
How to solve it?

An invitation to metaheuristics

Andrea Roli

andrea.roli@unibo.it

DEIS

Alma Mater Studiorum Università di Bologna

How to solve it? An invitation to metaheuristics – p. 1

Prologue

Given a combinatorial optimization problem, the goal of a search algorithm is to find a (near-)optimal solution.

but

How to solve it? An invitation to metaheuristics – p. 2

Prologue

Given a combinatorial optimization problem, the goal of a search algorithm is to find a (near-)optimal solution.

How to solve it? An invitation to metaheuristics – p. 2

Prologue

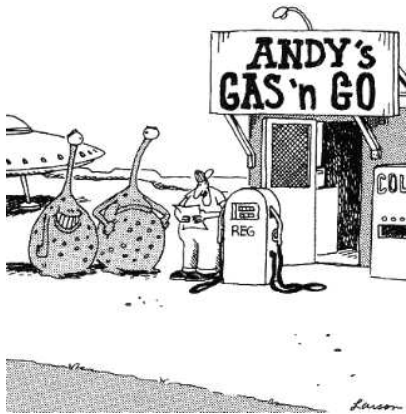
Given a combinatorial optimization problem, the goal of a search algorithm is to find a (near-)optimal solution.

but

How to decrease the probability of getting lost in the universe of feasible solutions?

How to solve it? An invitation to metaheuristics – p. 2

Prologue



“Shoot! You not only got the wrong planet, you got the wrong *solar* system. ... I mean, a wrong planet I can understand – but a whole solar system?”

Outline

- Combinatorial Optimization Problems
- Approximate algorithms
- Metaheuristics
 - Local search-based methods
 - Population-based metaheuristics
- Research issues

Goals

- Introduction to *metaheuristics*
 - Where we will get the intuition on how metaheuristics work
- Outline of ongoing research issues
 - Where we will get pointers to more technical/formal issues

COP

A **Combinatorial Optimization Problem** $\mathcal{P} = (\mathcal{S}, f)$ can be defined by:

- variables $X = \{x_1, \dots, x_n\}$;
- variable domains D_1, \dots, D_n ;
- constraints among variables;
- *Objective function* $f : D_1 \times \dots \times D_n \rightarrow \mathbb{R}^+$;
- The set of all possible feasible assignments $\mathcal{S} = \{s = \{(x_1, v_1), \dots, (x_n, v_n)\} \mid v_i \in D_i, s \text{ satisfies all the constraints}\}$

COP

Objective: find a solution $s^* \in \mathcal{S}$ with minimum objective function value, i.e., $f(s^*) \leq f(s) \forall s \in \mathcal{S}$.

COP

Objective: find a solution $s^* \in \mathcal{S}$ with minimum objective function value, i.e., $f(s^*) \leq f(s) \forall s \in \mathcal{S}$.

Many COPs are \mathcal{NP} -hard \Rightarrow no polynomial time algorithm exists (assuming $\mathcal{P} \neq \mathcal{NP}$)

Examples: Traveling salesman problem (TSP), quadratic assignment problem (QAP), maximum satisfiability problem (MAXSAT), timetabling and scheduling problems.

COP

Objective: find a solution $s^* \in \mathcal{S}$ with minimum objective function value, i.e., $f(s^*) \leq f(s) \forall s \in \mathcal{S}$.

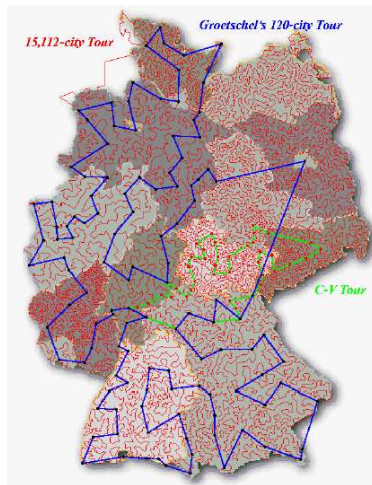
Many COPs are \mathcal{NP} -hard \Rightarrow no polynomial time algorithm exists (assuming $\mathcal{P} \neq \mathcal{NP}$)

TSP

Traveling Salesman Problem

Given an undirected graph, with n nodes and each arc associated with a positive value, find the Hamiltonian tour with the minimum total cost.

TSP



How to solve it? An invitation to metaheuristics – p. 9

Complete algorithms

Branch & bound, branch & cut, constraint programming approaches, ...

- Find an optimal solution in finite time (or return failure if the problem is infeasible)
- Disadvantage: for many applications are not efficient

How to solve it? An invitation to metaheuristics – p. 11

Solving algorithms

- Complete algorithms
- Approximate (or incomplete) algorithms

How to solve it? An invitation to metaheuristics – p. 10

Approximate algorithms

Heuristic alg., randomized alg., local search, metaheuristics, limited discrepancy search, ...

- No proof of optimality (if no solution exist, they do not terminate)
- Usually effective and efficient: they find (near-)optimal solutions efficiently

How to solve it? An invitation to metaheuristics – p. 12

Metaheuristics

- Approximate algorithms
- Applied to Combinatorial Optimization Problems and Constraint Satisfaction Problems
- Applied when:
 - Large size problems
 - The goal is to find a (near-)optimal solution quickly

How to solve it? An invitation to metaheuristics – p. 13

Metaheuristics

OBJECTIVE: Effectively and efficiently explore the search space

Ingredients:

- General strategies to balance *intensification* and *diversification*
- Use of *a priori* knowledge (heuristic)
- Exploit search history – adaptation
- Randomness and probabilistic choices

How to solve it? An invitation to metaheuristics – p. 14

Metaheuristics

OBJECTIVE: Effectively and efficiently explore the search space

How to solve it? An invitation to metaheuristics – p. 14

Etymology

Metaheuristic comes from the composition of two Greek words:

- *Heuristic* comes from *heuriskein* (εὕρισκειν): “to find”
- “meta” (μετά): “beyond, in an upper level”

How to solve it? An invitation to metaheuristics – p. 15

Metaheuristics

Encompass and combine:

- **Constructive methods** (e.g., random, heuristic, adaptive, etc.)
- **Local search-based methods** (e.g., Tabu Search, Simulated Annealing, Iterated Local Search, etc.)
- **Population-based methods** (e.g., Evolutionary Algorithms, Ant Colony Optimization, Scatter Search, etc.)

Heuristic construction

Use problem-specific knowledge (the *heuristic*) to construct a solution

Example: greedy algorithms on TSP – add the nearest city

Heuristic construction

Use problem-specific knowledge (the *heuristic*) to construct a solution

Heuristic construction

Use problem-specific knowledge (the *heuristic*) to construct a solution

Example: greedy algorithms on TSP – add the nearest city

Limit: myopic criterion (often solutions have poor quality)

Local search

The basic idea: start from a feasible solution and improve it by applying small (“local”) modifications.

Preliminary definitions

A **neighborhood structure** is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s .

A **locally minimal solution (or local minimum)** with respect to a neighborhood structure \mathcal{N} is a solution \hat{s} such that $\forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \leq f(s)$. We call \hat{s} a strict locally minimal solution if $f(\hat{s}) < f(s) \forall s \in \mathcal{N}(\hat{s})$.

Preliminary definitions

A **neighborhood structure** is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s .

Neighborhood: Examples

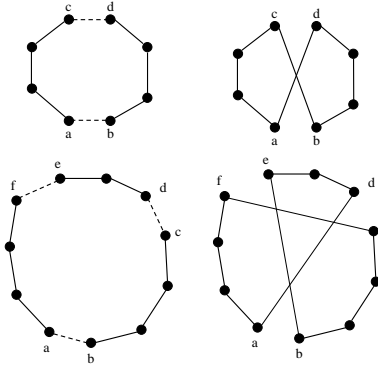
For problems defined on binary variables, the neighborhood can be defined on the basis of the Hamming distance (H_d) between two assignments. E.g.,

$$\mathcal{N}(s_i) = \{s_j \in \{0, 1\}^n \mid H_d(s_i, s_j) = 1\}$$

For example: $\mathcal{N}(000) = \{001, 010, 100\}$

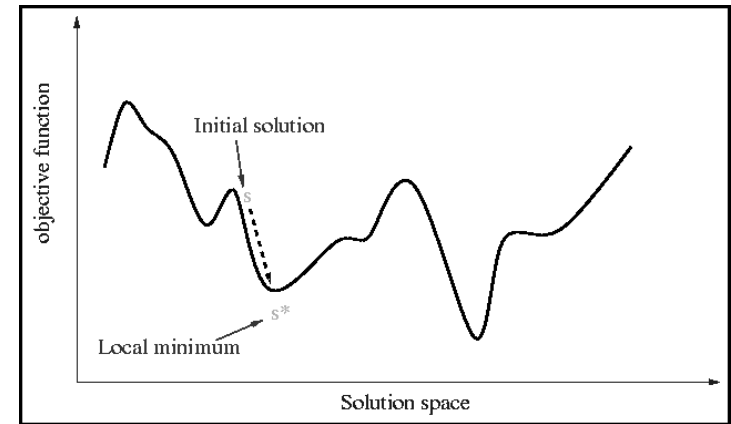
Neighborhood: Examples

In TSP, the neighborhood can be defined by means of arc exchanges on Hamiltonian tours



How to solve it? An invitation to metaheuristics – p. 21

A pictorial view



How to solve it? An invitation to metaheuristics – p. 23

Iterative Improvement

- Very basic local search
- A move is only performed if the solution it produces is better than the current solution (also called *hill-climbing*)
- The algorithm stops as soon as it finds a local minimum

How to solve it? An invitation to metaheuristics – p. 22

High-level algorithm

```
 $s \leftarrow \text{GenerateInitialSolution}()$   
repeat  
   $s \leftarrow \text{BestOf}(s, \mathcal{N}(s))$   
until no improvement is possible
```

How to solve it? An invitation to metaheuristics – p. 24

The fitness landscape

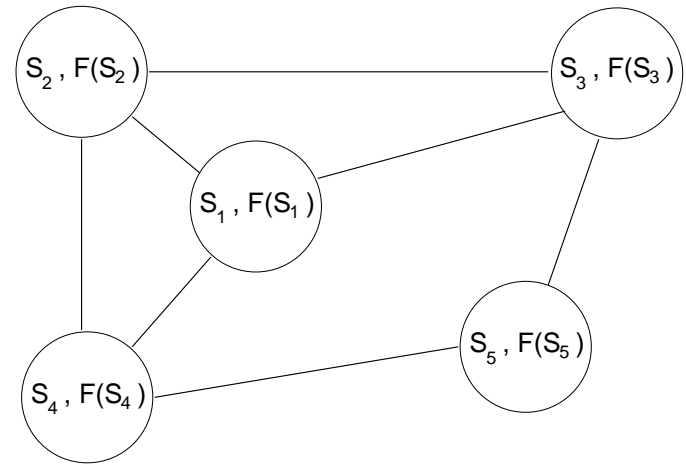
Defined by a triple:

$$\mathcal{L} = (S, \mathcal{N}, F)$$

- S is the set of solutions (or states);
- \mathcal{N} is the neighborhood function $\mathcal{N} : S \rightarrow 2^S$ that defines the neighborhood structure, by assigning to every $s \in S$ a set of states $\mathcal{N}(s) \subseteq S$.
- F is the objective function, in this specific case called *fitness function*, $F : S \rightarrow \mathbb{R}^+$.

How to solve it? An invitation to metaheuristics – p. 25

The fitness landscape



How to solve it? An invitation to metaheuristics – p. 27

The fitness landscape

- Metaheuristics can be seen as search processes in a graph
- The search starts from an initial node and explores the graph moving from a node to one of its neighbors, until it reaches a termination condition

How to solve it? An invitation to metaheuristics – p. 26

Escaping strategies...

Problem: Iterative Improvement stops at **local minima**, which can be very “poor”.

⇒ Strategies are required to prevent the search from getting trapped in local minima and to escape from them

How to solve it? An invitation to metaheuristics – p. 28

Three basic ideas

1) **Accept *up-hill* moves**

i.e., the search moves toward a solution with a *worse* objective function value

Three basic ideas

2) **Change neighborhood structure during the search**

Three basic ideas

1) **Accept *up-hill* moves**

i.e., the search moves toward a solution with a *worse* objective function value

Intuition: climb the hills and go downward in another direction

Three basic ideas

2) **Change neighborhood structure during the search**

Intuition: different neighborhoods generate different search space topologies

Three basic ideas

3) **Change the objective function so as to “fill-in” local minima**

Trajectory methods

- The search process is characterized by a trajectory in the search space
- The search process can be seen as the evolution in (discrete) time of a discrete dynamical system

Examples: Tabu Search, Simulated Annealing, Iterated Local Search, ...

Three basic ideas

3) **Change the objective function so as to “fill-in” local minima**

Intuition: modify the search space with the aim of making more “desirable” not yet explored areas

Simulated Annealing

Simulated Annealing exploits the first idea: *accept also up-hill moves*

- Origins in statistical mechanics (Metropolis algorithm)
- It allows moves resulting in solutions of worse quality than the current solution
- The probability of doing such a move is decreased during the search

Simulated Annealing

Simulated Annealing exploits the first idea: *accept also up-hill moves*

- Origins in statistical mechanics (Metropolis algorithm)
- It allows moves resulting in solutions of worse quality than the current solution
- The probability of doing such a move is decreased during the search

Usually, $p(\text{accept up-hill moves } s') = \exp\left(-\frac{f(s') - f(s)}{T}\right)$

How to solve it? An invitation to metaheuristics – p. 33

Cooling schedules

The temperature T can be varied in different ways:

- **Logarithmic:** $T_{k+1} = \frac{\Gamma}{\log(k+k_0)}$.
The algorithm is guaranteed to converge to the optimal solution with probability 1. Too slow for applications
- **Geometric:** $T_{k+1} = \alpha T_k$, where $\alpha \in]0, 1[$
- **Non-monotonic:** the temperature is decreased (intensification is favored), then increased again (to increase diversification)

How to solve it? An invitation to metaheuristics – p. 35

SA: High-level algorithm

```
s ← GenerateInitialSolution()
T ← T0
while termination conditions not met do
  s' ← PickAtRandom( $\mathcal{N}(s)$ )
  if  $f(s') < f(s)$  then
    s ← s' {s' replaces s}
  else
    Accept s' as new solution with probability  $p(T, s', s)$ 
  end if
  Update(T)
end while
```

How to solve it? An invitation to metaheuristics – p. 34

Tabu Search

Tabu Search exploits the second idea: *change neighborhood structure*.

- Explicitly exploits the search history to dynamically change the neighborhood to explore
- **Tabu list:** keeps track of recent visited solutions or moves and forbids them \Rightarrow escape from local minima and no cycling
- Many important concepts developed “around” the basic TS version (e.g., general exploration strategies)

How to solve it? An invitation to metaheuristics – p. 36

High-level algorithm

```
s ← GenerateInitialSolution()
TabuList ← ∅
while termination conditions not met do
  s ← ChooseBestOf( $s \cup \mathcal{N}(s) \setminus \text{TabuList}$ )
  Update(TabuList)
end while
```

How to solve it? An invitation to metaheuristics – p. 37

Tabu Search

Storing a list of solutions is often inefficient, therefore *moves* are stored instead.

BUT: storing moves we could cut good not yet visited solutions

How to solve it? An invitation to metaheuristics – p. 38

Tabu Search

Storing a list of solutions is often inefficient, therefore *moves* are stored instead.

How to solve it? An invitation to metaheuristics – p. 38

Tabu Search

Storing a list of solutions is often inefficient, therefore *moves* are stored instead.

BUT: storing moves we could cut good not yet visited solutions



we use *ASPIRATION CRITERIA* (e.g., accept a forbidden move toward a solution better than the current one)

How to solve it? An invitation to metaheuristics – p. 38

High-level algorithm

```
 $s \leftarrow \text{GenerateInitialSolution}()$   
 $\text{InitializeTabuLists}(TL_1, \dots, TL_r)$   
 $k \leftarrow 0$   
while termination conditions not met do  
   $\text{AllowedSet}(s, k) \leftarrow \{z \in \mathcal{N}(s) \mid \text{no tabu condition is violated or at least one aspiration condition is satisfied}\}$   
   $s \leftarrow \text{ChooseBestOf}(s \cup \text{AllowedSet}(s, k))$   
   $\text{UpdateTabuListsAndAspirationConditions}()$   
   $k \leftarrow k + 1$   
end while
```

How to solve it? An invitation to metaheuristics – p. 39

Guided Local Search

GLS penalizes solutions which contains some defined *features* (e.g., arcs in a tour, unsatisfied clauses, etc.)

If feature i is present in solution s , then $I_i(s) = 1$, otherwise $I_i(s) = 0$

How to solve it? An invitation to metaheuristics – p. 41

Guided Local Search

GLS exploits the third idea: *dynamically change the objective function*.

- Basic principle: help the search to move out gradually from local optima by changing the search landscape
- The objective function is dynamically changed with the aim of making the current local optimum “less desirable”

How to solve it? An invitation to metaheuristics – p. 40

Guided Local Search

Each feature i is associated a *penalty* p_i which weights the importance of the features.

The objective function f is modified so as to take into account the penalties.

How to solve it? An invitation to metaheuristics – p. 42

Guided Local Search

Each feature i is associated a *penalty* p_i which weights the importance of the features.

The objective function f is modified so as to take into account the penalties.

$$f'(s) = f(s) + \lambda \sum_{i=1}^m p_i \cdot I_i(s)$$

High-Level Algorithm

```
s ← GenerateInitialSolution()
while termination conditions not met do
  s ← LocalSearch(s, f')
  for all selected features  $i$  do
     $p_i \leftarrow p_i + 1$ 
  end for
  Update( $f'$ ,  $\mathbf{p}$ ) {where  $\mathbf{p}$  is the penalty vector}
end while
```

Guided Local Search

Each feature i is associated a *penalty* p_i which weights the importance of the features.

The objective function f is modified so as to take into account the penalties.

$$f'(s) = f(s) + \lambda \sum_{i=1}^m p_i \cdot I_i(s)$$

λ scales the contribution of the penalties wrt to the original objective function

Lessons learnt

- The effectiveness of a metaheuristic strongly depends on the dynamical interplay of intensification and diversification
- General search strategies have to be applied to effectively explore the search space
- The use of search history characterizes the nowadays most effective algorithms
- Optimal parameter tuning is crucial and sometimes very difficult to achieve

Trajectory methods

Other important trajectory methods:

- Variable neighborhood search (along with variants)
- Iterated local search

Population-based methods

- Population-based metaheuristics perform search processes which describes the evolution of a set of points in the search space.
- Some are inspired by natural processes, such as natural evolution and social insects foraging behavior.

Population-based methods

- Population-based metaheuristics perform search processes which describes the evolution of a set of points in the search space.

Population-based methods

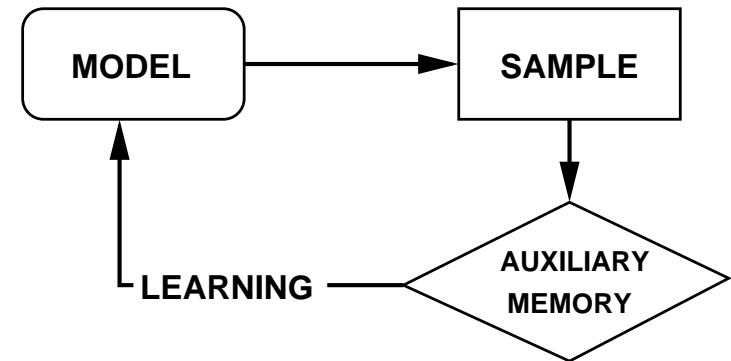
- Population-based metaheuristics perform search processes which describes the evolution of a set of points in the search space.
- Some are inspired by natural processes, such as natural evolution and social insects foraging behavior.
- Basic principle: *learning* correlations between “good” solution components

Population-based methods

- Evolutionary Algorithms
 - Evolutionary Programming
 - Evolution Strategies
 - Genetic Algorithms
- Ant Colony Optimization
- Scatter Search
- Population-Based Incremental Learning
- Estimation of Distribution Algorithms

How to solve it? An invitation to metaheuristics – p. 47

The basic principle



How to solve it? An invitation to metaheuristics – p. 49

The basic principle

Model-based search: *Candidate solutions are generated using a parametrized probabilistic model, updated using the previously seen solutions in such a way that the search will concentrate in the regions containing high quality solutions.*

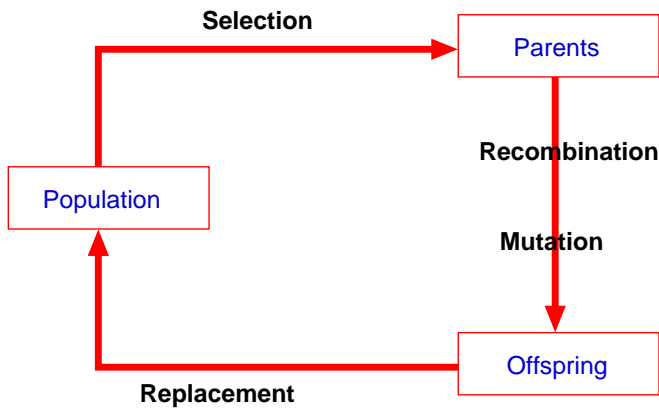
How to solve it? An invitation to metaheuristics – p. 48

Evolutionary Algorithms

- Inspired by Nature's capability to evolve living beings well adapted to their environment
- Computational models of evolutionary processes

How to solve it? An invitation to metaheuristics – p. 50

The Evolutionary Cycle



Ant Colony Optimization

Population-based metaheuristic inspired by the foraging behavior of ants. Ants can find the shortest path between the nest and a food source.

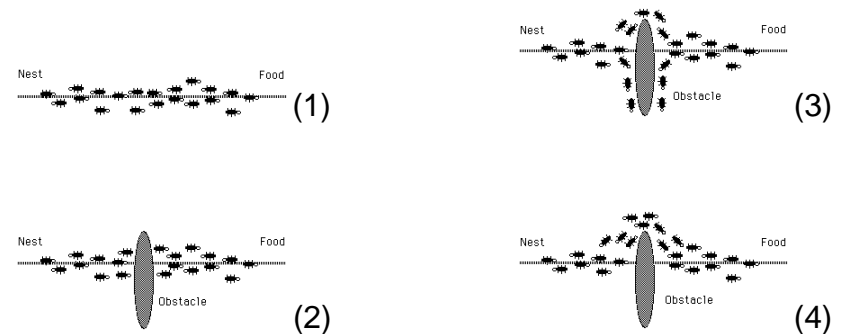
- While walking ants deposit a substance called *pheromone* on the ground.
- When they decide about a direction to go, they choose with higher probability paths that are marked by stronger pheromone concentrations.
- This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.

High-level algorithm

```

P ← GenerateInitialPopulation()
Evaluate(P)
while termination conditions not met do
  P' ← Recombine(P)
  P'' ← Mutate(P')
  Evaluate(P'')
  P ← Select(P'' ∪ P)
end while
  
```

Ant foraging behavior



Ant Colony Optimization

ACO algorithms are based on a parametrized probabilistic model – the *pheromone model* – that is used to model the chemical pheromone trails.

Artificial ants incrementally construct solutions by adding opportunely defined solution components to a partial solution under consideration

Artificial ants perform randomized walks on the *construction graph*: a completely connected graph $\mathcal{G} = (\mathcal{C}, \mathcal{L})$.

How to solve it? An invitation to metaheuristics – p. 55

Example

One possible TSP model for ACO:

- nodes of \mathcal{G} (the components) are the cities to be visited;
- states are partial or complete paths in the graph;
- a solution is an Hamiltonian tour in the graph;
- constraints are used to avoid cycles (an ant can not visit a city more than once).

How to solve it? An invitation to metaheuristics – p. 57

ACO construction graph

$$\mathcal{G} = (\mathcal{C}, \mathcal{L})$$

- vertices are the solution components \mathcal{C}
- \mathcal{L} are the connections
- *states* are paths in \mathcal{G} .

Solutions are *states*, i.e., encoded as paths on \mathcal{G}

Constraints are also provided in order to construct feasible solutions

How to solve it? An invitation to metaheuristics – p. 56

Sources of information

- Connections, components (or both) can have associated **pheromone** trail and **heuristic** value.
- **Pheromone** trail takes the place of natural pheromone and encodes a long-term memory about the whole ants' search process
- **Heuristic** represents a priori information about the problem or dynamic heuristic information (in the same way as static and dynamic heuristics are used in constructive algorithms).

How to solve it? An invitation to metaheuristics – p. 58

Ant system

- First ACO example
- Ants construct a solution by building a path along the construction graph
- The *transition rule* is used to choose the next node to add
- Both heuristic and pheromone are used
- The pheromone values are updated on the basis of the quality of solutions built by the ants

How to solve it? An invitation to metaheuristics – p. 59

Ant System

Pheromone update rule:

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

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

ρ is the evaporation coefficient; L_k is the length of the tour built by ant k .

How to solve it? An invitation to metaheuristics – p. 61

Ant system

The probability of moving from city i to city j for ant k is:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{k \in \text{feasible}_k} [\tau_{ik}]^\alpha [\eta_{ik}]^\beta} & \text{if } j \in \text{feasible}_k \\ 0 & \text{otherwise} \end{cases}$$

α e β weight the relative influence of pheromone and heuristic

How to solve it? An invitation to metaheuristics – p. 60

High-level algorithm

```
while termination conditions not met do  
  ScheduleActivities  
    AntBasedSolutionConstruction()  
    PheromoneUpdate()  
    DaemonActions() {optional}  
  end ScheduleActivities  
end while
```

How to solve it? An invitation to metaheuristics – p. 62

Research lines

- Algorithm behavior
 - Theoretical approach (markov, dynamical systems, landscape properties)
 - Empirical approach (scientific method, statistics)
- Problem structure vs. algorithm behavior
- Integration with complete algorithms
- Software engineering approach (tools, multi-agent systems)
- Parallelization

How to solve it? An invitation to metaheuristics – p. 63

Dynamical systems

- More complex dynamics
- Basins of attraction → are optima reachable? Which is the probability to reach them from a random initial state (heuristic solution)?

How to solve it? An invitation to metaheuristics – p. 65

Dynamical systems

Execution of an algorithm \leftrightarrow dynamics of a (stochastic) dynamical system

- Attractors \leftrightarrow *stagnation*
 - Local minimum: fixed point
 - “Trap”: cyclic attractor
 - ?????: chaotic attractor

How to solve it? An invitation to metaheuristics – p. 64

Dynamical systems

Advantages:

- Convergence proofs
- Estimation of *completeness* probability
- Dynamic parameter tuning (no more rule of thumbs...)

How to solve it? An invitation to metaheuristics – p. 66

Problem structure vs. algorithm behavior

The impact of *structure* – whatever it is – on search algorithms is relevant, especially for the so-called ‘real-world problems’.

- Identify most difficult instances (for a given algorithm)
- Understand *why* an instance is difficult
- Exploit this information to choose the best solver, or a combination of solvers
- Evaluate the quality of benchmarks

How to solve it? An invitation to metaheuristics – p. 67

Structure

- Diverse meanings
 - *Structure vs. random*
 - Usually *real world* problems are said to be structured
 - Attempts to define quantitative measures (entropy, compression ratio, etc.)
- ▶ Graph representation of relations among problem entities

How to solve it? An invitation to metaheuristics – p. 68

Structure

- Diverse meanings
- *Structure vs. random*
- Usually *real world* problems are said to be structured
- Attempts to define quantitative measures (entropy, compression ratio, etc.)

How to solve it? An invitation to metaheuristics – p. 68

Graph prop. vs search

- Node degree distribution & ‘multi-flip’ local search
- Small-world & instance hardness

How to solve it? An invitation to metaheuristics – p. 69

Metaheuristics and systematic methods

1. Metaheuristics are applied before systematic methods, providing a valuable input, or vice versa.
2. Metaheuristics use CP and/or tree search to efficiently explore the neighborhood.
3. A “tree search”-based algorithm applies a metaheuristic in order to improve a solution (i.e., a leaf of the tree) or a partial solution (i.e., an inner node). Metaheuristic concepts can also be used to obtain incomplete but efficient tree exploration strategies.