中山大學
SUN YAT-SEN UNIVERSITY

Lecture 4
Metaheuristic Algorithms (I)

Algorithm

张子臻，中山大学计算机学院

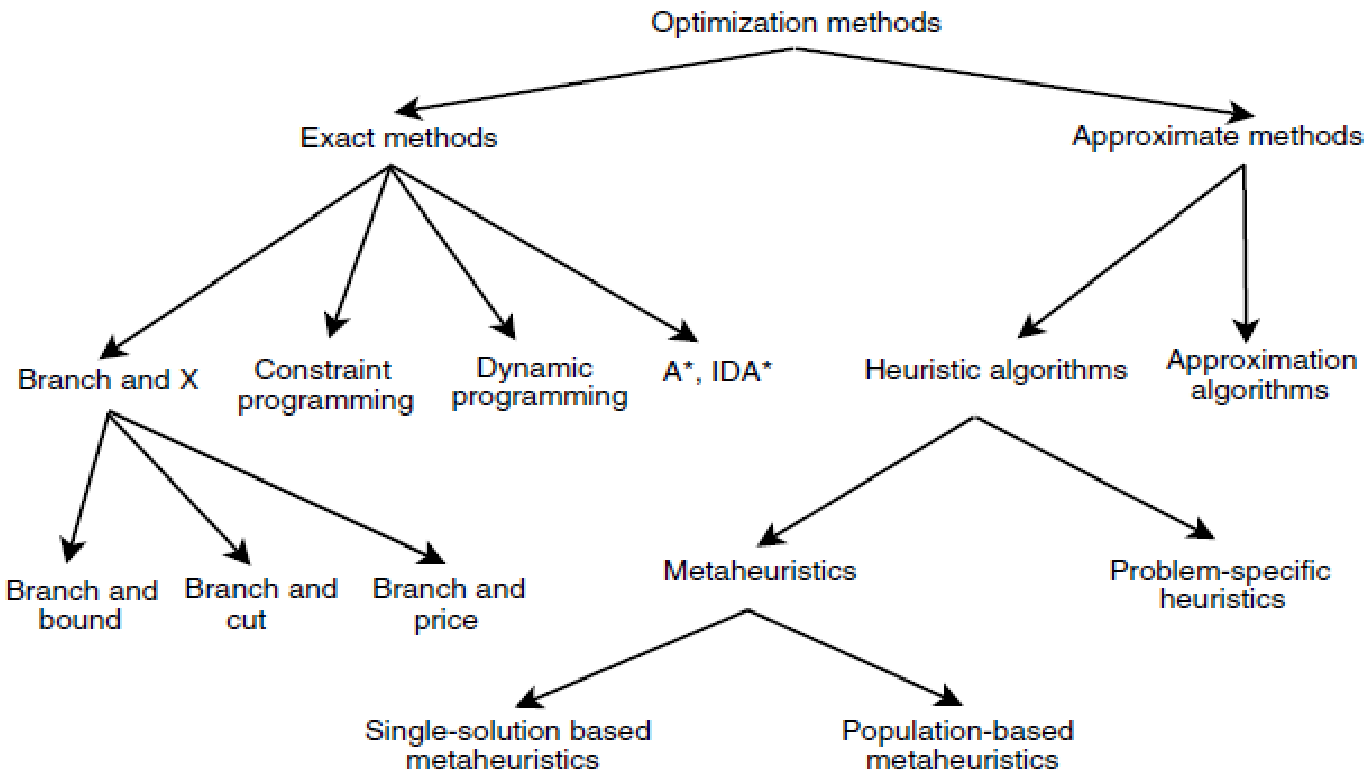
zhangzizhen@gmail.com

Outline

- Introduction of Metaheuristics
- Main Components of Metaheuristics
  - Representation
  - Objective Function
  - Constraint Handling
  - Neighborhood
- Single-Solution Based Metaheuristics
  - Basic Concepts
  - Local Search

Classical Optimization Methods

- Exact methods obtain optimal solutions and guarantee their optimality.
- Approximate (or heuristic) methods generate high-quality solutions in reasonable time for practical use, but there is no guarantee of finding a global optimal solution.

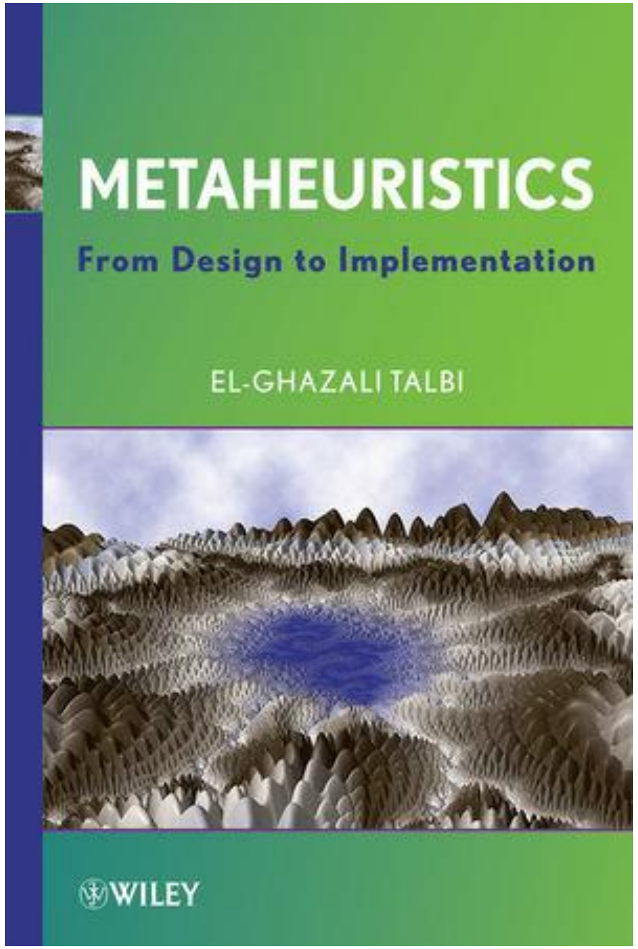


Heuristic vs. Metaheuristic

- ***Heuristic* (启发式)**
  - is origin in the old Greek word *heuriskein*, which means the art of discovering new strategies (rules) to solve problems.
- ***Meta* (元)**
  - A Greek word, means “upper level methodology”.
- ***Meta-heuristic* (元启发式)**
  - can be defined as upper level general methodologies that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems.

Metaheuristics

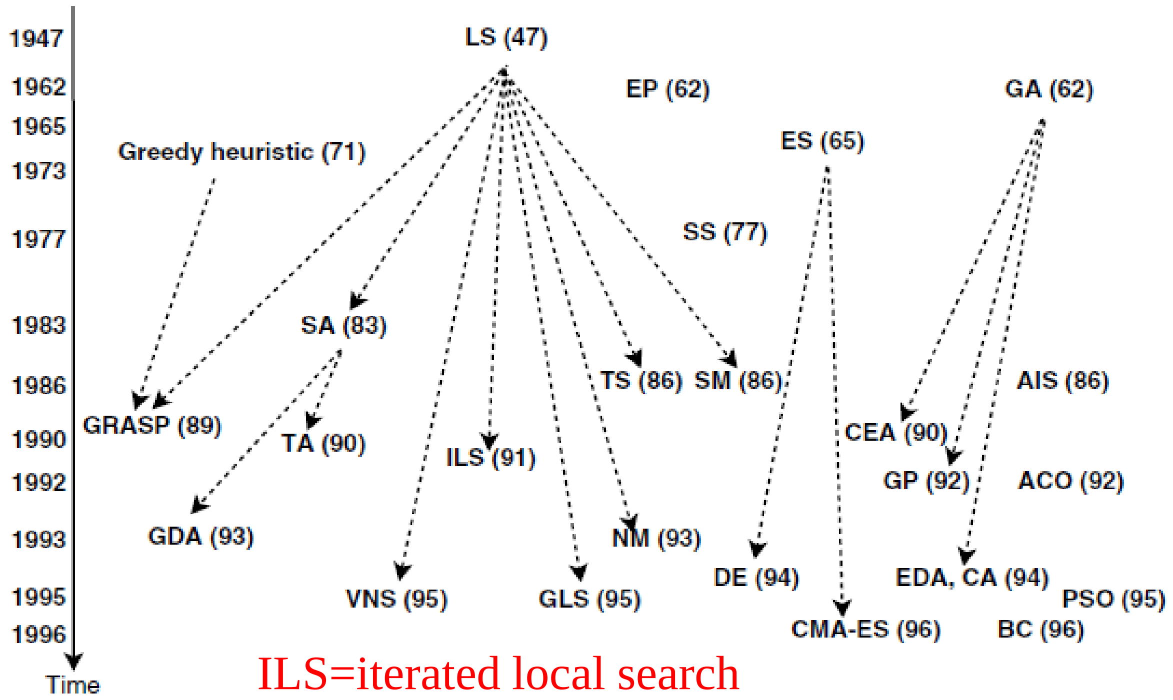
- Metaheuristics are able to tackle large-size problem instances by delivering satisfactory solutions in a reasonable time.
- There is no guarantee to find global optimal solutions or even bounded solutions.
- Metaheuristics are efficient and effective to solve large and complex problems.



Application of Metaheuristics

- Application of metaheuristics falls into a large number of areas; some of them are:
  - Engineering design, topology optimization and structural optimization in electronics and VLSI, aerodynamics, fluid dynamics, telecommunications, automotive, and robotics.
  - Machine learning and data mining in bioinformatics and computational biology, and finance.
  - System modeling, simulation and identification in chemistry, physics, and biology; control, signal, and image processing.
  - Planning in routing problems, robot planning, scheduling and production problems, logistics and transportation, supply chain management, environment, and so on.

Genealogy (家谱) of Metaheuristics



ILS=iterated local search
NM=noisy method
PSO=particle swarm optimization
SA=simulated annealing
SM=smoothing method
SS=scatter search
TS=tabu search

ACO=ant colonies optimization
AIS=artificial immune systems
BC=bee colony
CA=cultural algorithms
CEA=coevolutionary algorithms
CMA-ES=covariance matrix adaptation evolution strategy
DE=differential evolution
EDA=estimation of distribution algorithms
EP=evolutionary programming
ES=evolution strategies
GA=genetic algorithms
GDA=great deluge
GLS=guided local search
GP =genetic programming
GRASP=greedy adaptive search procedure
VNS =variable neighborhood search

Classification of Metaheuristics

- **Nature inspired versus non-nature inspired**
  - Evolutionary algorithms and artificial immune systems from biology;
  - Ants, bees colonies, and particle swarm optimization from swarm intelligence into different species;
  - Simulated annealing from physics.

Classification of Metaheuristics

- **Memory usage versus memoryless methods**
  - Some metaheuristic algorithms are memoryless; that is, no information extracted dynamically is used during the search.
  - Some representatives of this class are local search, GRASP, and simulated annealing.
  - While other metaheuristics use a memory that contains some information extracted online during the search.
  - For instance, short-term and long-term memories in tabu search.

Classification of Metaheuristics

- **Deterministic versus stochastic**
  - A deterministic metaheuristic solves an optimization problem by making deterministic decisions (e.g., local search, tabu search).
  - In stochastic metaheuristics, some random rules are applied during the search (e.g., simulated annealing, evolutionary algorithms).
  - In deterministic algorithms, using the same initial solution will lead to the same final solution, whereas in stochastic metaheuristics, different final solutions may be obtained from the same initial solution.

Classification of Metaheuristics

- **Population-based search versus single-solution based search**
  - Single-solution based algorithms (e.g., local search, simulated annealing) manipulate and transform a single solution during the search while in population-based algorithms (e.g., particle swarm, evolutionary algorithms) a whole population of solutions is evolved.
  - These two families have complementary characteristics: single-solution based metaheuristics are exploitation oriented; they have the power to intensify the search in local regions. Population-based metaheuristics are exploration oriented; they allow a better diversification in the whole search space.

Classification of Metaheuristics

- **Iterative versus greedy**
  - In iterative algorithms, we start with a complete solution (or population of solutions) and transform it at each iteration using some search operators.
  - Greedy algorithms start from an empty solution, and at each step a decision variable of the problem is assigned until a complete solution is obtained.
  - Most of the metaheuristics are iterative algorithms.

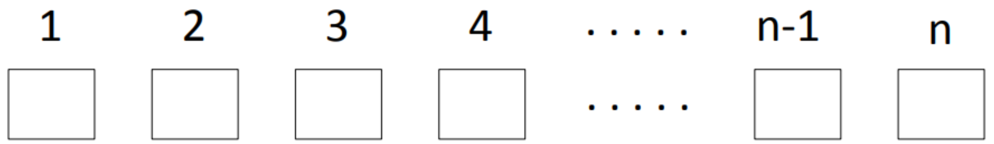
Main Components of Metaheuristics

- The representation of solutions and the definition of the objective function
- **Representation**
  - Designing any iterative metaheuristic needs an encoding (representation) of a solution.
  - The encoding plays a major role in the efficiency and effectiveness of any metaheuristic and constitutes an essential step in designing a metaheuristic.
  - The encoding must be suitable and relevant to the tackled optimization problem.
  - The efficiency of a representation is also related to the search operators.

Binary Encoding

- **0/1 knapsack problem.** For a 0/1-knapsack problem of *n* objects, a vector **s** of binary variables of size *n* may be used to represent a solution:

$$\forall i, s_i = \begin{cases} 1 & \text{if object } i \text{ is in the knapsack} \\ 0 & \text{otherwise} \end{cases}$$



0 1 0 1 1 0 1 0 1 0 1 0 1. . . . . . .1 0 1

A binary string of n-bits

Real-value Encoding

- The real-value encoding is most suitable for optimization in a continuous search space.
- It uses the direct representations for the designed parameters.
- It avoids any intermediate encoding and decoding steps.

x	y

5.28	-475.36

Real-value representation

Real-value Encoding

- For any continuous design variable x such that $X_L \leq x \leq X_U$, and if ε is the precision required, then string length n should be equal to

$$n = \log_2\left(\frac{X_U - X_L}{\varepsilon} + 1\right)$$

- Equivalently,

$$\varepsilon = \frac{X_U - X_L}{2^n - 1}$$

- For example,
  - $1 \leq x \leq 4$, $\varepsilon = 0.5$
  - $n = \log_2(7) \leq 3$
  - $1 \leftrightarrow (000)$, $1.5 \leftrightarrow (001)$, $2 \leftrightarrow (010)$, $2.5 \leftrightarrow (011)$, $3 \leftrightarrow (100)$, $3.5 \leftrightarrow (101)$, $4 \leftrightarrow (110)$

Real-value Encoding

- Once we know the length of binary string *n* for an obtainable accuracy (i.e., precision), then we can have the following mapping relation from a real value *X* to its binary equivalent decoded value X_B, which is given by

$$X = X_L + \frac{X_U - X_L}{2^n - 1} \times X_B,$$

where X_B is the decoded value of a binary string, $X_L \leftrightarrow (0\ 0\ \ldots\ 0)$, $X_U \leftrightarrow (1\ 1\ \ldots\ 1)$

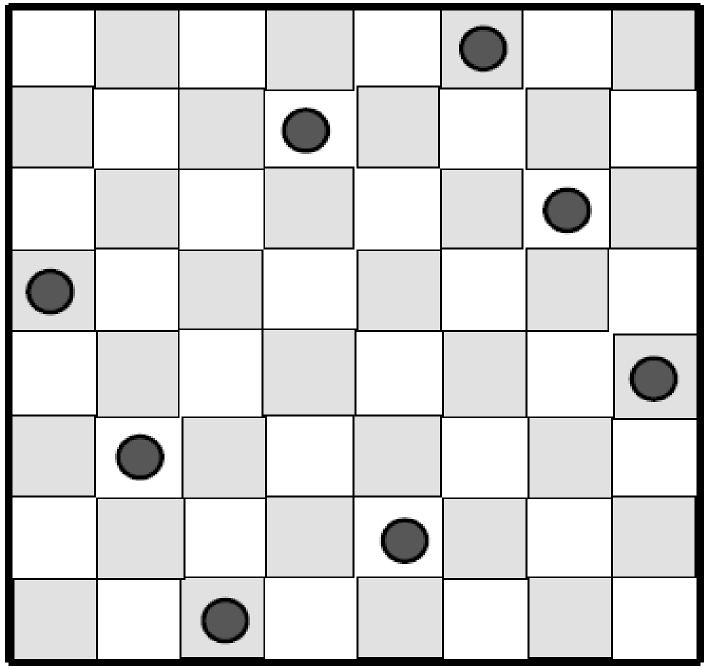
- Example:
  - $X_L = 2 \leftrightarrow (0\ 0\ 0\ 0)$, $X_U = 17 \leftrightarrow (1\ 1\ 1\ 1)$, $n = 4$
  - $X_B = 10 \leftrightarrow (1\ 0\ 1\ 0)$
  - Then, $X = 2 + \frac{17-2}{2^4-1} \times 10 = 12$

Permutation Encoding

- Traveling salesman problem:
  - For a TSP problem with n cities, a tour may be represented by a permutation of size n.
  - Each permutation decodes a unique solution.
  - The solution space is represented by the set of all permutations.
  - Its size is $|S| = (n - 1)!$ if the first city of the tour is fixed.

Permutation Encoding

- The 8-Queen Problem:
  - The following solution can be represented by the permutation (6,4,7,1,8,2,5,3).
  - What if it is not a feasible solution?



Main Concepts for Metaheuristics

- A representation must have the following characteristics:
  - **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 the search operators. The time and space complexities of the operators dealing with the representation must be reduced.

Objective Function

- The objective function f formulates the goal to achieve.
- It associates with each solution of the search space a real value that describes the quality or the fitness of the solution, $f : S \to R$.
- From the representation space of the solutions R, some decoding functions d may be applied, $d : R \to S$, to generate a solution that can be evaluated by the function f .
- The objective function is an important element in designing a metaheuristic.
- It will guide the search toward “good” solutions of the search space.

Self-sufficient Objective Functions

- **Example**
  - In many routing problems such as TSP and vehicle routing problems, the formulated objective is to minimize a given global distance.
  - For instance, the objective corresponds to the total distance of the Hamiltonian tour:

$$f(s) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

where π represents a permutation encoding a tour and n is the number of cities.

Guiding Objective Functions

- The objective function will guide the search in a more efficient manner.
- Example—Objective function to *k*-satisfiability problems (*k*-SAT).
  - We are given a function F, composed of m clauses C_i of k Boolean variables.

$$F = (x_1 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3})$$

  - The objective of the problem is to find an assignment of the k Boolean variables such that the value of the function F is *true*.

Guiding Objective Functions

- A solution for the problem may be represented by a vector of *k* binary variables. A straightforward objective function is to use the original *F* function:

$$f = \begin{cases} 0 & \text{if is } F \text{ false} \\ 1 & \text{otherwise} \end{cases}$$

- If one considers two solutions $s_1 = (1, 0, 1,1)$ and $s_2 = (1, 1, 1, 1)$, they will have the same objective function (what's that?).

$$F = (x_1 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee x_2 \vee x_4)$$
$$\wedge (x_2 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3})$$

- The drawback of this objective function is that it has a poor differentiation between solutions.

Guiding Objective Functions

- A more interesting objective function to solve the problem will be to count the number of satisfied clauses.
- Hence, the objective will be to maximize the number of satisfied clauses.
- This function is better in terms of guiding the search toward the optimal solution.
- In this case, the solution s_1 (resp. s_2) will have a value of 5 (resp. 6)

$$F = (x_1 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3})$$

Constraint Handling

- **Reject Strategies**
  - Only feasible solutions are kept during the search and then infeasible solutions are automatically discarded.
  - Good if the portion of infeasible solutions of the search space is very small.
  - Do not exploit infeasible solutions.
- However,
  - Feasible regions of the search space may be discontinuous.
  - A path between two feasible solutions exists if it is composed of infeasible solutions.

Constraint Handling

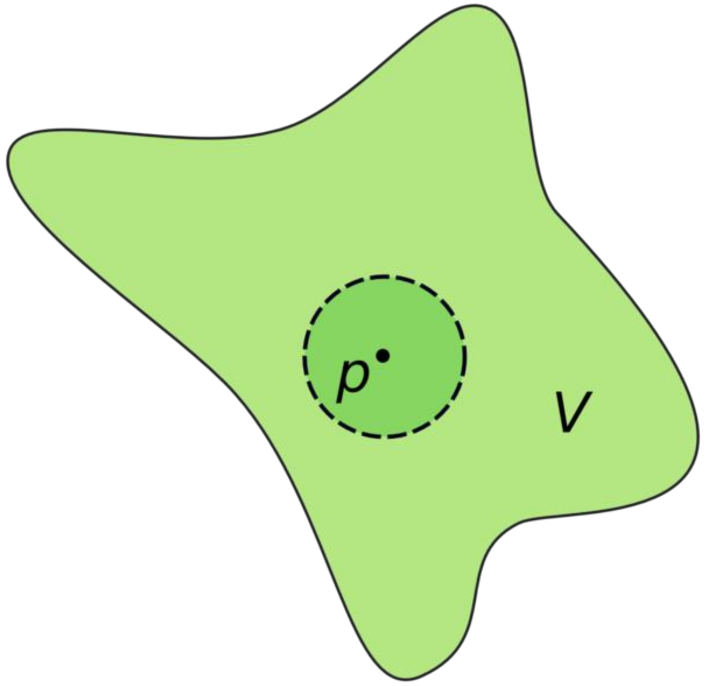
- **Penalizing Strategies**
  - Infeasible solutions are considered during the search process.
  - The objective function is extended by a penalty function that will penalize infeasible solutions.
  - The objective function f may be penalized in a linear manner:

$$f'(s) = f(s) + \lambda c(s),$$

where $c(s)$ represents the cost of the constraint violation and λ is a weight. (e.g., knapsack problem)

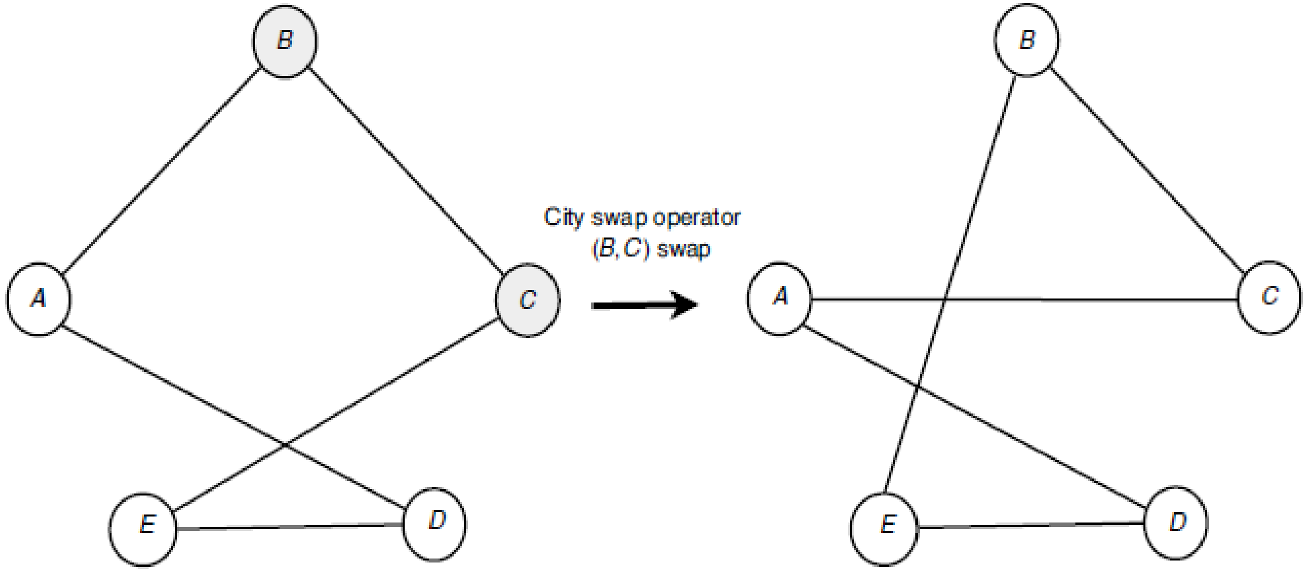
Neighborhood

- It plays a crucial role in the performance of a metaheuristic.
- A solution in the neighborhood is called a neighbor.
- A neighbor *s*' is generated by modifying the current solution *s*.
- The area of the neighborhood is relied on the operator employed. (operators can be regarded the ways or rules of modifying *s*.)



Neighborhood Operators

- For permutation problems, such as the TSP, single machine scheduling problem and *N* queens problem, the **exchange operator** (swap operator) may be used.



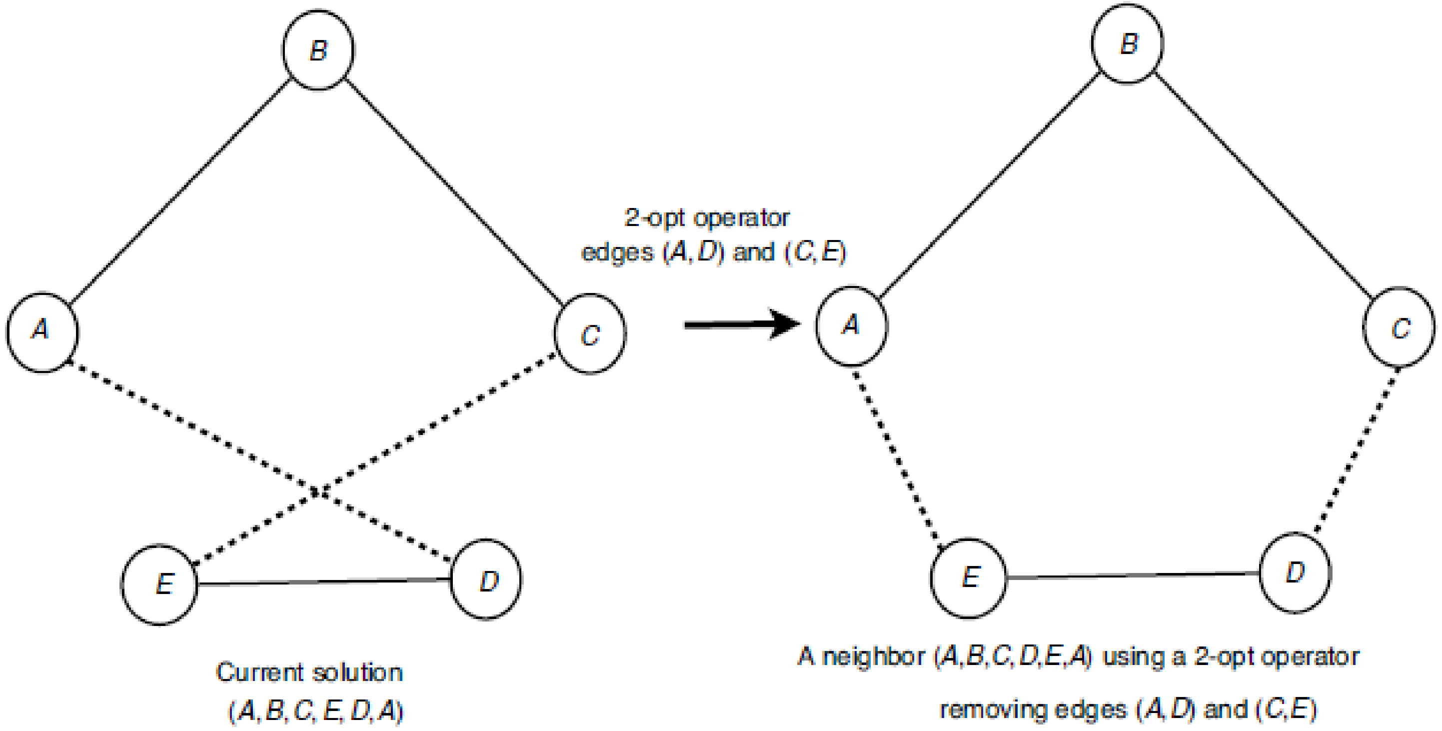
Current solution
(A, B, C, E, D, A)

A neighbor using a city swap operator
between B and C (A, C, B, E, D, A)

The size of this neighborhood is $n(n-1)/2$,
where n is the number of cities.

Neighborhood Operators

- **2-opt operator**

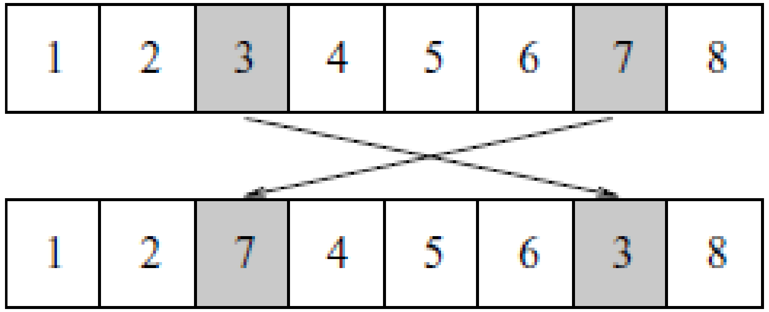


The size of the neighborhood for the 2-opt operator is $[(n(n-1)/2) - n]$;
All pairs of edges are concerned except the adjacent pairs.

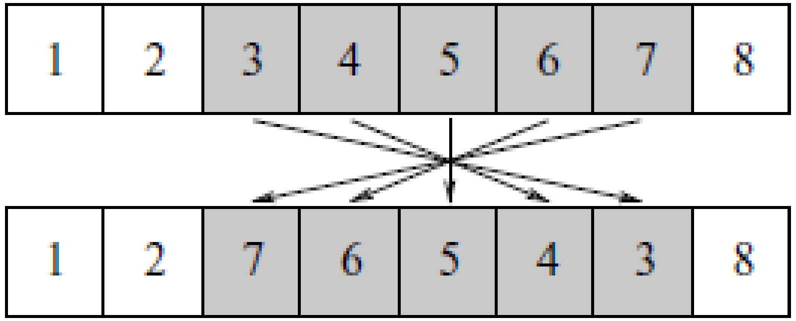
Neighborhood Operators

Insertion operator

Exchange operator

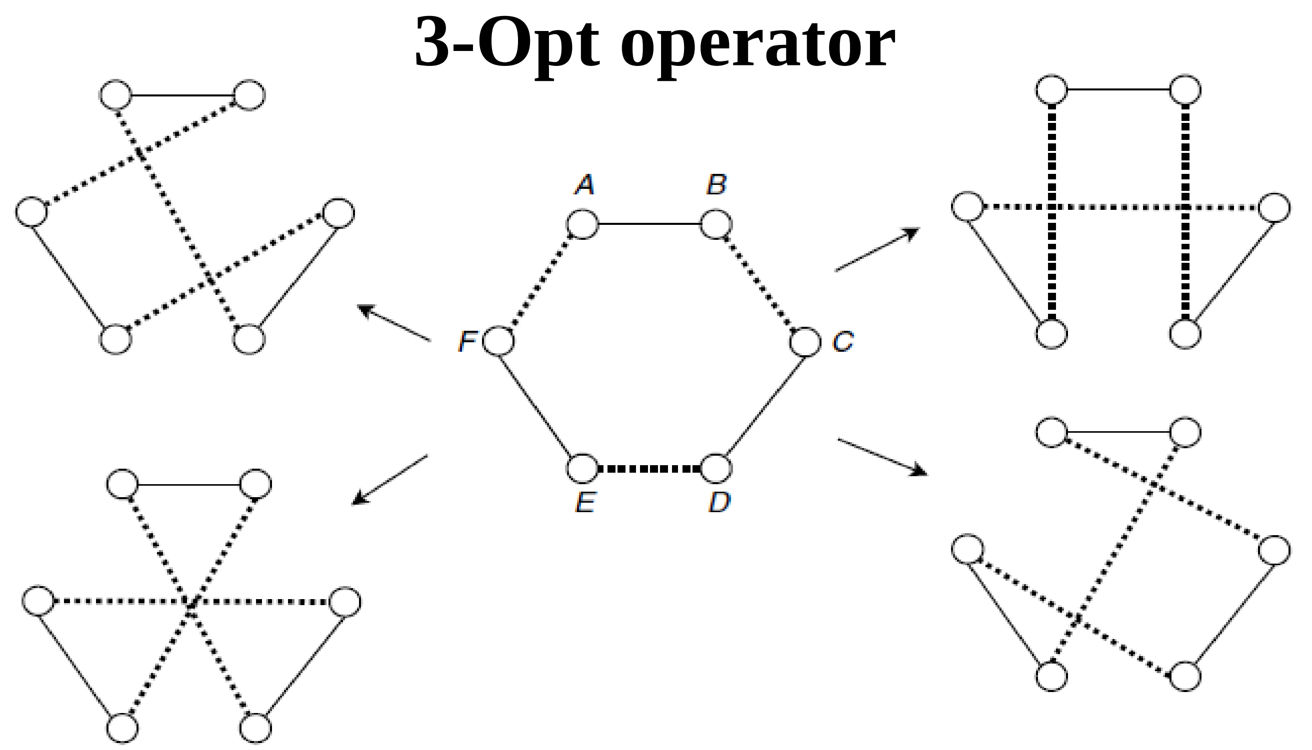


Inversion operator



Neighborhood Operators

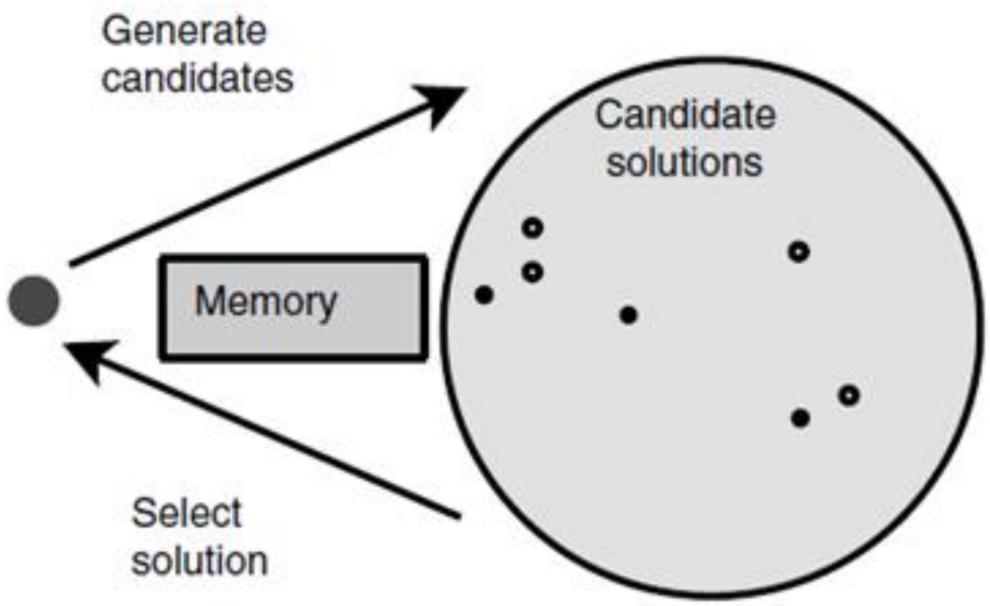
- Another widely used operator is the *k*-opt operator, where *k* edges are removed from the solution and replaced with other *k* edges.
- The time complexity for 2-opt, 3-opt and 4-opt is $O(n^2)$, $O(n^3)$ and $O(n^4)$.



3-opt operator for the TSP. The neighbors of the solution (A,B,C,D,E,F) are (A,B,F,E,C,D), (A,B,D,C,F,E), (A,B,E,F,C,D), and (A,B,E,F,D,C).

Single-Solution Based Metaheuristics

- Single-metaheuristics iteratively apply the *generation* and *replacement* procedure from the current single solution.
- Examples of Single-metaheuristics include ***local search***, ***simulated annealing***, and ***tabu search***.



Memory usage versus memoryless

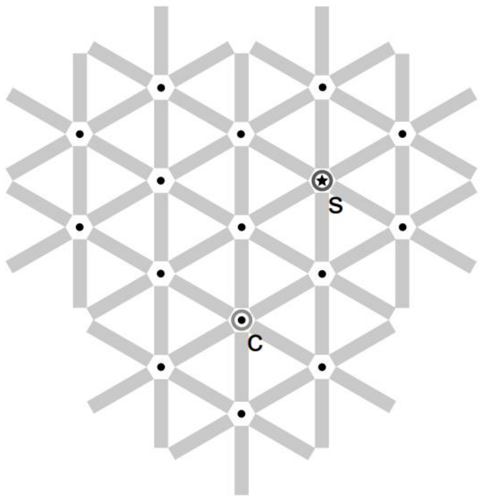
Algorithm 2.1 High-level template of S-metaheuristics.

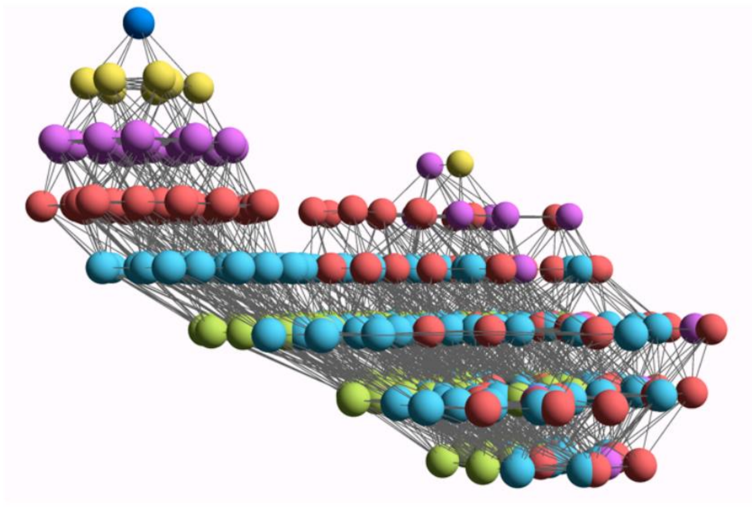
Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  $s_{t+1} = \text{Select}(C(s_t))$;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.

- The generation and the replacement phases may be *memoryless*. In this case, the two procedures are based only on the current solution.
- Otherwise, some *history* of the search stored in a memory can be used in the generation of the candidate list of solutions and the selection of the new solution.

Neighborhood

- **Neighborhood**
  - The set of neighboring solutions
  - The area of the neighborhood is relied on the ***operator*** employed
- **Neighborhood graph**
  - vertices: candidate solutions (search positions)
  - vertex labels: evaluation function
  - edges: connect “neighboring” positions
  - *s*: (optimal) solution
  - *c*: current search position



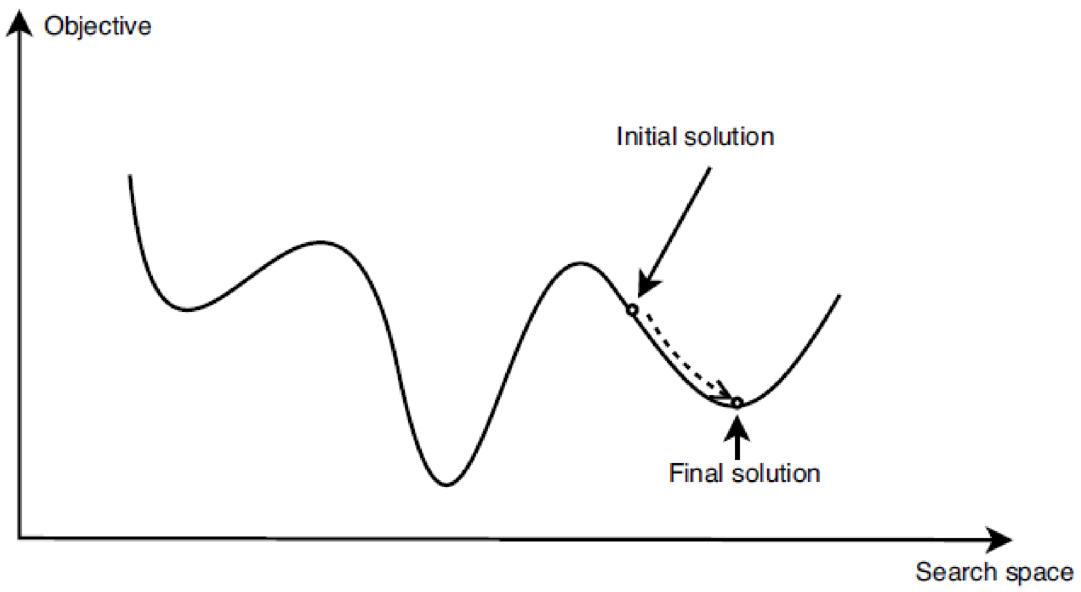


Components of Search

- Given a (combinatorial) optimization problem Π and one of its instances π,
  1. search space $S(\pi)$
  2. evaluation function $f_\pi : S(\pi) \to R$
  3. neighborhood function $N_\pi : S \to 2^{S(\pi)}$
  4. set of memory states $M(\pi)$
  5. initialization function init: $\emptyset \to S(\pi)$
  6. step function step: $S(\pi) \times M(\pi) \to S(\pi) \times M(\pi)$
  7. termination predicate terminate : $S(\pi) \times M(\pi) \to \{0, 1\}$

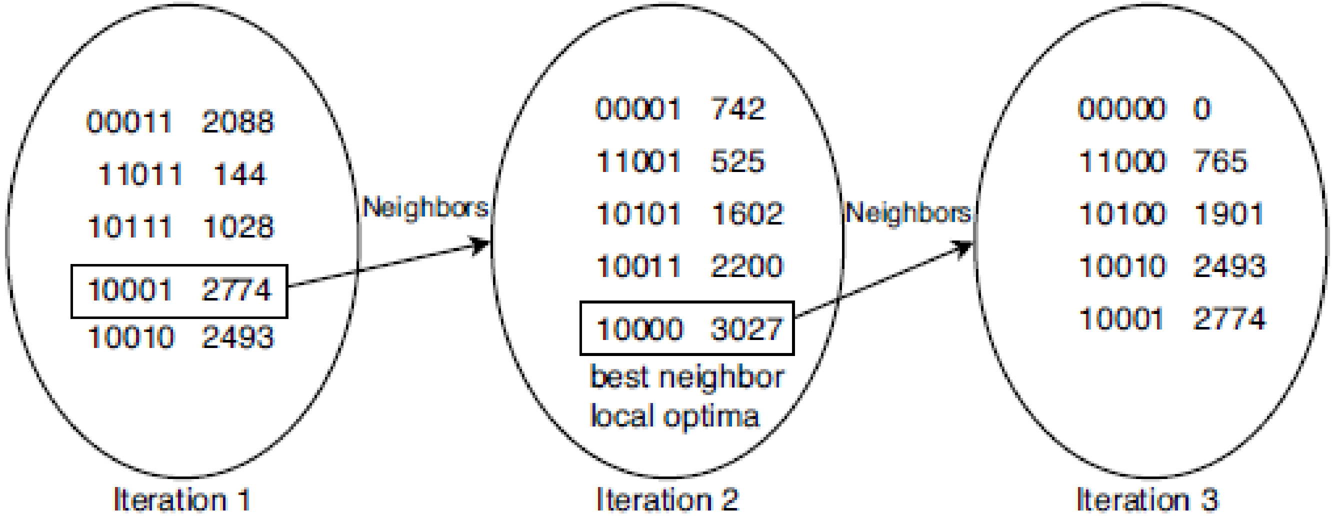
Local Search

- It is also called *hill climbing*, *descent*, *iterative improvement*, etc.
- It is likely the oldest and simplest metaheuristic method.
- It starts at a given initial solution.
- At each iteration, the heuristic *replaces* the current solution by a neighbor that *improves* the objective function.
- It stops when all candidate neighbors are worse than the current solution, i.e., a local minimum is reached.



LS Example

- Maximize $x^3 - 60x^2 + 900x$, x is discrete



- Local search process using a binary representation of solutions, a 1-flip move operator, and the best neighbor selection strategy.
- The global optimal solution is $f([01010]_2) = f(10) = 4000$, while the final local optimal found is s = [10000], starting from the solution s_0=[10001]

How LS Works

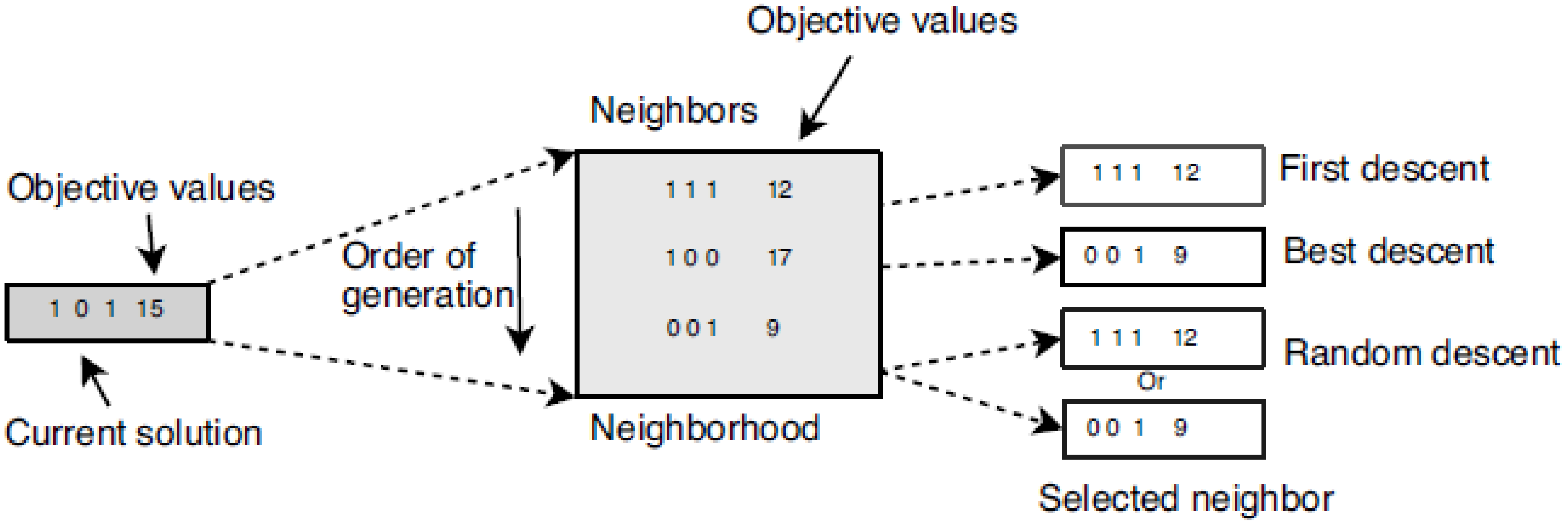
- LS may be seen as a descent walk in the neighborhood graph $G=(S, V)$ representing the search space.
  - S represents the set of all feasible solutions.
  - V represents the neighborhood relation.
  - Each edge (i, j) in the graph will connect any neighboring s_i and s_j.
  - For a given solution s, the number of associated edges will be $|N(s)|$.

Template of a local search algorithm.

$s = s_0$; /* Generate an initial solution s_0 */
While not Termination_Criterion **Do**
  Generate $(N(s))$; /* Generation of candidate neighbors */
  If there is no better neighbor **Then** Stop ;
  $s = s'$; /* Select a better neighbor $s' \in N(s)$ */
Endwhile
Output Final solution found (local optima).

How LS Works

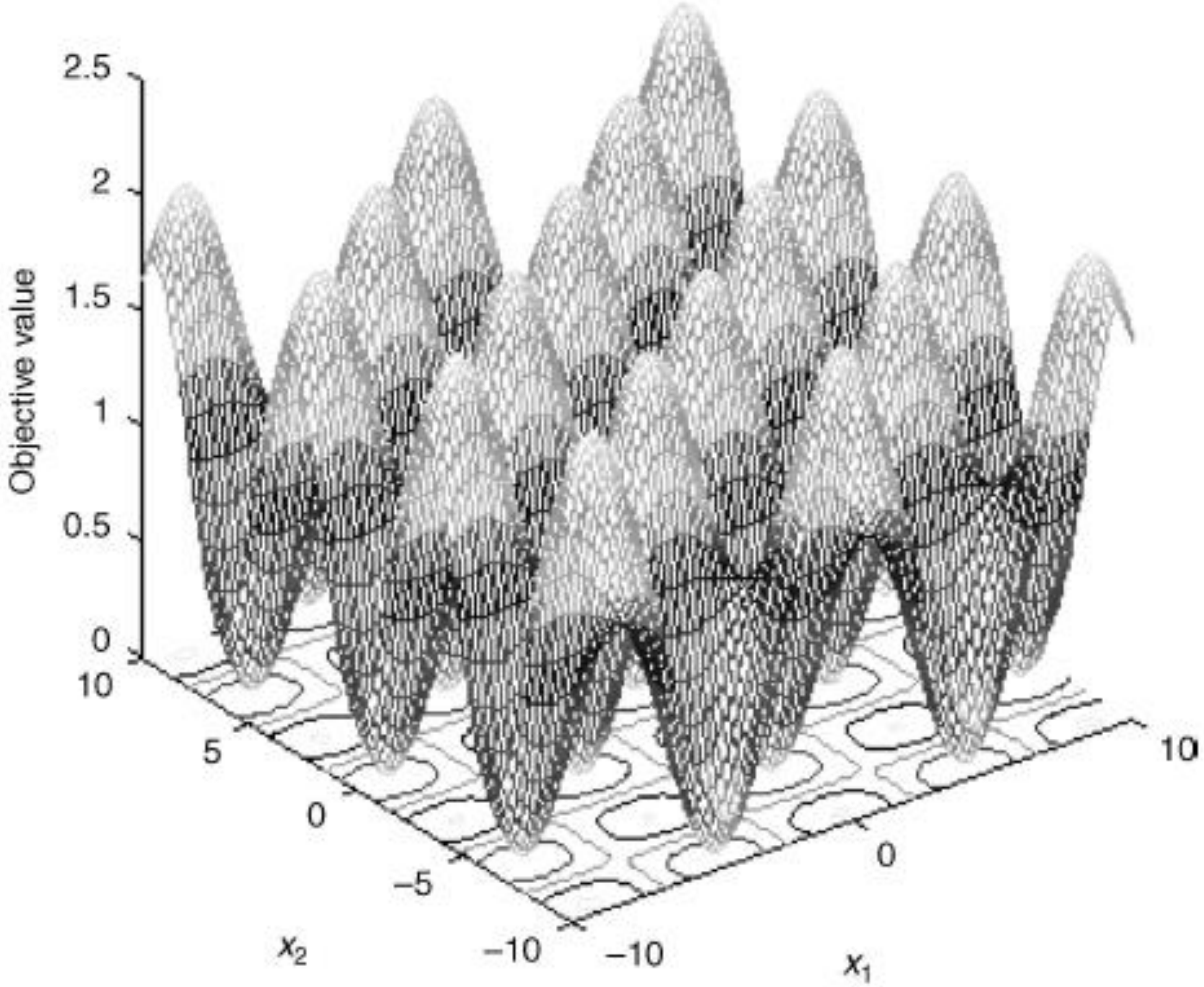
- Selection of the Neighbor
  - Best improvement
  - First improvement
  - Random selection



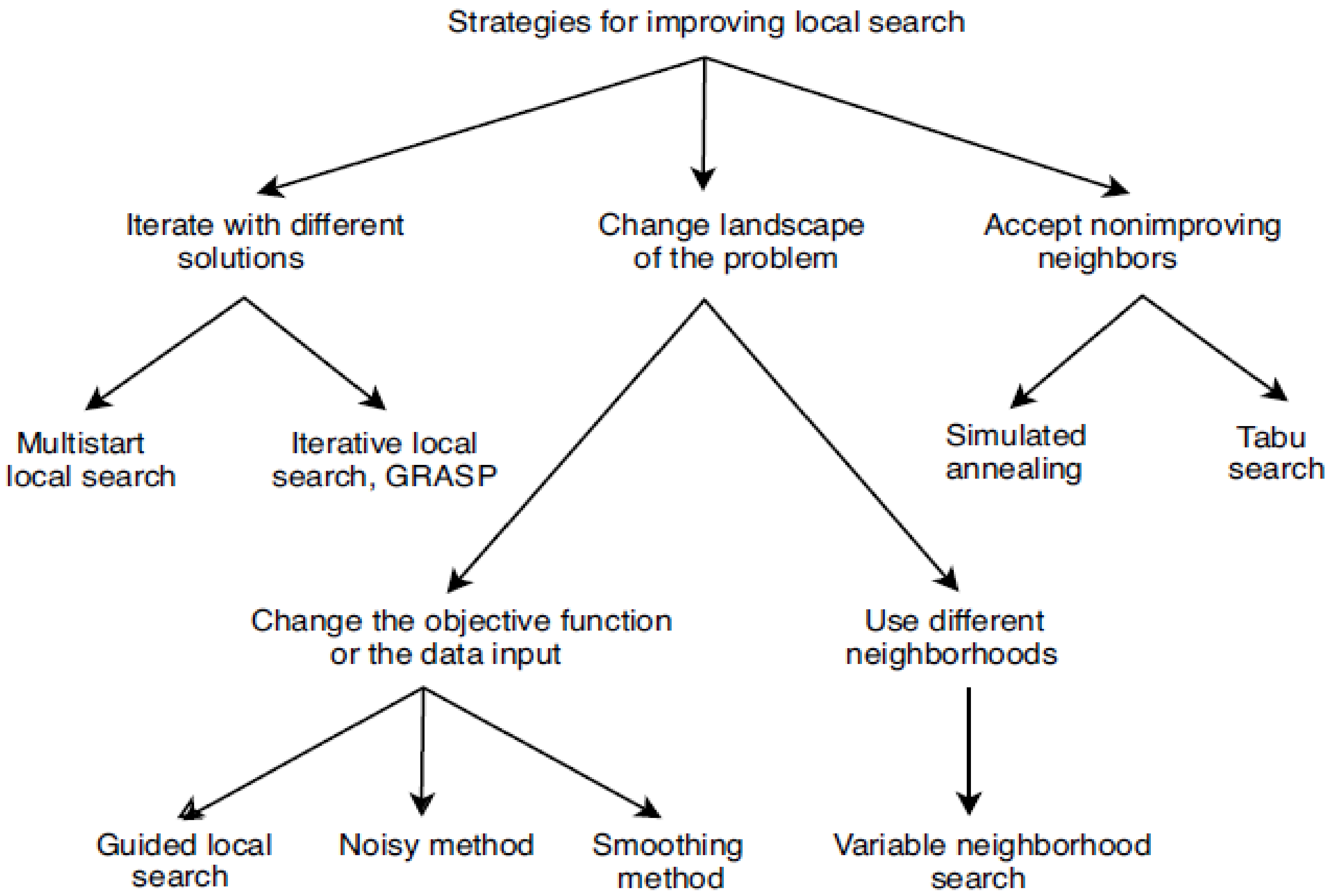
How LS Works

- Escaping from Local Optima
  - The LS is very sensitive to the initial solution.
  - No means to estimate the gap between the local optimum and the global optimum.
  - The number of iterations performed may not be known in advance.
  - Even if the LS runs very quickly, its worst case complexity is *exponential*.
  - Local search works well if there are not too many local optima.

Highly Multimodal Function



How to Avoid Local Optima



Thank you!