How to solve it?

An invitation to metaheuristics

Andrea Roli

`a.roli@unich.it`

Dipartimento di Scienze
Università degli Studi “G. D’Annunzio”
Chieti-Pescara

Prologue

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

Prologue

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

but

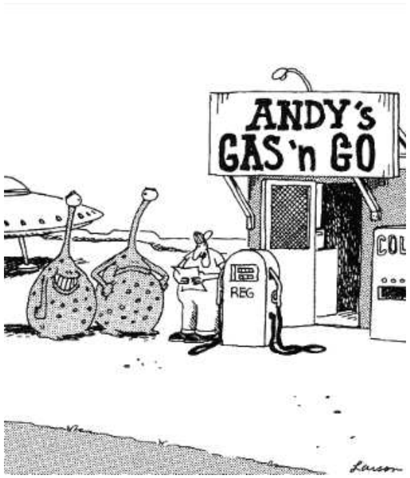
Prologue

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?

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?"

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

---

Outline

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

COP

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

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

COP

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

COP

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

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., $f(s^*) \leq f(s)\ \forall s \in S$.

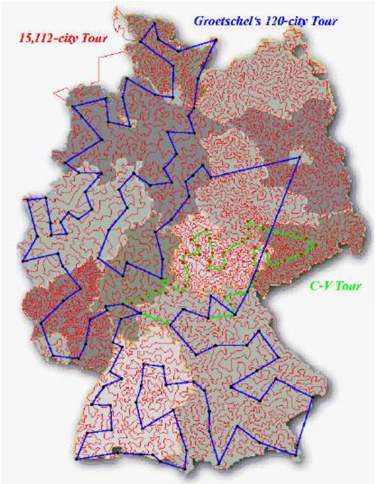
Many COPs are $\mathcal{NP}$-hard $\Rightarrow$ no polynomial time algorithm exists (assuming $\mathcal{P} \neq \mathcal{NP}$)

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



Solving algorithms

- Complete algorithms
- Approximate (or incomplete) algorithms

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

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

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

Metaheuristics

OBJECTIVE: Effectively and efficiently explore the search space

---

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

Etymology

Metaheuristic comes from the composition of two Greek words:

- *Heuristic* comes from *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$): “to find”
- “meta” ($\mu\epsilon\tau\alpha$): “beyond, in an upper level”

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.

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 $f(\hat{s}) < f(s)\ \forall\ s \in \mathcal{N}(\hat{s})$.

Neighborhood: Examples

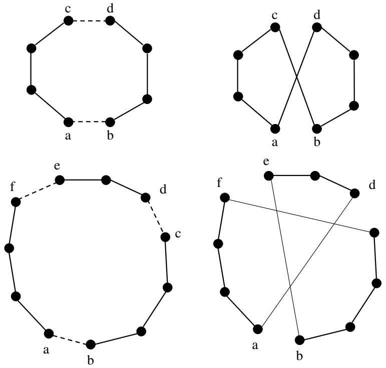
For problems defined on binary variables, the neighborhood can be defined on the basis of the Hamming distance (H_d) between two assignments. E.g.,

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

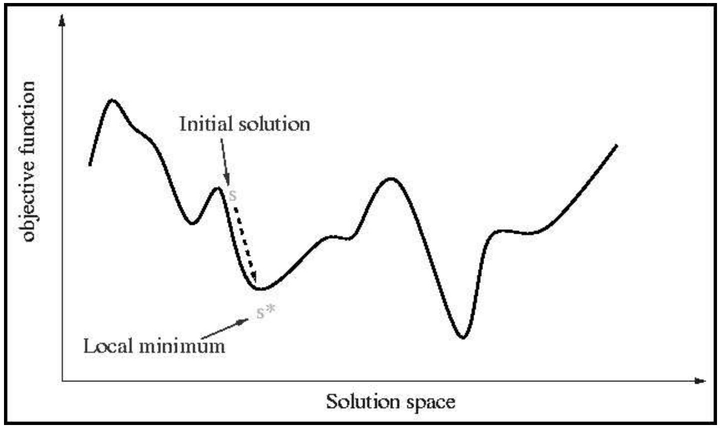
For example: $\mathcal{N}(000) = \{001, 010, 100\}$

Neighborhood: Examples

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



A pictorial view



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

High-level algorithm

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

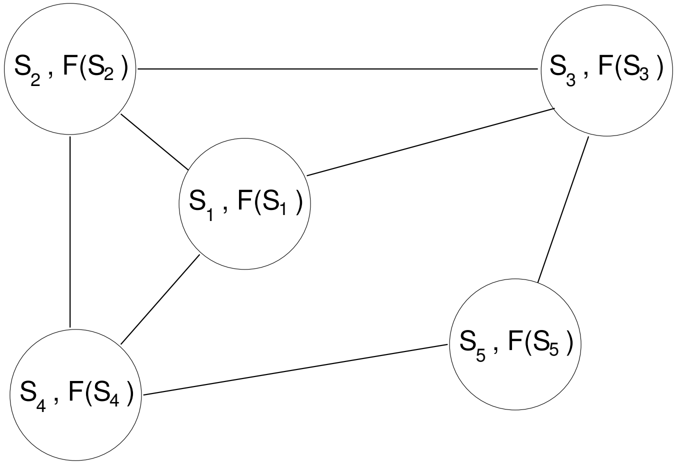
The fitness landscape

Defined by a triple:

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

- *S* is the set of solutions (or states);
- $\mathcal{N}$ is the neighborhood function $\mathcal{N} : S \to 2^S$ 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*: $S \to \mathbb{R}^+$.

The fitness landscape



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

---

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

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**

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

---

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, ...

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

SA: High-level algorithm

```
$s \leftarrow$ GenerateInitialSolution()
$T \leftarrow T_0$
while termination conditions not met do
    $s' \leftarrow$ PickAtRandom($\mathcal{N}(s)$)
    if $f(s') < f(s)$ then
        $s \leftarrow s'$ {$s'$ replaces $s$}
    else
        Accept $s'$ as new solution with probability $p(T, s', s)$
    end if
    Update($T$)
end while
```

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 (intensifications is favored), then increased again (to increase diversification)

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 ⇒ escape from local minima and no cycling
- Many important concepts developed “around” the basic TS version (e.g., general exploration strategies)

High-level algorithm

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

Tabu Search

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

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

---

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

$$\Downarrow$$

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

High-level algorithm

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

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 $I_i(s) = 0$

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”

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.

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

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

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.

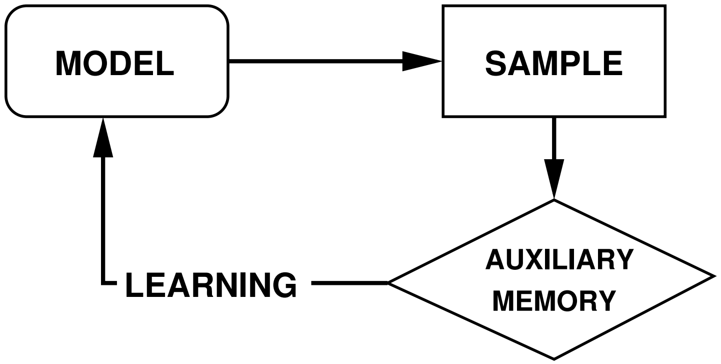
Population-based methods

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

The basic principle



MODEL

SAMPLE

AUXILIARY MEMORY

LEARNING

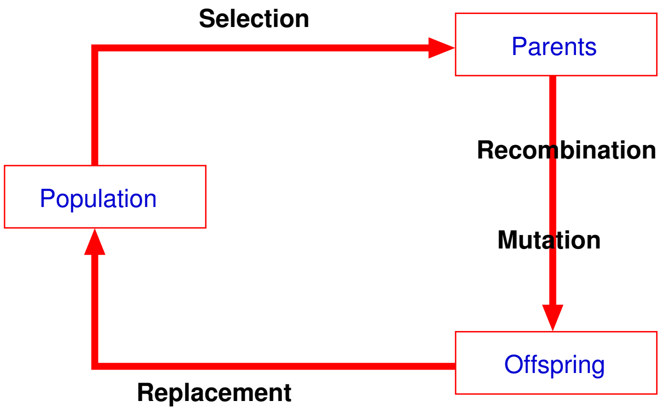
The basic principle

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.*

Evolutionary Algorithms

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

The Evolutionary Cycle



Selection

Parents

Recombination

Mutation

Population

Offspring

Replacement

High-level algorithm

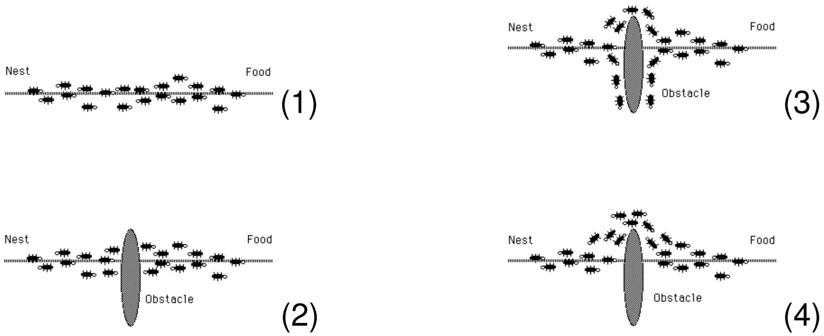
```
$P \leftarrow$ GenerateInitialPopulation()
Evaluate($P$)
while termination conditions not met do
    $P' \leftarrow$ Recombine($P$)
    $P'' \leftarrow$ Mutate($P'$)
    Evaluate($P''$)
    $P \leftarrow$ Select($P'' \cup P$)
end while
```

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.

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

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

---

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).

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).

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

---

Ant system

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

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

α e β weight the relative influence of pheromone and heuristic

---

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.

High-level algorithm

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

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

Dynamical systems

Execution of an algorithm ↔ dynamics of a (stochastic) dynamical system

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

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)?

Dynamical systems

Advantages:

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

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

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

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 prop. vs search

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

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.