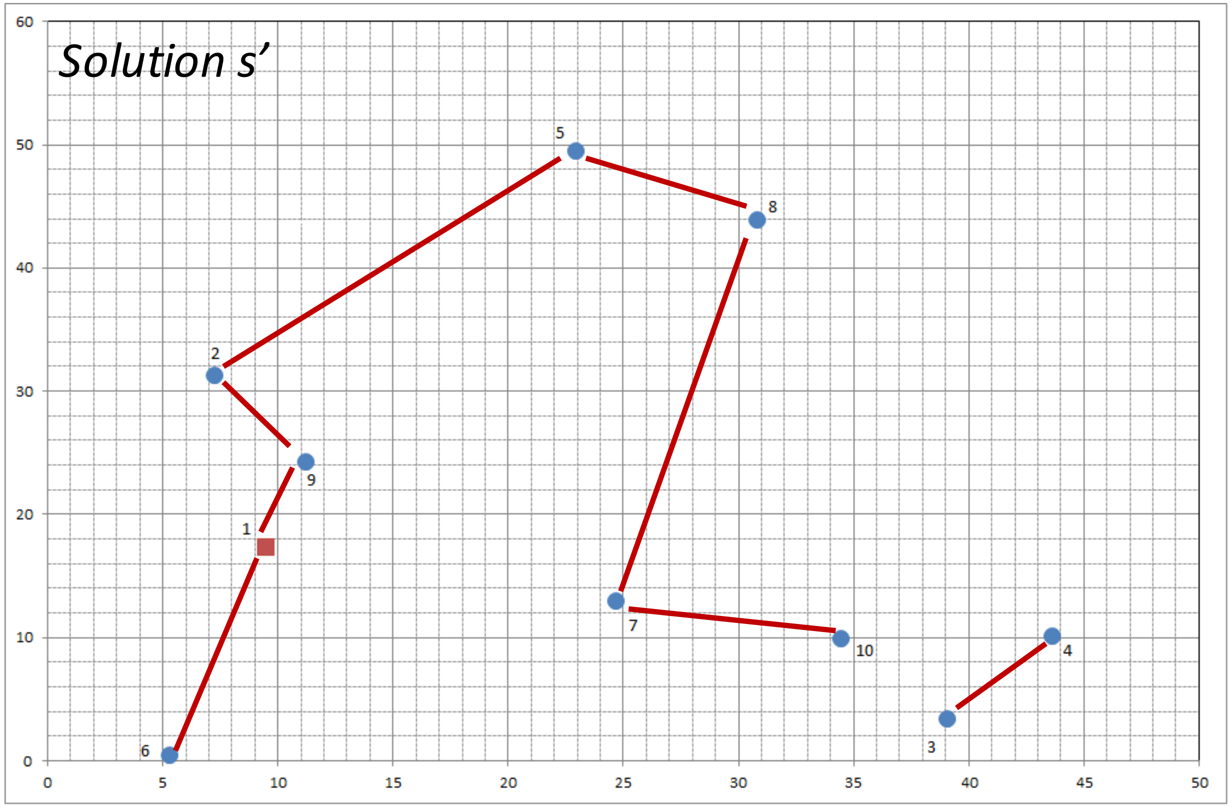
$N = \{\}$
for $i = 0$ to $n-1$ **do**
    $v = s_i$
    $s' = s \setminus v$
    for $a = 1$ to $|s'|$ **do**
        $s'' = relocate(s', v, a)$
        $N = N \cup s''$
    end for
end for
return N

Example: moves and neighborhoods for the TSP

2Opt move: eliminate 2 non-adjacent arcs and reconnect the tour

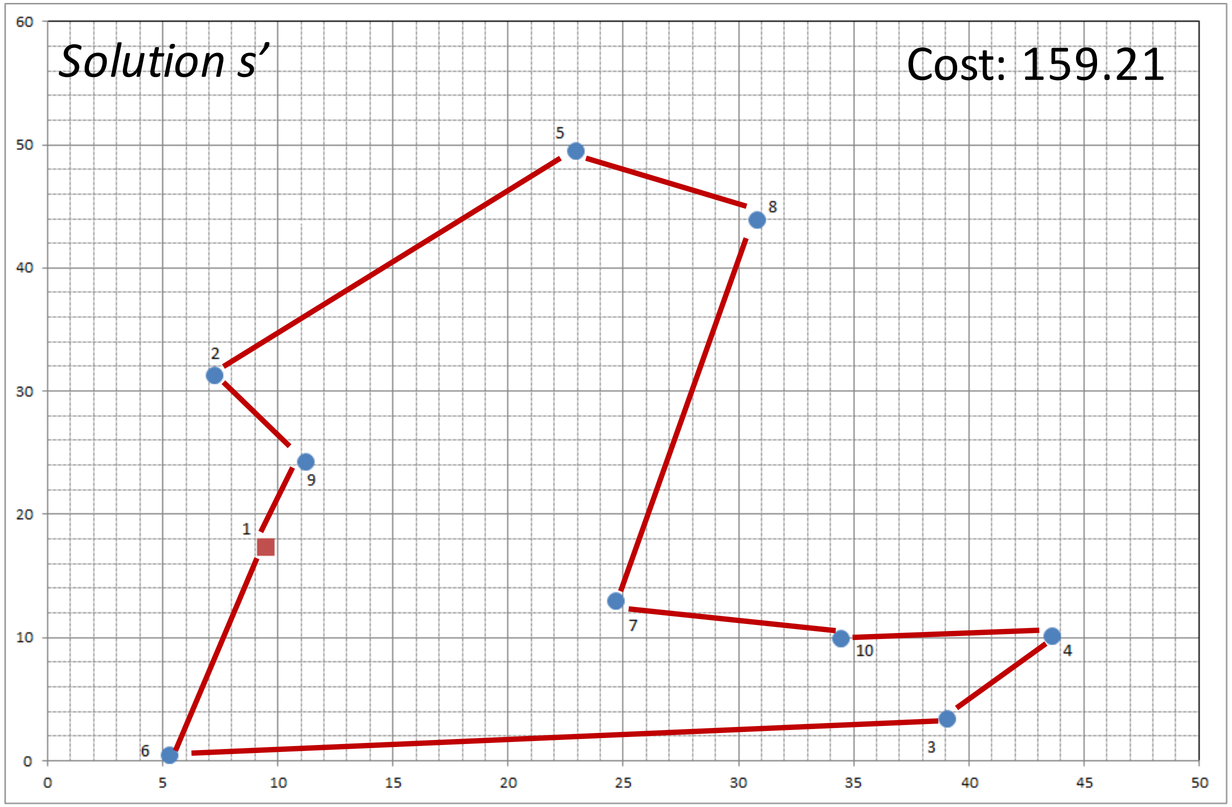
Example: moves and neighborhoods for the TSP

2Opt move: eliminate 2 non-adjacent arcs and reconnect the tour



Example: moves and neighborhoods for the TSP

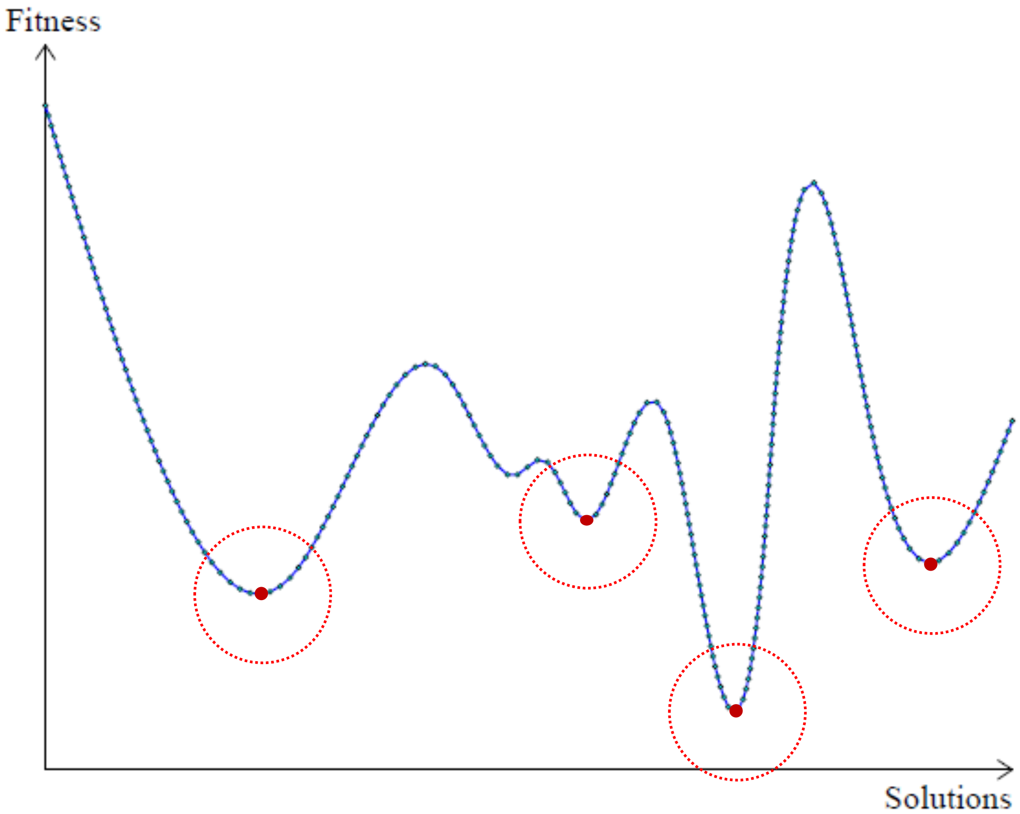
2Opt move: eliminate 2 non-adjacent arcs and reconnect the tour



twoOptNeighborhood(s)

?

Local optima

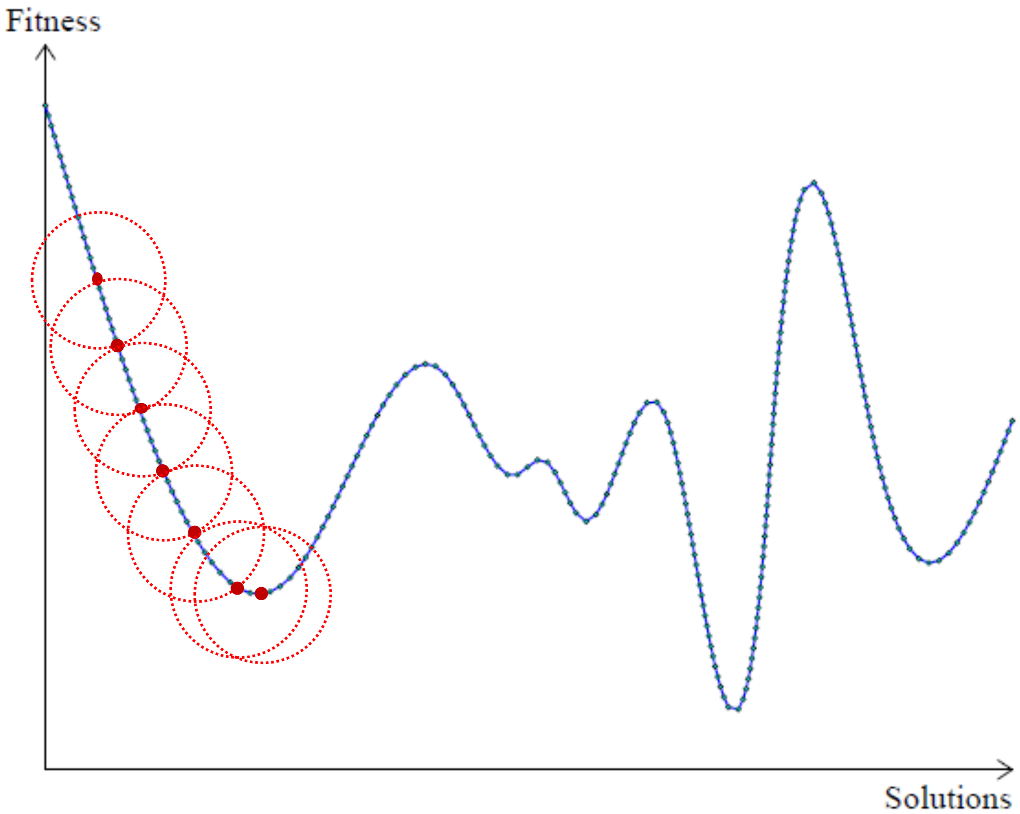


Local optimum:
Relatively to neighborhood N, a solution s ∈ S is a local optimum if it has a better quality than all its neighbors; that is, f (s) ≤ f (s') for all s' ∈ N(s)

A local optimum for a neighborhood N_1 **may not be** a local optimum for a different neighborhood N_2 !!!!!

Local Search

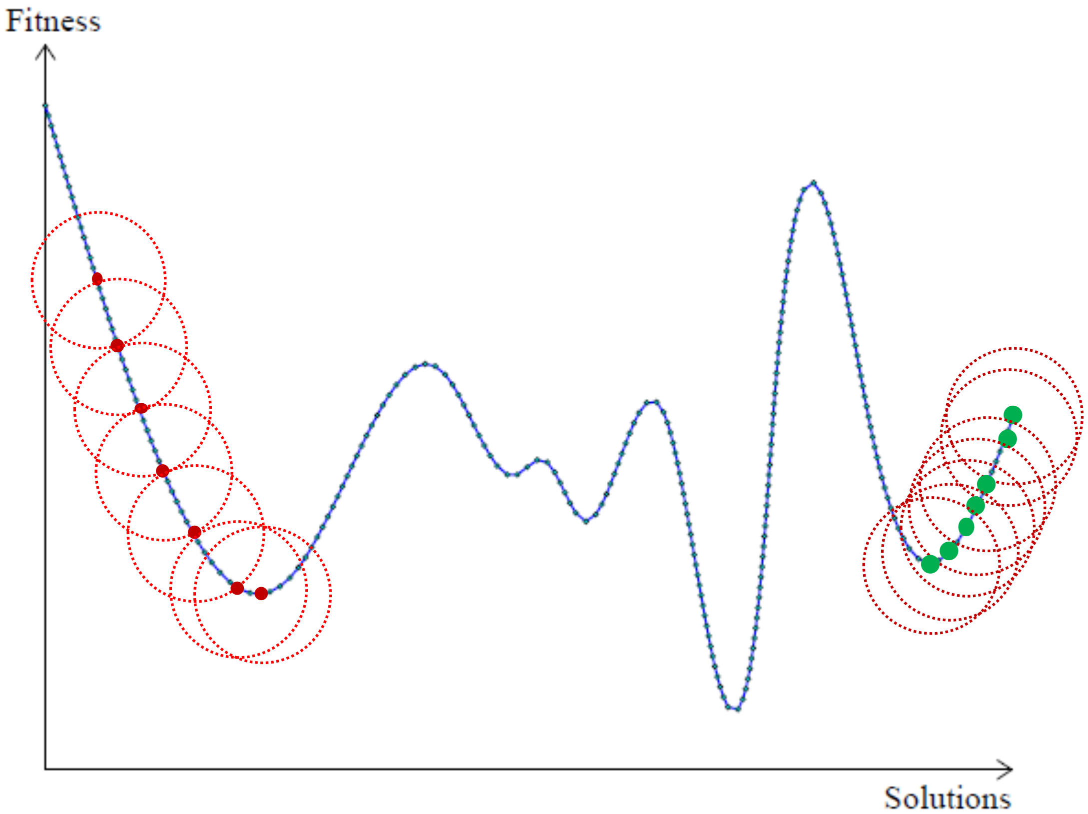
Principle: pure descent



- Create a starting solution s
- Explore the neighborhood for a better solution
- If you find a better solution, explore the neighborhood of that solution looking for a better one
- Repeat until you get trap in a local optimum

Local Search

Exploring neighborhoods: first vs. best improvement



Best improvement:

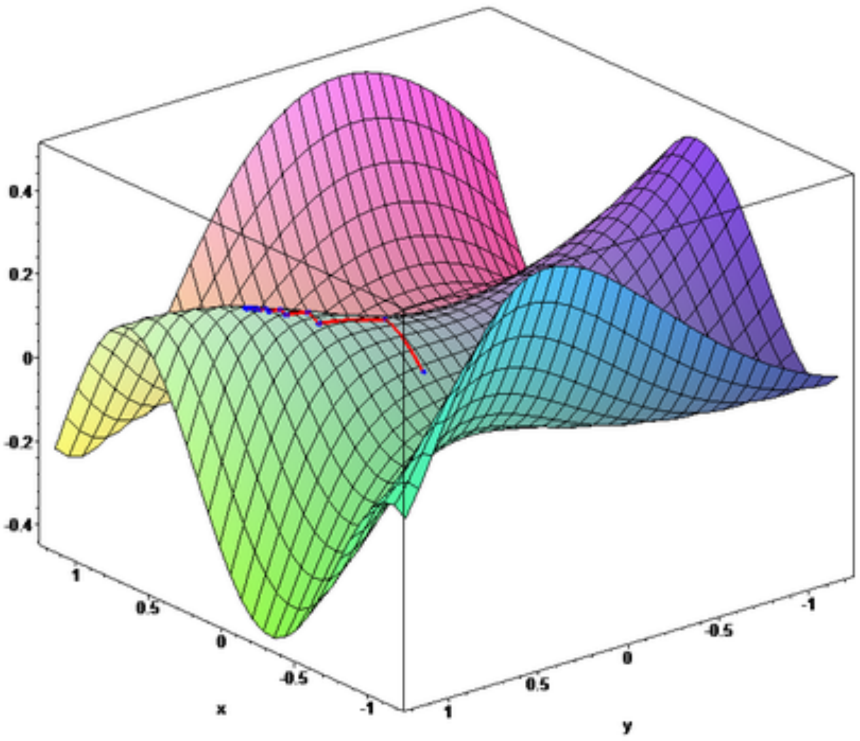
- Explore the entire neighborhood
- Move the search to the best solution found
- Start over

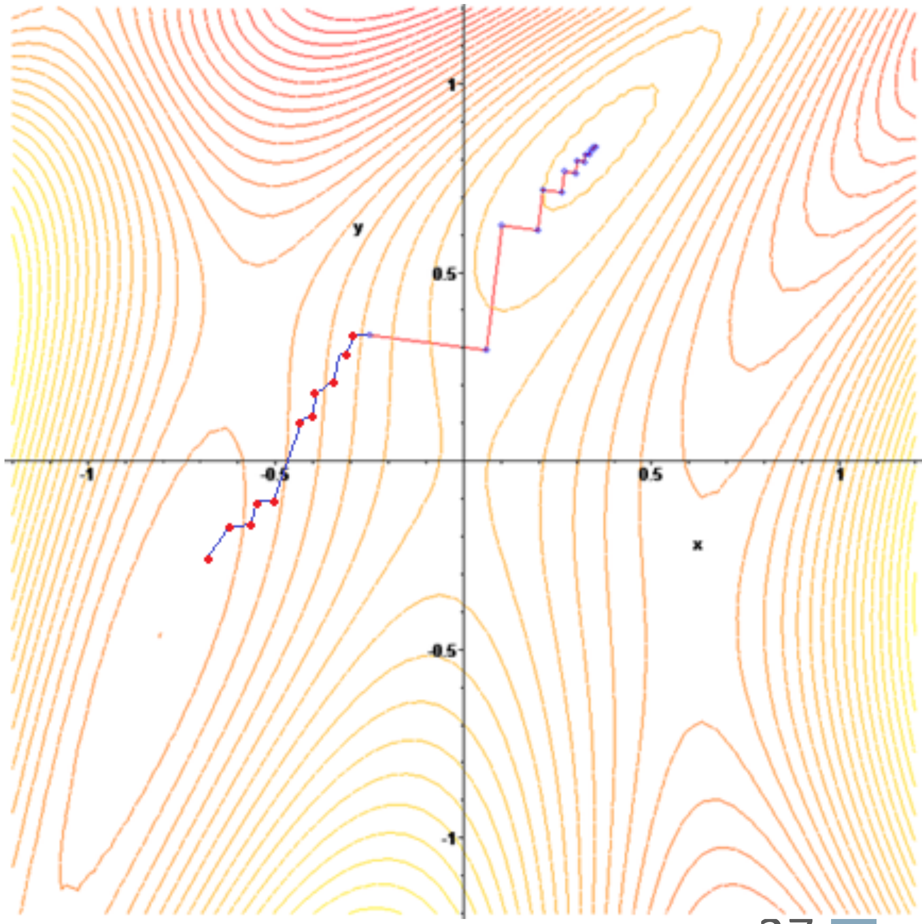
First improvement:

- Explore the neighborhood until you find an improving solution
- Move the search to that solution
- Start over

Local Search

Exploring neighborhoods: first vs. best improvement





Local Search

General framework

Step 1 (initialization)

a) choose an initial solution $s \in S$
b) $s^* \leftarrow s$ (i.e. record the best solution found so far)

step 2 (choice)

a) choose $s' \in N(s)$
b) $s \leftarrow s'$ (i.e. replace s by s')

step 3 (update & termination)

a) $s^* \leftarrow s$ if $f(s) < f(s^*)$
b) if the stop test is verified terminate and return s*; otherwise go to 2

Local Search

Choosing an initial solution: some ideas

- Random initialization
- Constructive heuristic
- Partially constructed + random completion

Local Search

Stopping criteria: some ideas

- Maximum number of iterations
- A number of iterations without improvement
- The improvement gap between two iterations is lower than a given constant
- Reach of an objective function target
- Maximum number of objective function evaluations
- Time limit

Local Search

General framework

Step 1 (initialization)

a) choose an initial solution $s \in S$
b) $s^* \leftarrow s$ (i.e. record the best solution found so far)

step 2 (choice)

a) choose $s' \in N(s)$
b) $s \leftarrow s'$ (i.e. replace s by s')

The million-dollar question:
how do we escape local optima?

step 3 (update & termination)

a) $s^* \leftarrow s$ if $f(s) < f(s^*)$
b) if the stop test is verified terminate and return s*; otherwise go to 2

Local Search

Escaping local optima

- Changing neighborhood structures
  - Variable neighborhood descent/search
- Starting from different solutions
  - Multi-start local search, GRASP
- Allow hill climbing moves
  - Tabu Search, Simulated Annealing
- Jumping to a different search region
  - Iterated local search

Local Search-based metaheuristics

Multi-start Local Search

Step 1 (initialization)

a) choose an initial solution $s \in S$
b) $s^* \leftarrow s$ (i.e. record the best solution found so far)

step 2 (choice)

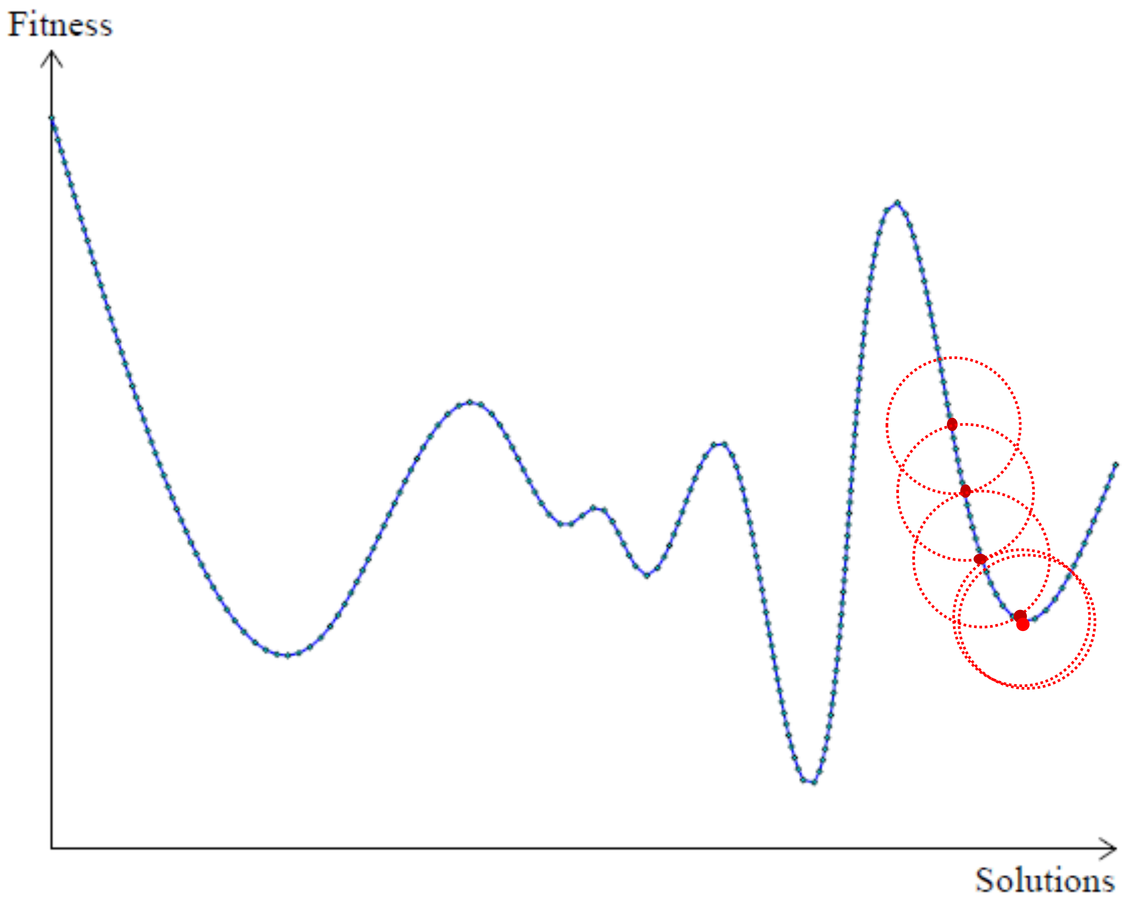
a) choose $s' \in N(s)$
b) $s \leftarrow s'$ (i.e. replace s by s')

step 3 (update & termination)

a) $s^* \leftarrow s$ if $f(s) < f(s^*)$
b) if the stop test is verified go to 3c; otherwise go to 2
c) If the second stop test is verified terminate and return s*; otherwise go to 1

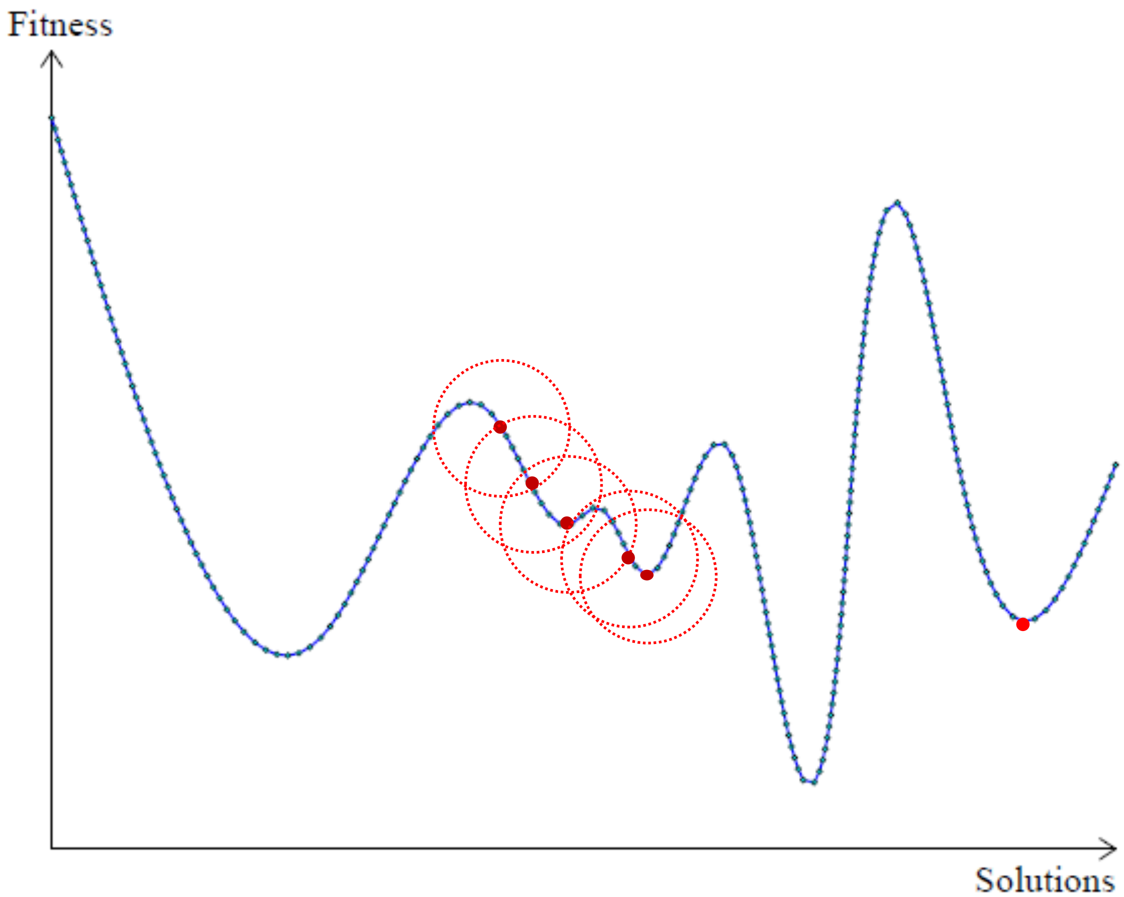
Local Search-based metaheuristics

Multi-start Local Search



Local Search-based metaheuristics

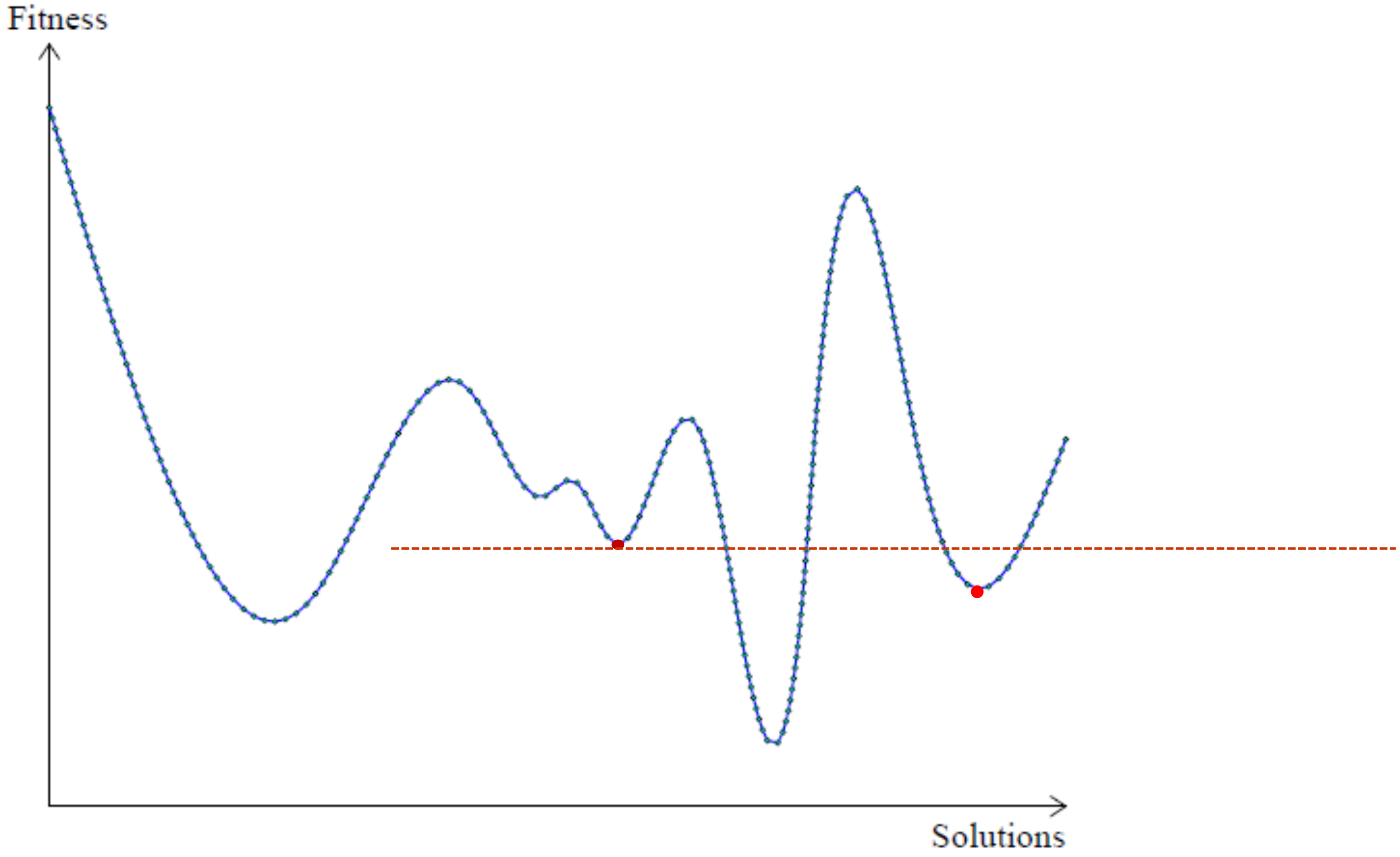
Multi-start Local Search



POLYTECH®
TOURS
Département Informatique

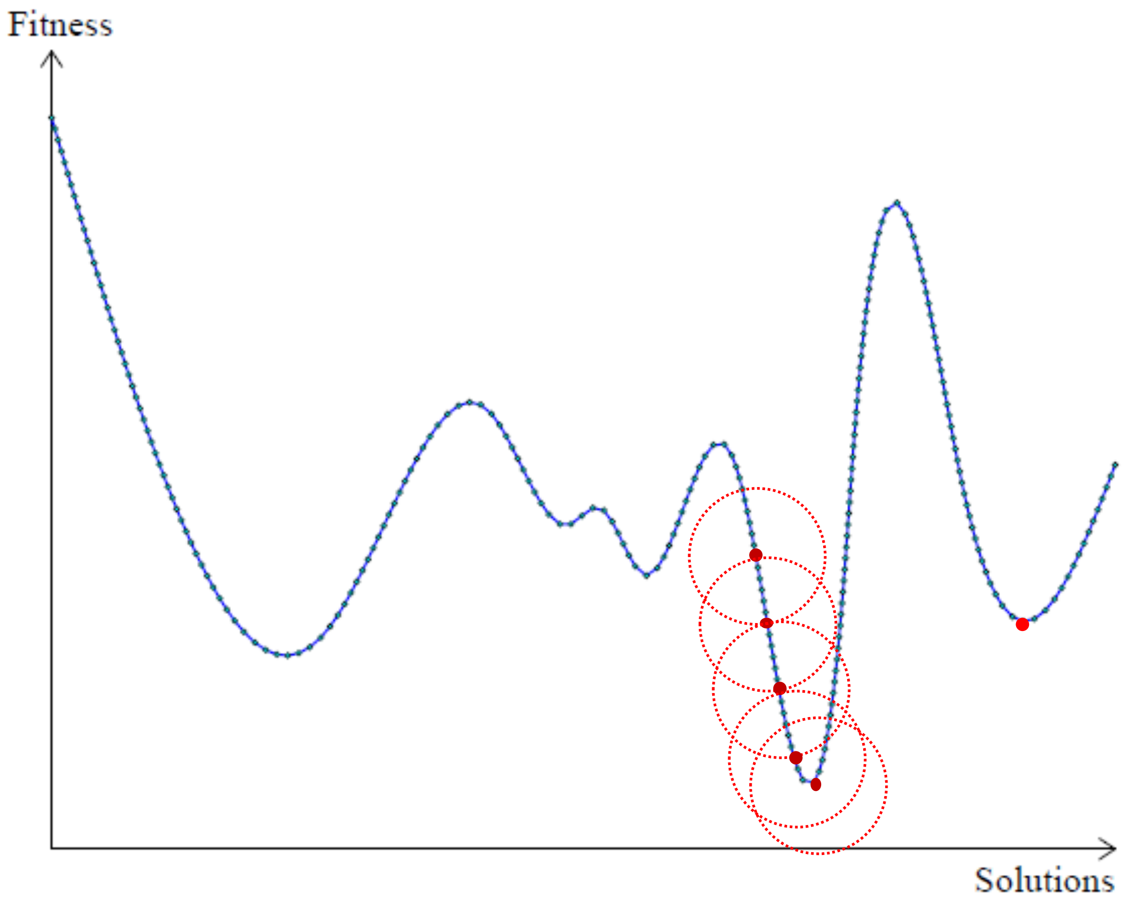
Local Search-based metaheuristics

Multi-start Local Search



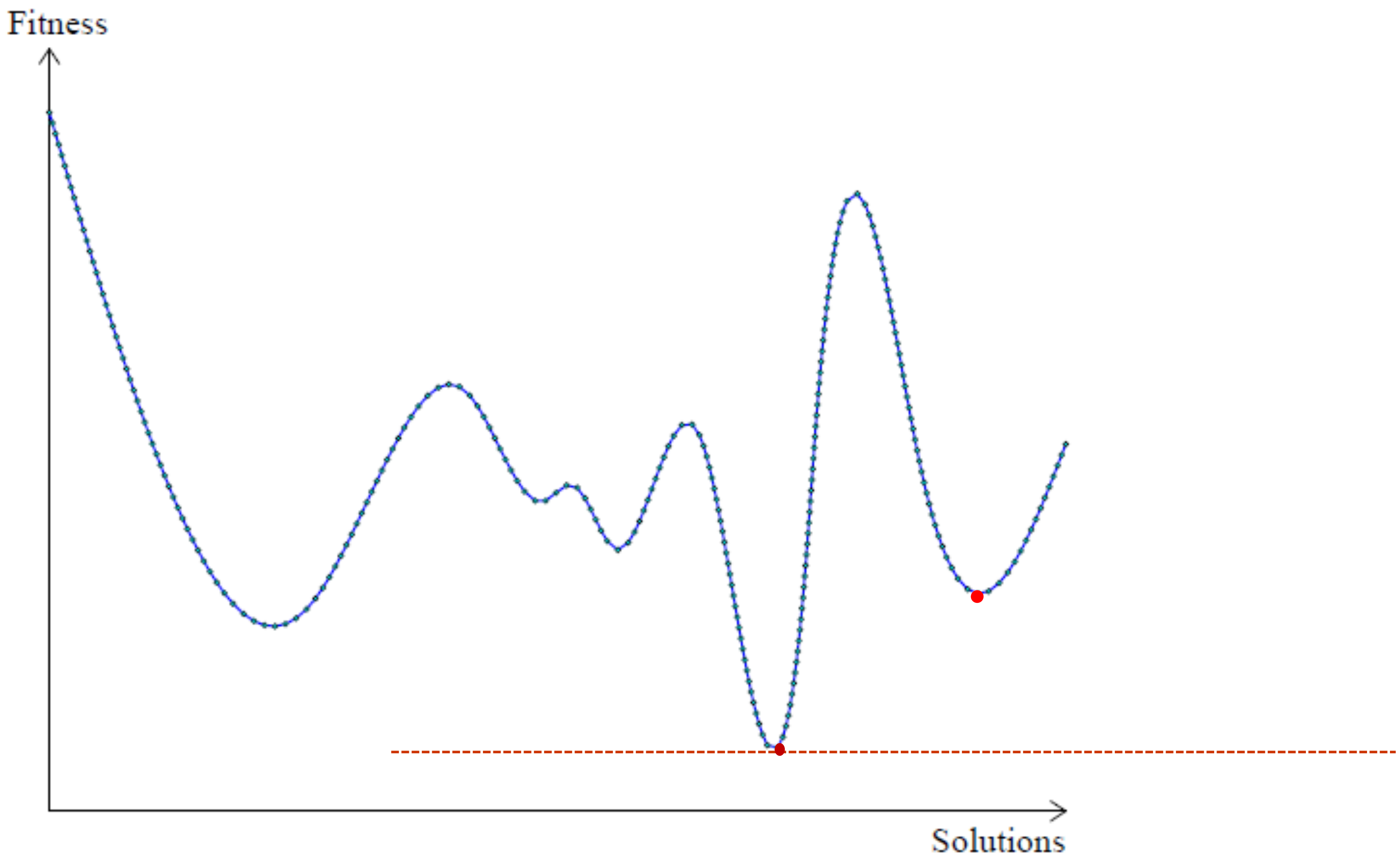
Local Search-based metaheuristics

Multi-start Local Search and GRASP



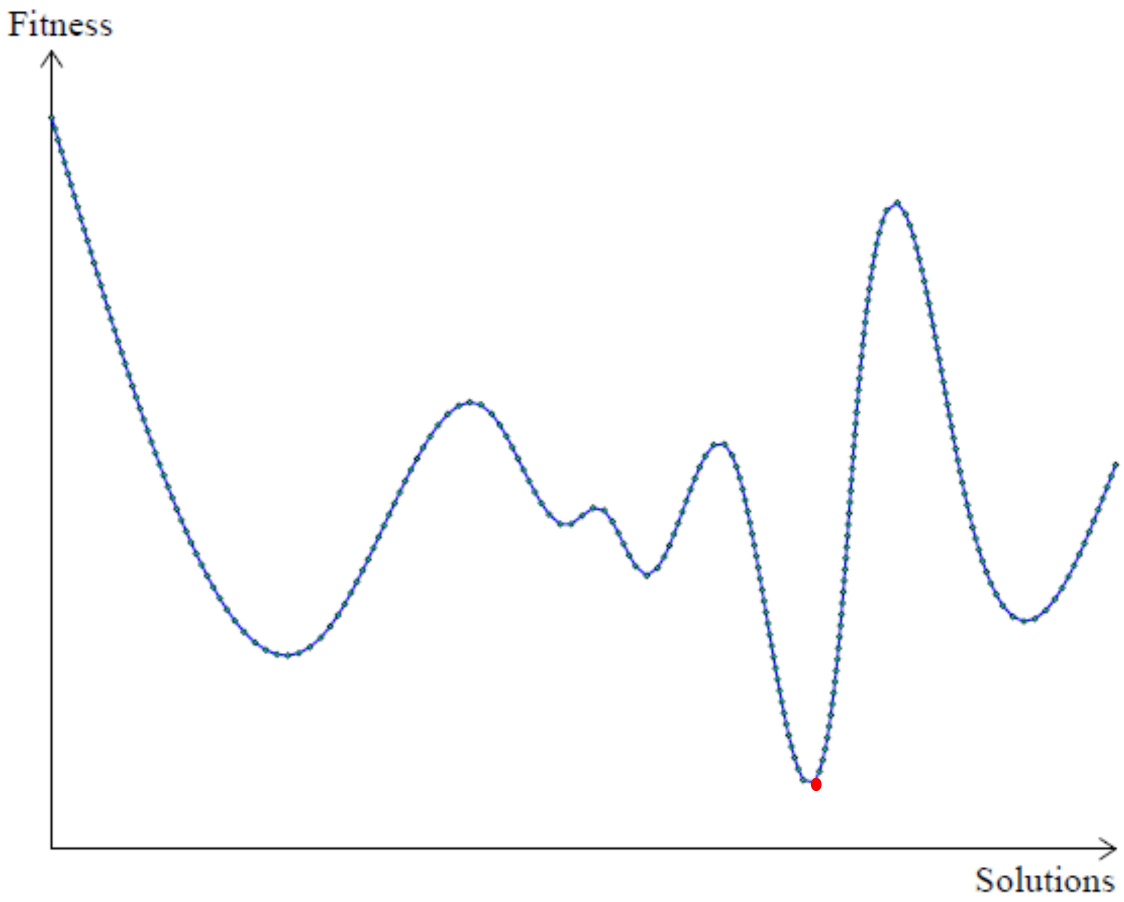
Local Search-based metaheuristics

Multi-start Local Search



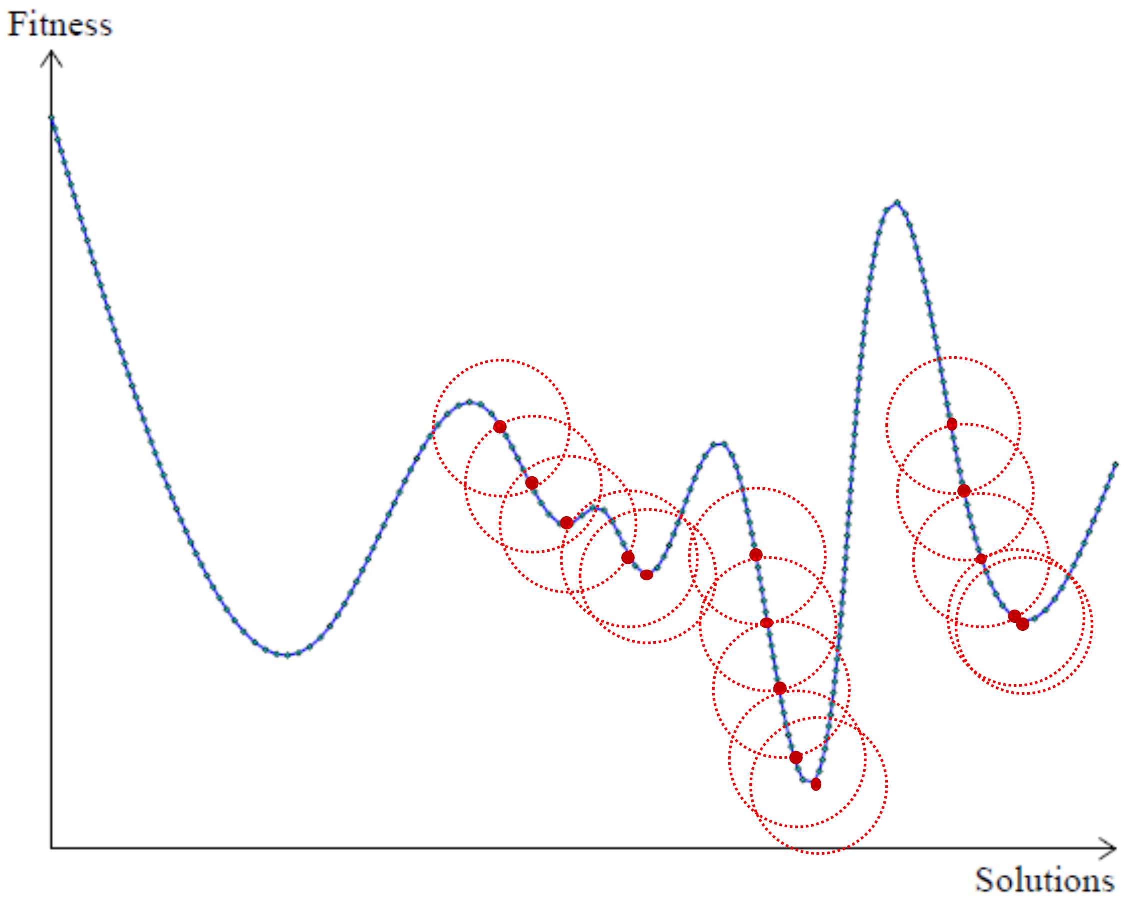
Local Search-based metaheuristics

Multi-start Local Search



Local Search-based metaheuristics

Multi-start Local Search



Multi-start Local Search

- Classic approach: Start from a randomly generated solution
- Alternative: start from a solution generated by a different heuristic each time

Multi-start local search

Pseudocode

```
mls(){
s* = generateRandomSolution()
while(!stop())
      s = generateRandomSolution()
      s’ = localSearch(s)
      if(f(s’)<f(s*))
            s*=s
end while
return s*
}
```

Multi-start local search

Pseudocode

```
localSearch(s){
continue=true
while(continue)
      s’ = exploreNeighborhood(s)
      if(f(s’)<f(s))
            s=s’
      else
            continue=false
end while
return s
}
```

Multi-start local search

Pseudocode

```
exploreNeighborhood(s){
s* = s
for(i=0 to s.size)
        for(j=0 to s.size)
                s'=swap(i,j,s)  //control special cases (e.g., i=j)
                if(f(s')<f(s*))
                        s*=s'
        end for
end for
return s*
}
```

Example with swap moves and best improvement configuration

Your assignment

- Design a parallel version of the multi-start local search algorithm for the TSP (there are plenty of opportunities!)
- Implement your algorithm in Java
- Conduct a small computational study on standard instances
  - How much can you speed up your computations?
  - What is the best configuration for your algorithm?
    - Number of threads?
    - Number of tasks?