An introduction to Metaheuristics

Andrea Roli

aroli@deis.unibo.it

Università degli Studi “G. D’Annunzio” – Chieti

DEIS - Università degli Studi di Bologna

Outline

- Definition of Metaheuristics
- Combinatorial Optimization Problems
- Trajectory Methods
- Population-based Methods
- Hybrid approaches

Metaheuristics

- Approximate algorithms: they do not guarantee to find the optimal solution in bounded time.
- Applied to Combinatorial Optimization Problems and Constraint Satisfaction Problems

Metaheuristics

- Approximate algorithms: they do not guarantee to find the optimal solution in bounded time.
- Applied to Combinatorial Optimization Problems and Constraint Satisfaction Problems
- Applied when:
  - Problems have large size
  - 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

Techniques:

- Use of the search history
- Adaptivity
- General strategies to balance intensification and diversification.

Metaheuristics

OBJECTIVE: Effectively and efficiently explore the search space

Techniques:

- Use of the search history
- Adaptivity
- General strategies to balance intensification and diversification.

► Sometimes, they are **erroneously** simply called “local search methods”.

Two-faced Janus

Intensification and Diversification are the driving forces of metaheuristic search.

Intensification: exploitation of the accumulated search experience (e.g., by concentrating the search in a confined, small search space area)

Diversification: exploration ’in the large‘ of the search space

Two-faced Janus

I&D are contrary and complementary: their dynamical balance determines the effectiveness of metaheuristics.

Two levels of I&D balance:

- Basic level: intrinsic exploration mechanism
- Strategic level: general criteria to guide the exploration of the search space

Other characteristics

- Metaheuristics are strategies that guide the search process.
- Metaheuristic algorithms are usually non-deterministic.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge as heuristic controlled by the upper level strategy.

Metaheuristics

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.)

A classification

We will classify metaheuristics in two basic classes:

- Trajectory methods
- Population-based methods

Other classifications are possible, based on different key concepts (e.g., the use of memory)

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, ...

Population-based methods

- Deal in every iteration of the algorithm with a set – a population – of solutions
- The search process can be seen as the evolution in (discrete) time of a set of points in the search space

Examples: Evolutionary Algorithms, Ant Colony Optimization, Scatter Search, ...

Combinatorial Optimization Problems

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

Combinatorial Optimization Problems

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.

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

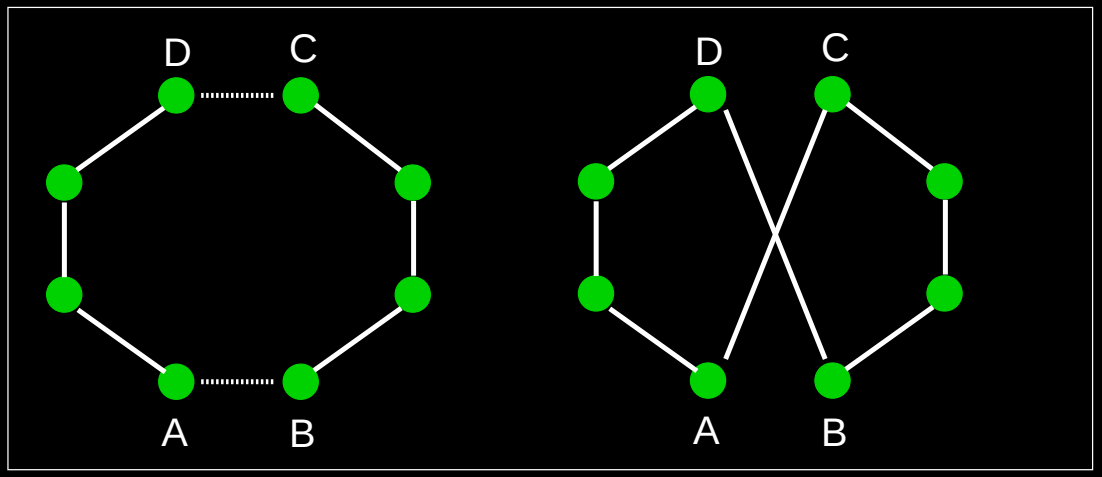
For problems defined on binary variables, the neighborhood can be defined on the basis of the Hamming distance (H_d) between two assignments. E.g.,

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

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. E.g.,



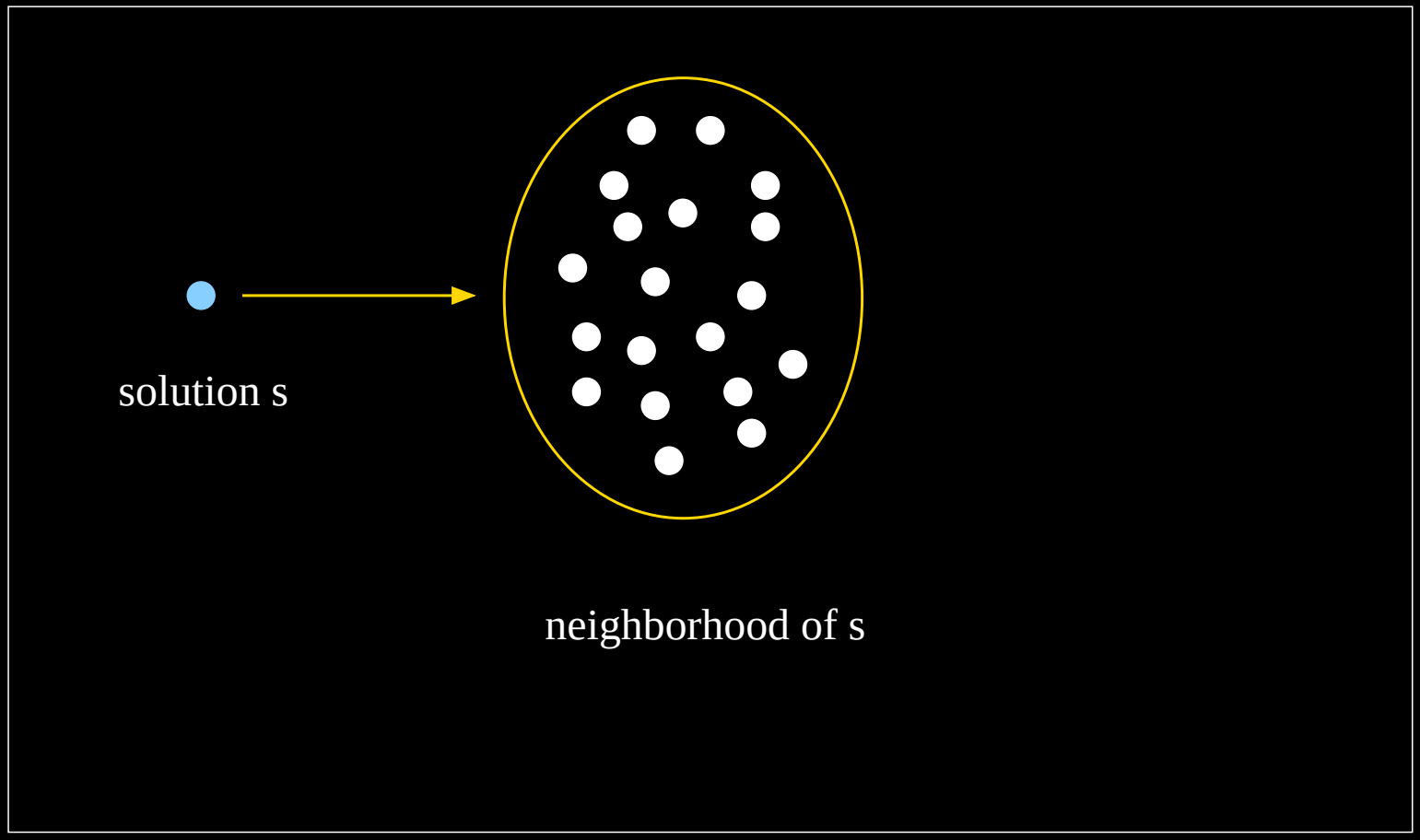
Trajectory methods

- Iterative Improvement
- Simulated Annealing
- Tabu Search
- Variable Neighborhood Search
- Guided Local Search
- Iterated Local Search

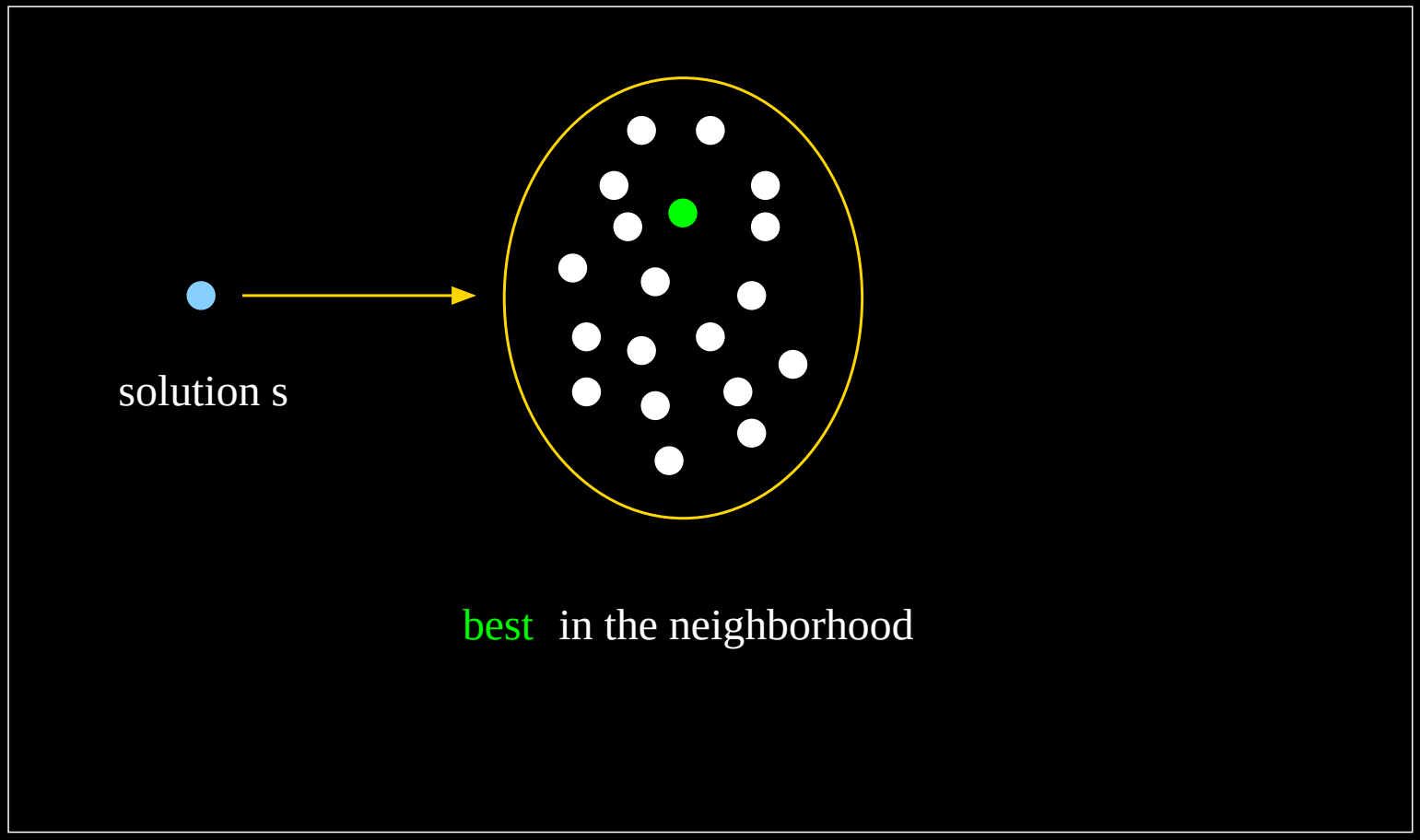
Iterative Improvement

- Very basic local search
- Each 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.

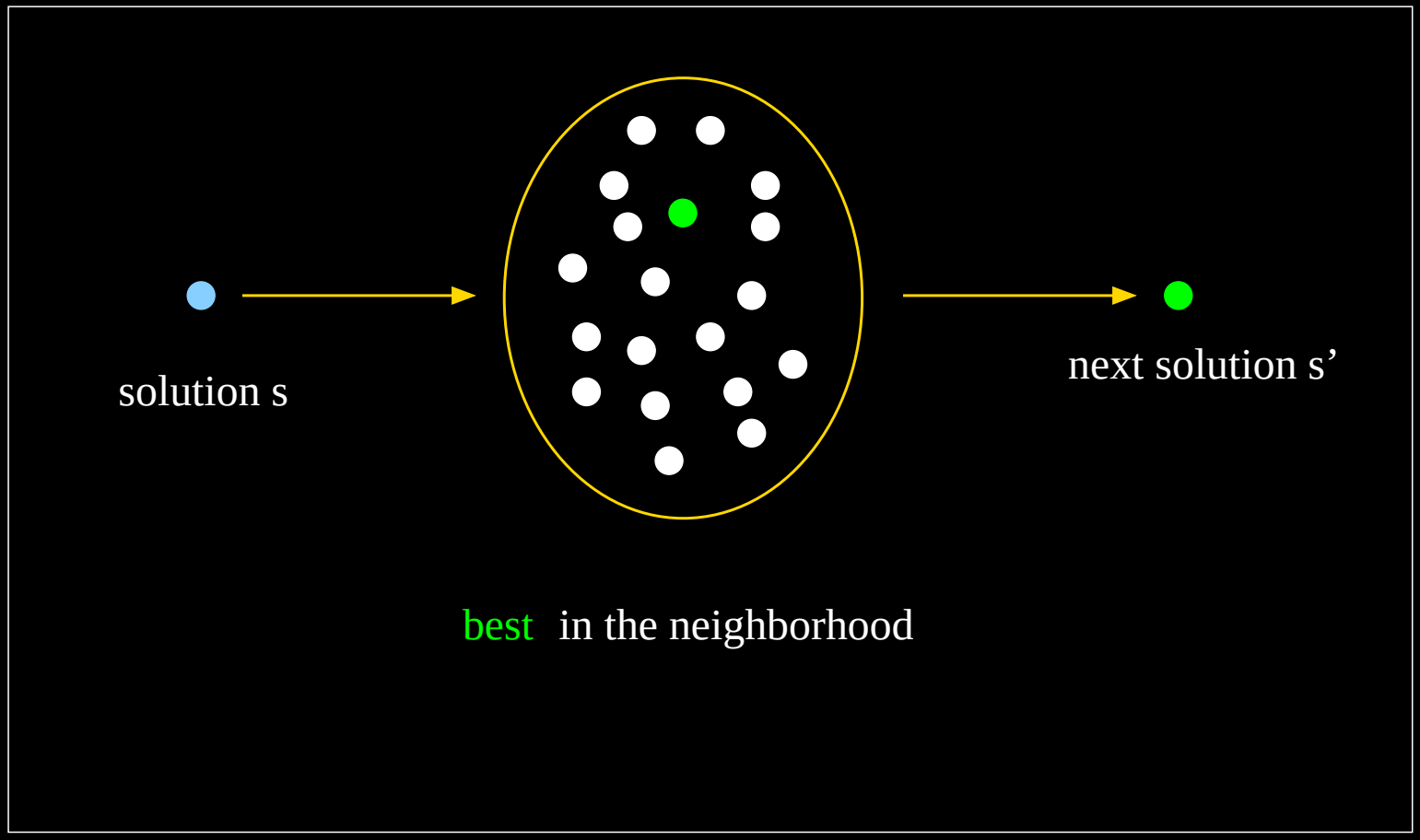
Iterative Improvement



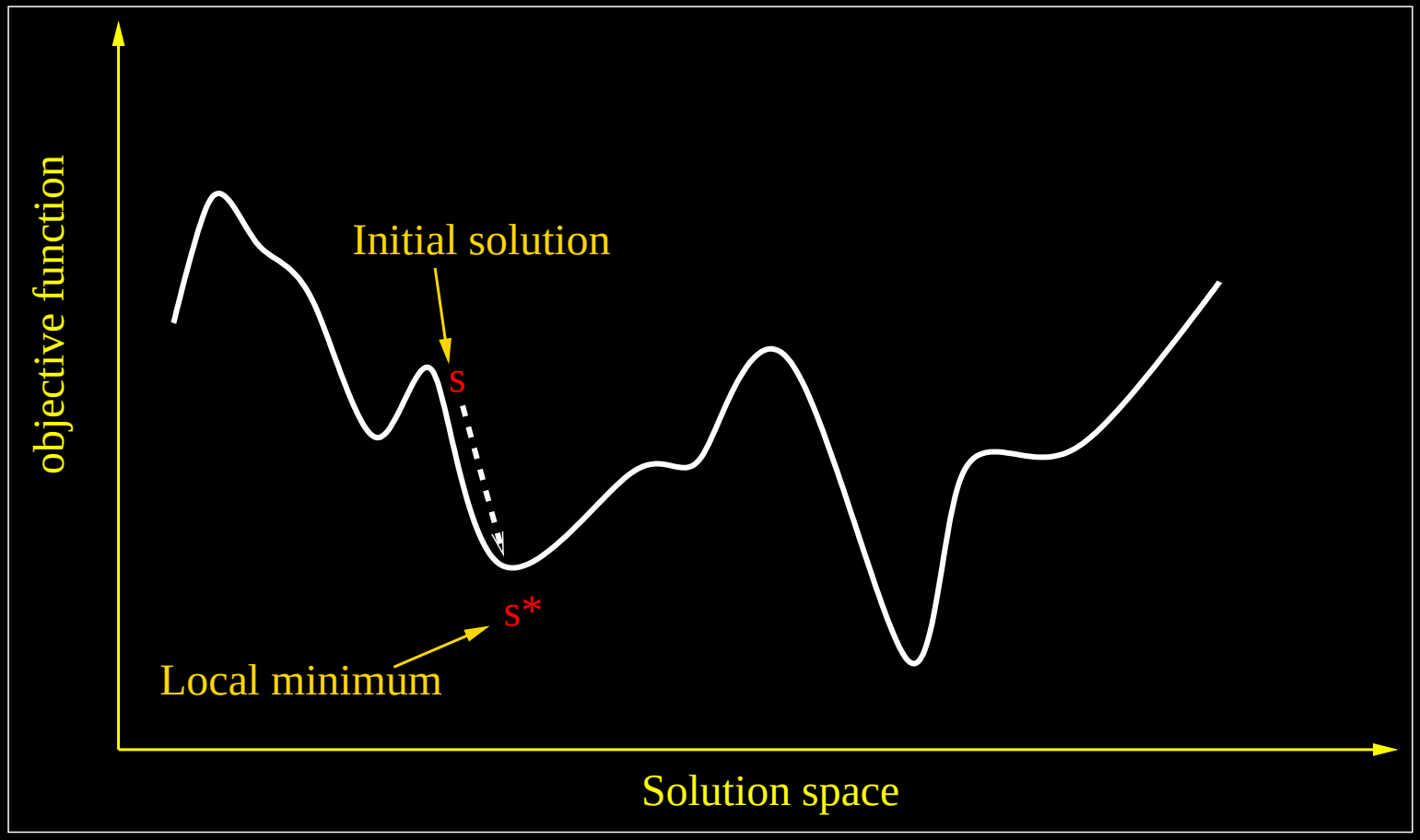
Iterative Improvement



Iterative Improvement



A pictorial view



The algorithm

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

Escaping strategies...

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

$\Rightarrow$ 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

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**

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

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.

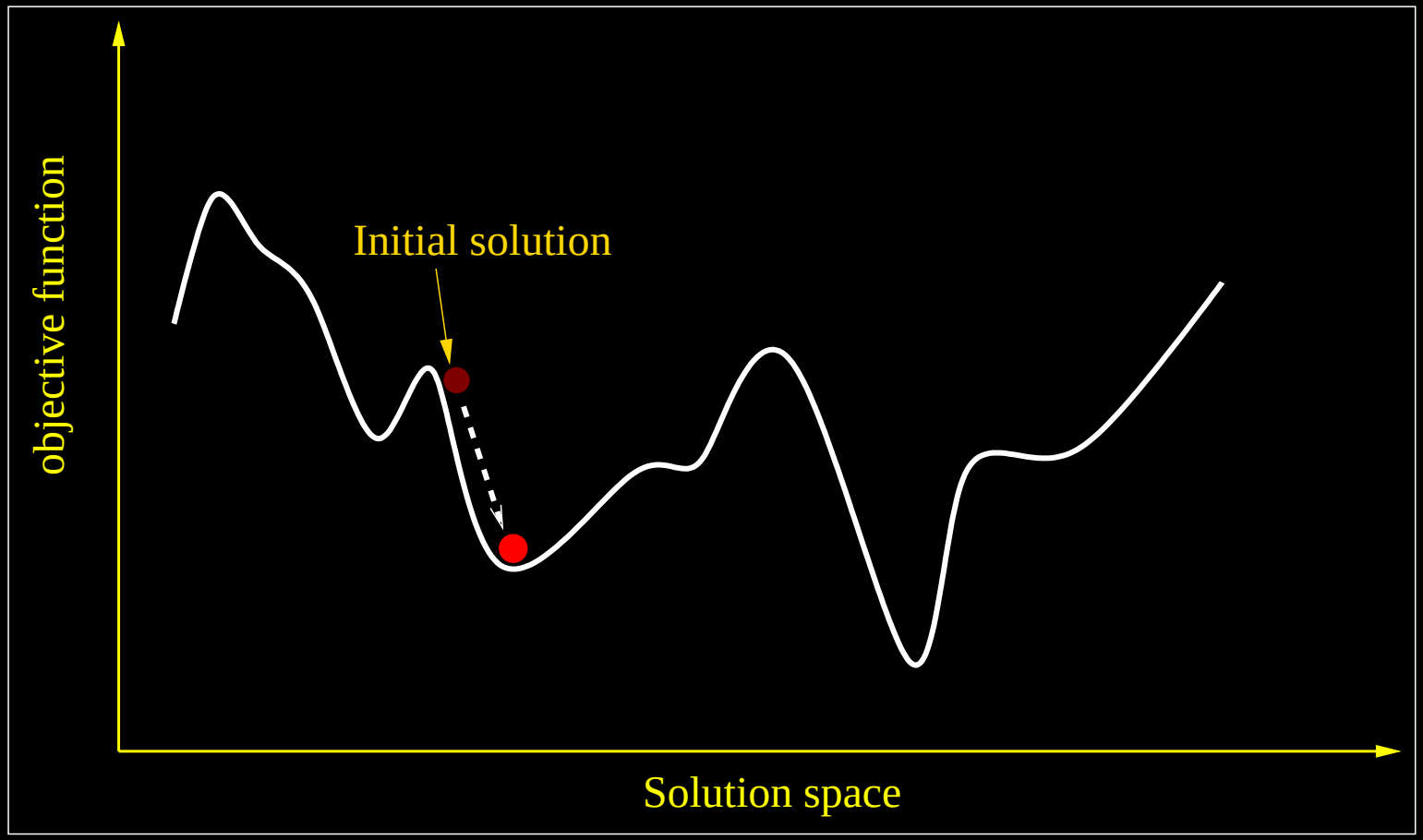
Simulated Annealing

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

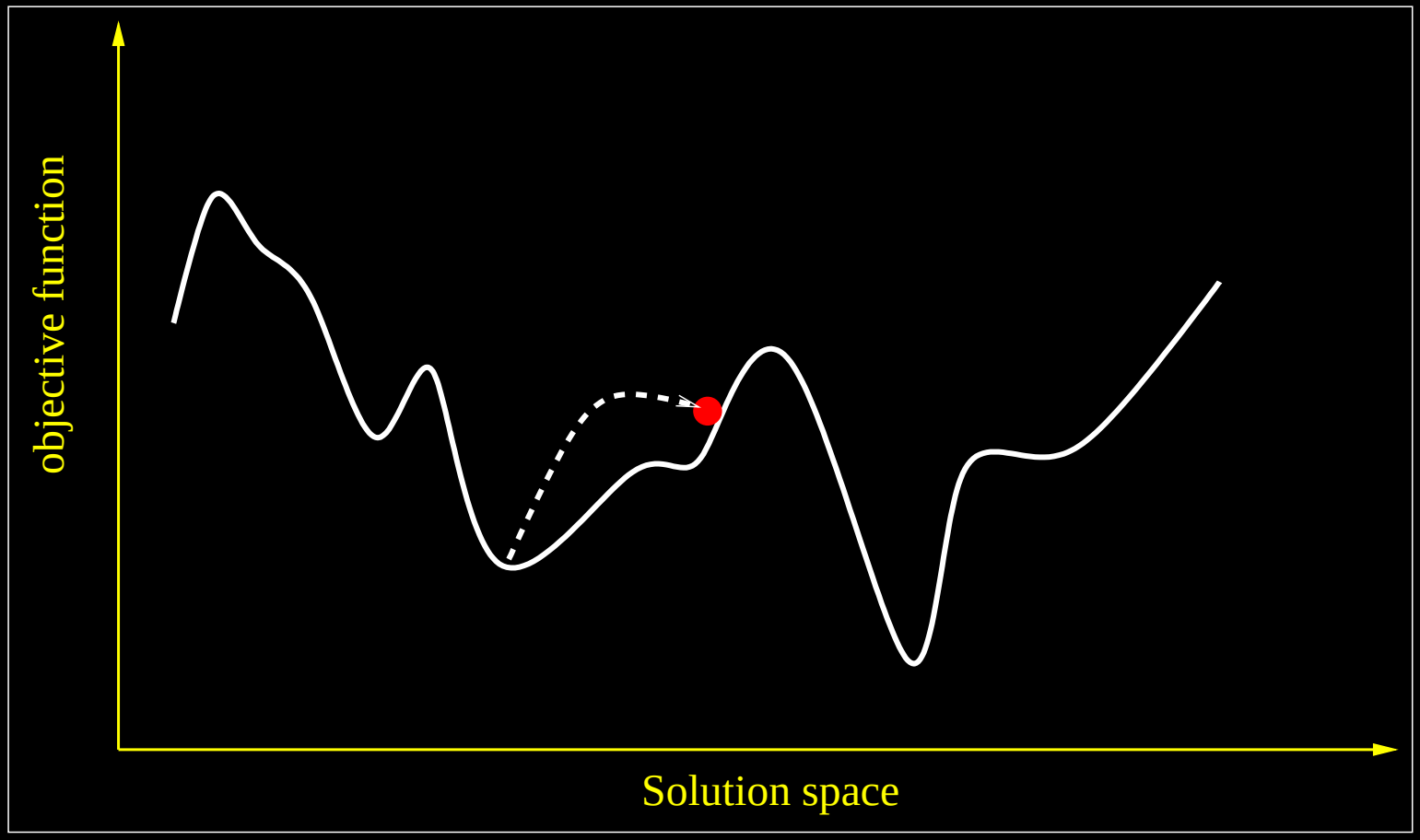
- Origins in statistical mechanics (Metropolis algorithm)
- It allows moves resulting in solutions of worse quality than the current solution
- The probability of doing such a move is decreased during the search.

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

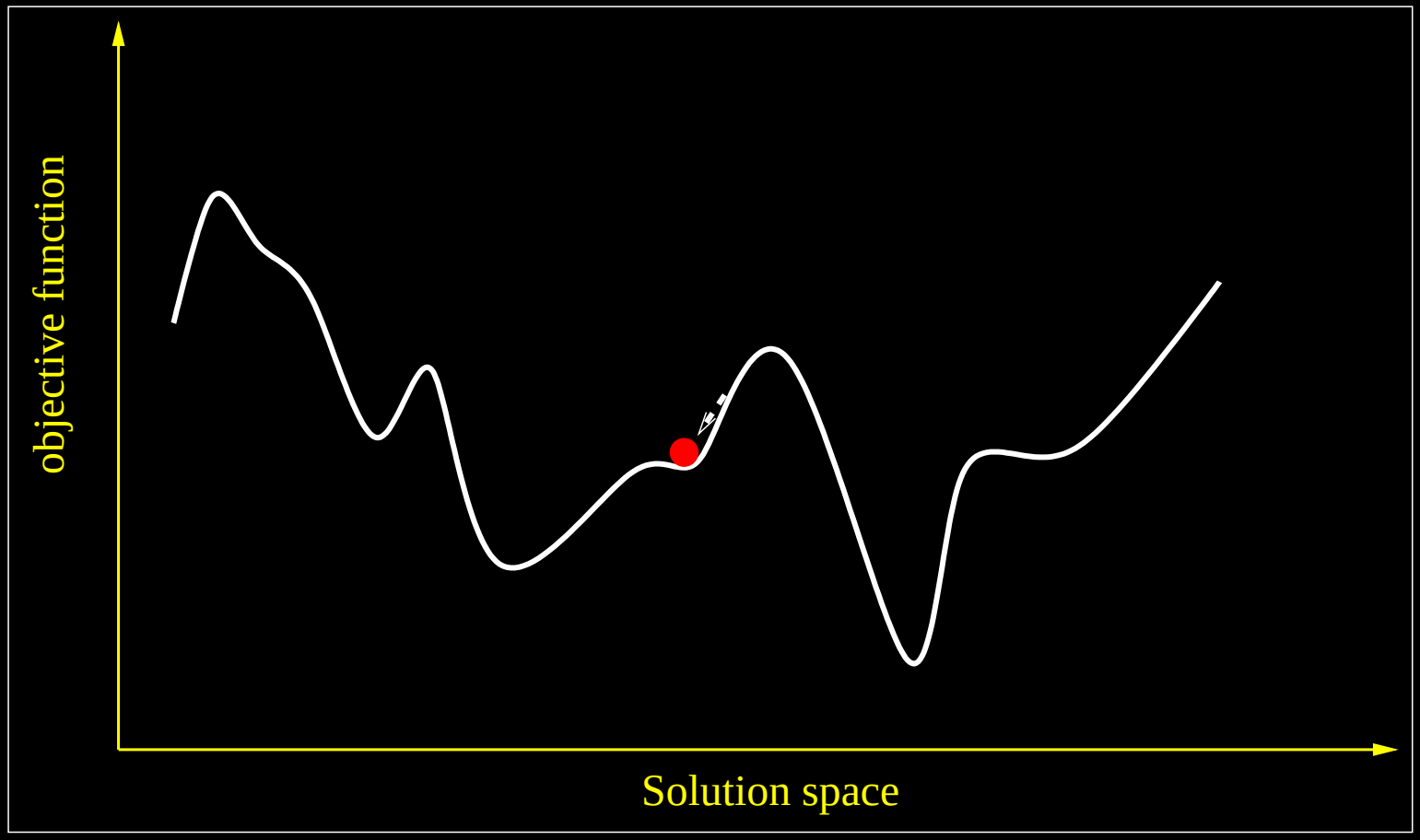
A pictorial view



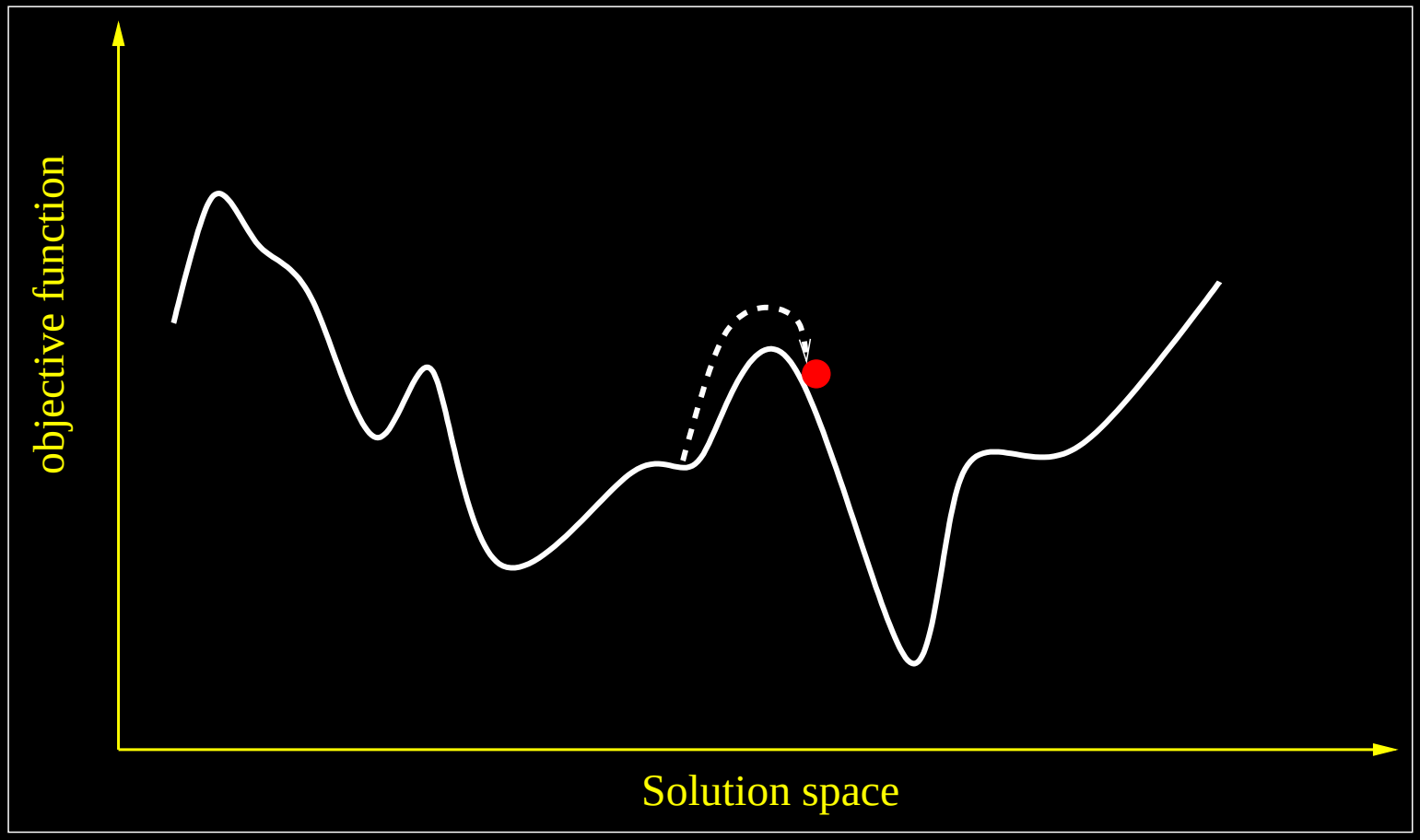
A pictorial view



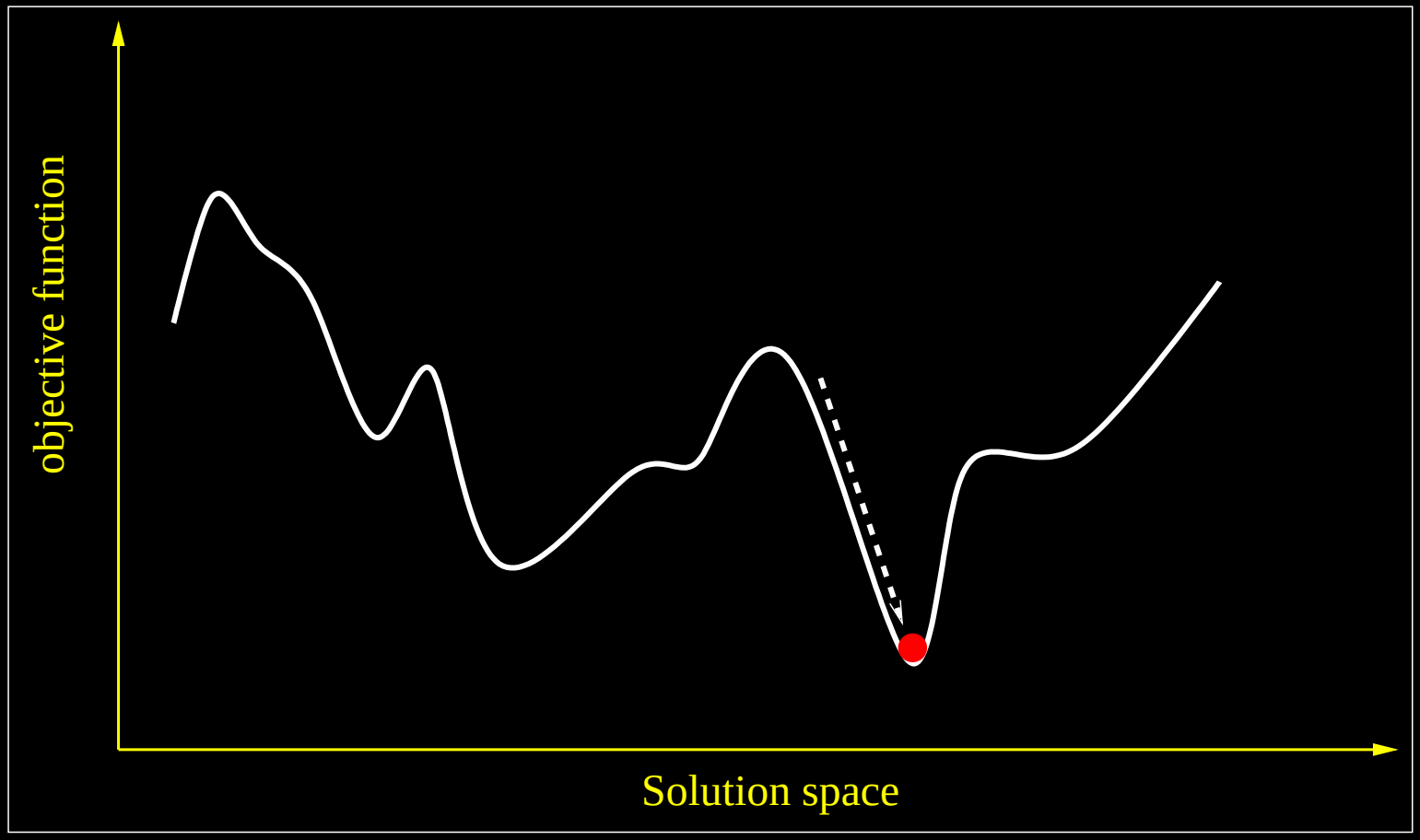
A pictorial view



A pictorial view



A pictorial view



SA: the 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).

Applications

SA is usually not very effective when used as stand-alone metaheuristic.

References:

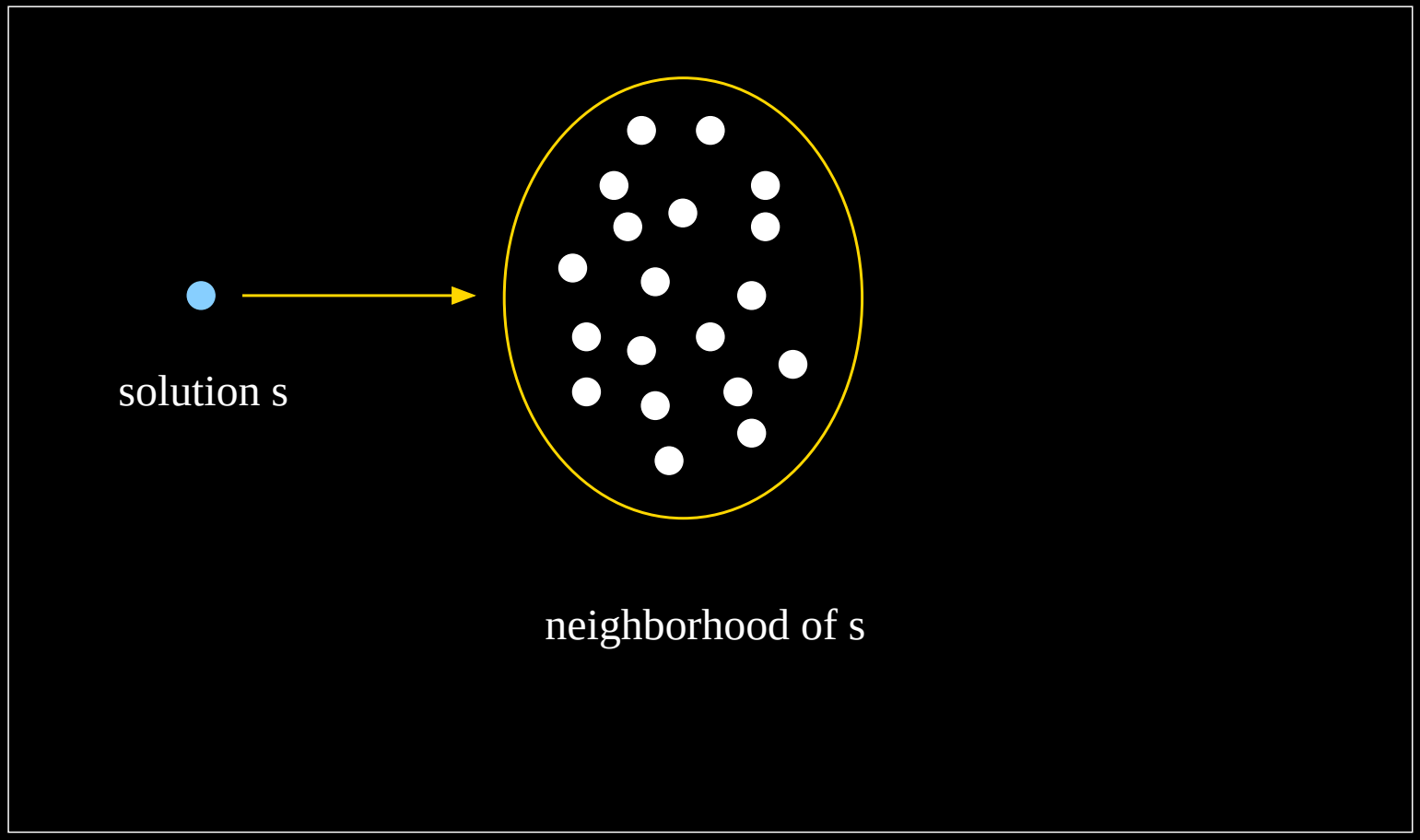
- E. H. L. Aarts and J. H. M. Korst and P. J. M. van Laarhoven, Simulated Annealing, in *Local Search in Combinatorial Optimization*, Wiley-Interscience, 1997.
- S. Kirkpartick and C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science* 13 May 1983.

Tabu Search

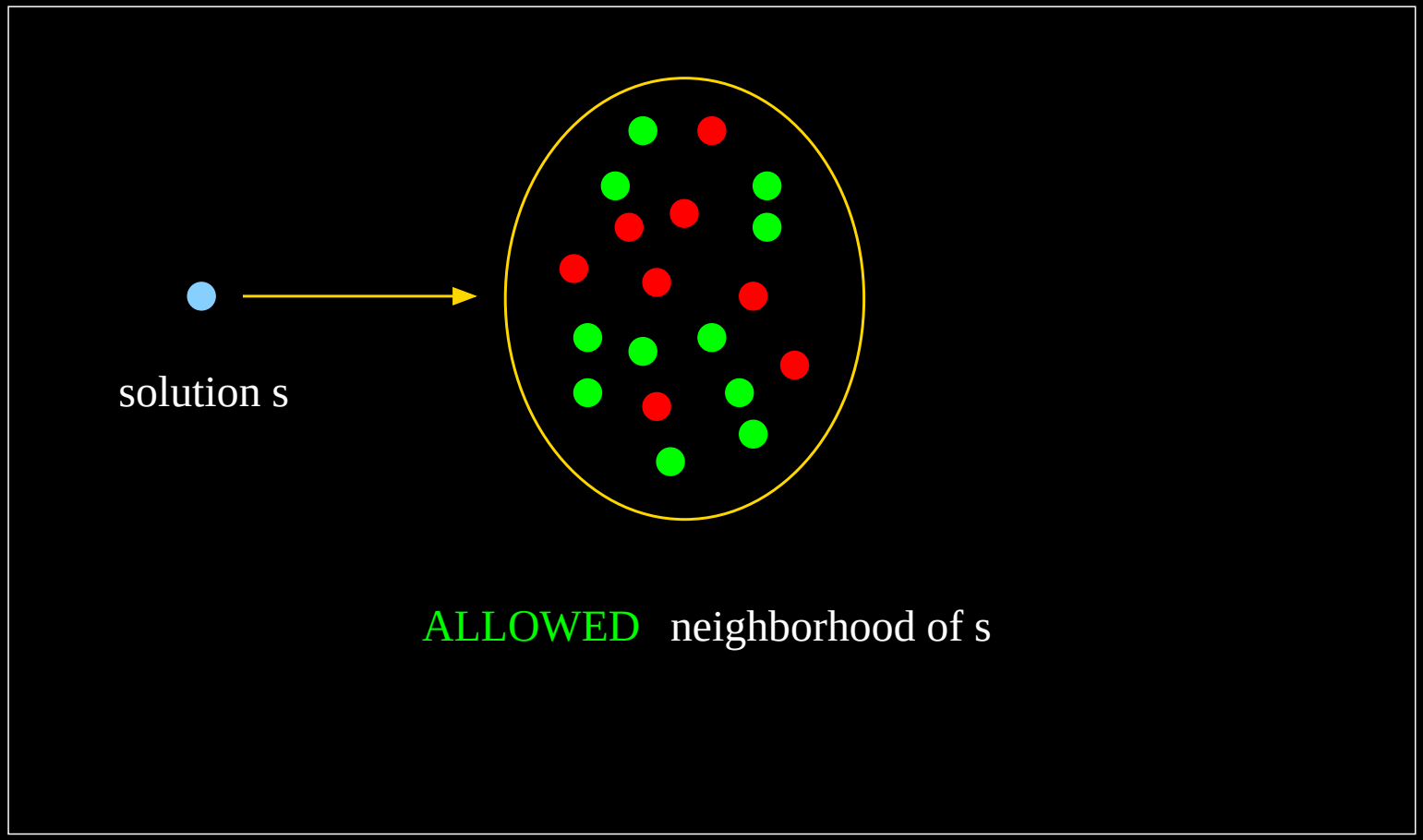
Tabu Search exploits the second idea: *change neighborhood structure*.

- 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)

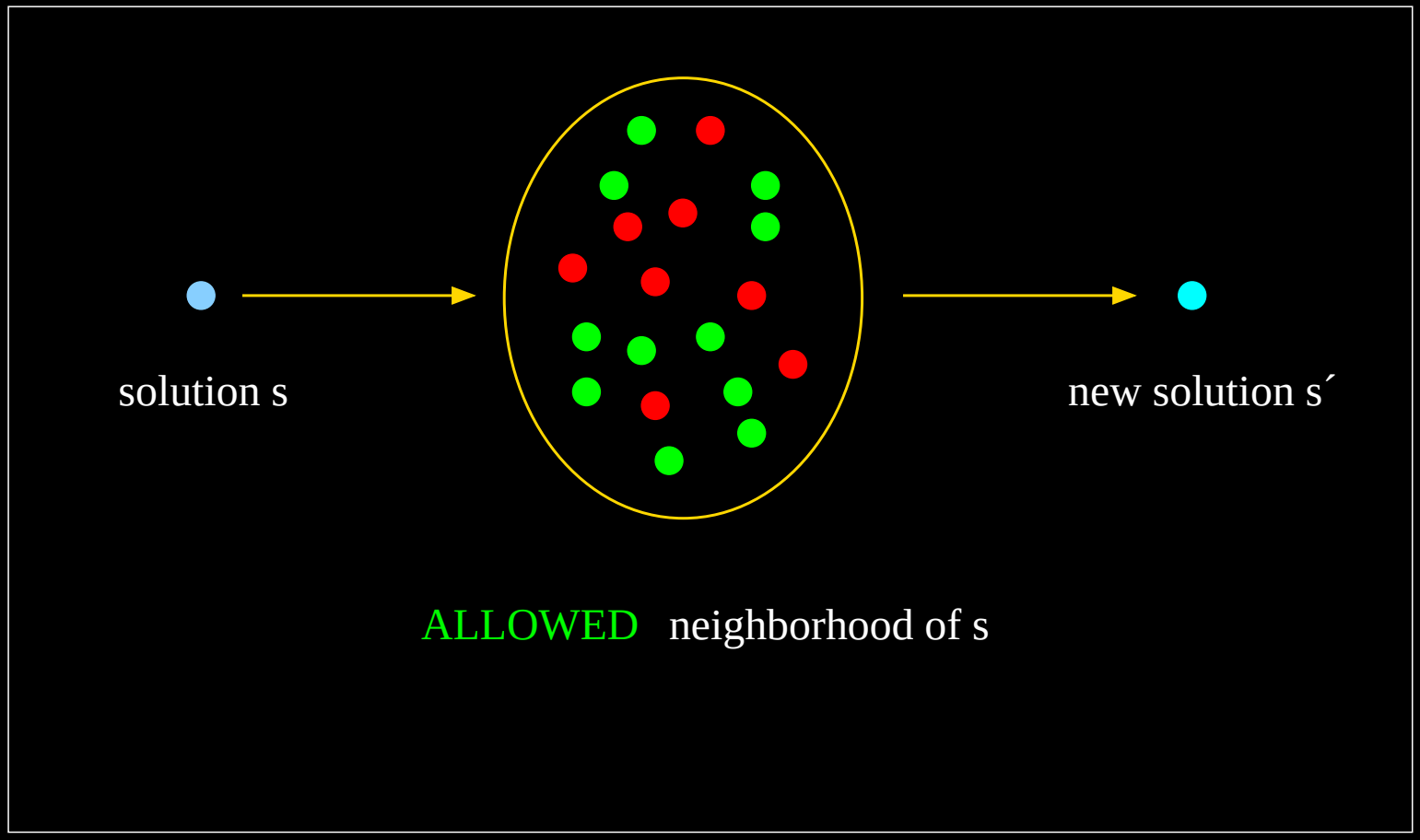
A pictorial view



A pictorial view



A pictorial view



Basic TS: the 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

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

TS: the 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

Applications

- Among the best performing metaheuristics (when applied with general strategies to balance intensification and diversification)
- Applied to many COPs
- TS approaches dominate the Job Shop Scheduling area

References:

- F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.

Variable Neighborhood Search

VNS exploits the second idea: *change neighborhood structure*.

- VNS uses different neighborhood structures during search
- A neighborhood $\mathcal{N}_i$ is substituted by neighborhood $\mathcal{N}_j$ as soon as local search can not improve the current best solution.

VNS: the algorithm

Select a set of neighborhood structures $\mathcal{N}_k,\ k = 1, \ldots, k_{max}$

$s \leftarrow$ GenerateInitialSolution()

while termination conditions not met **do**

  $k \leftarrow 1$

  while $k < k_{max}$ **do** {Inner Loop}

    $s' \leftarrow$ PickAtRandom($\mathcal{N}_k(s)$){Shaking phase}

    $s'' \leftarrow$ LocalSearch(s')

    if $f(s'') < f(s)$ **then**

      $s \leftarrow s''; k \leftarrow 1$

    else

      $k \leftarrow k + 1$

    end if

  end while

Applications

- Graph based COPs (e.g., p-Median problem, the Steiner tree problem, k-Cardinality Tree problem)
- Some variants also very effective

References:

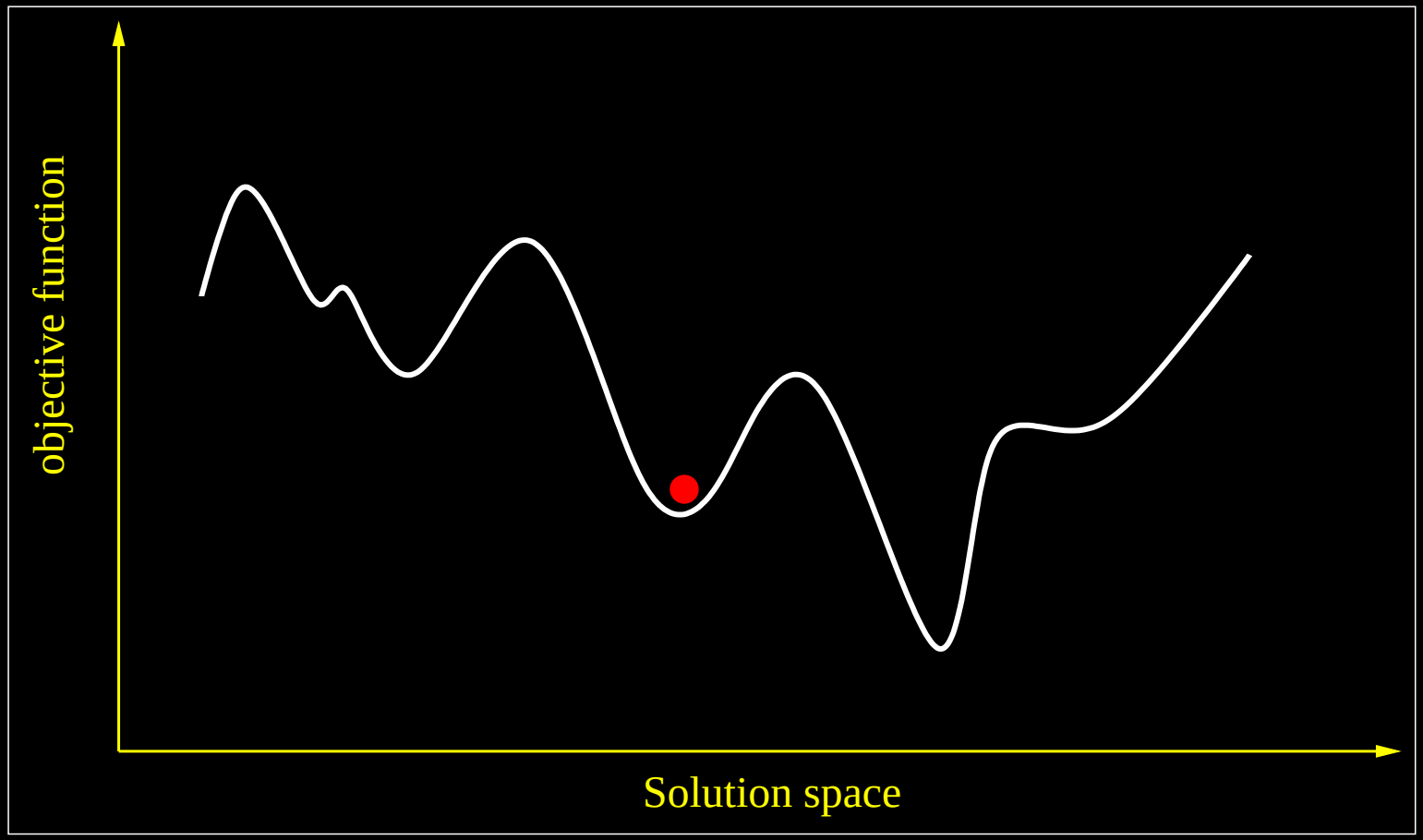
- P. Hansen and N. Mladenović, Variable neighborhood search: Principles and applications, European Journal of Operational Research, 2001.

Guided Local Search

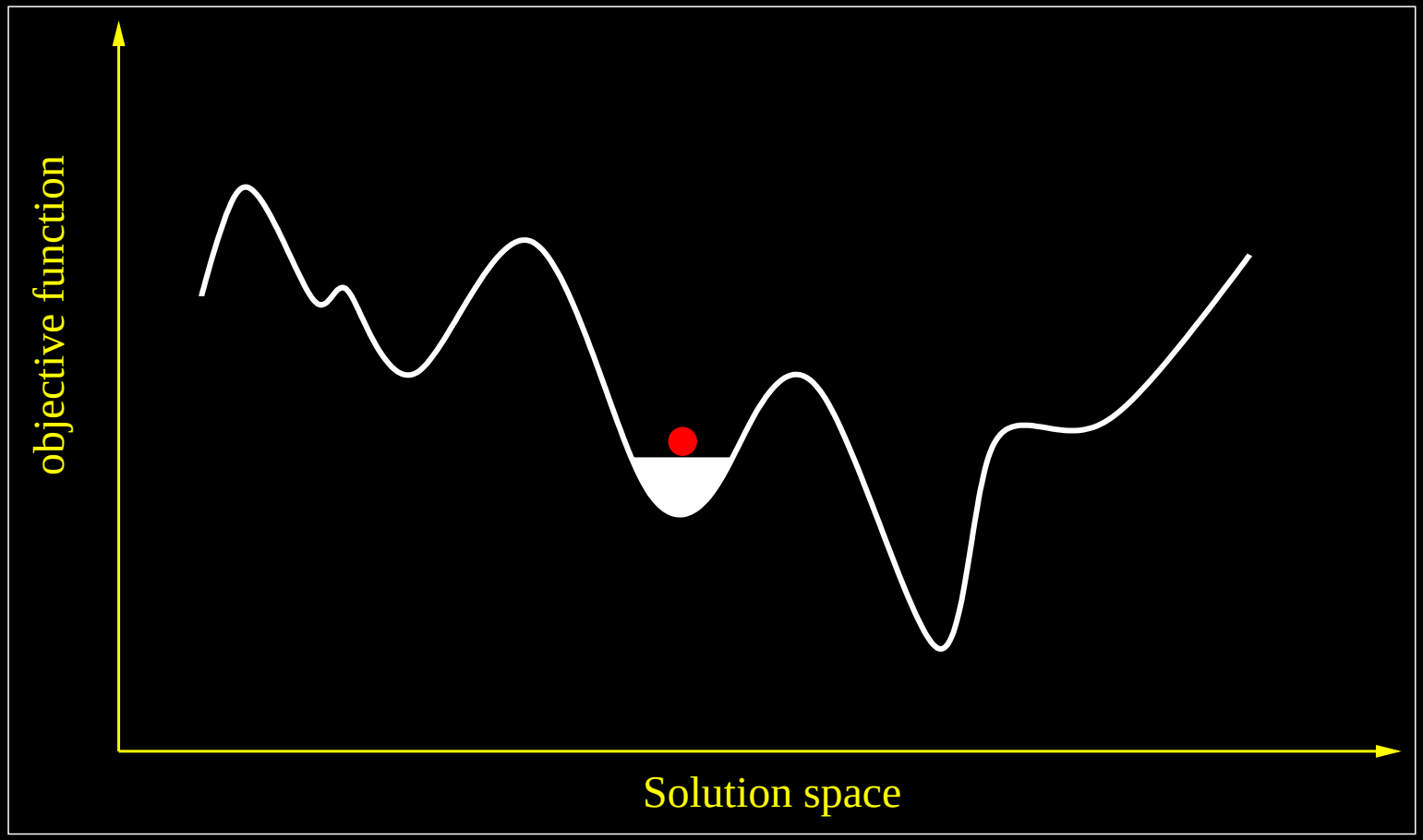
GLS exploits the third idea: *dynamically change the objective function*.

- 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”

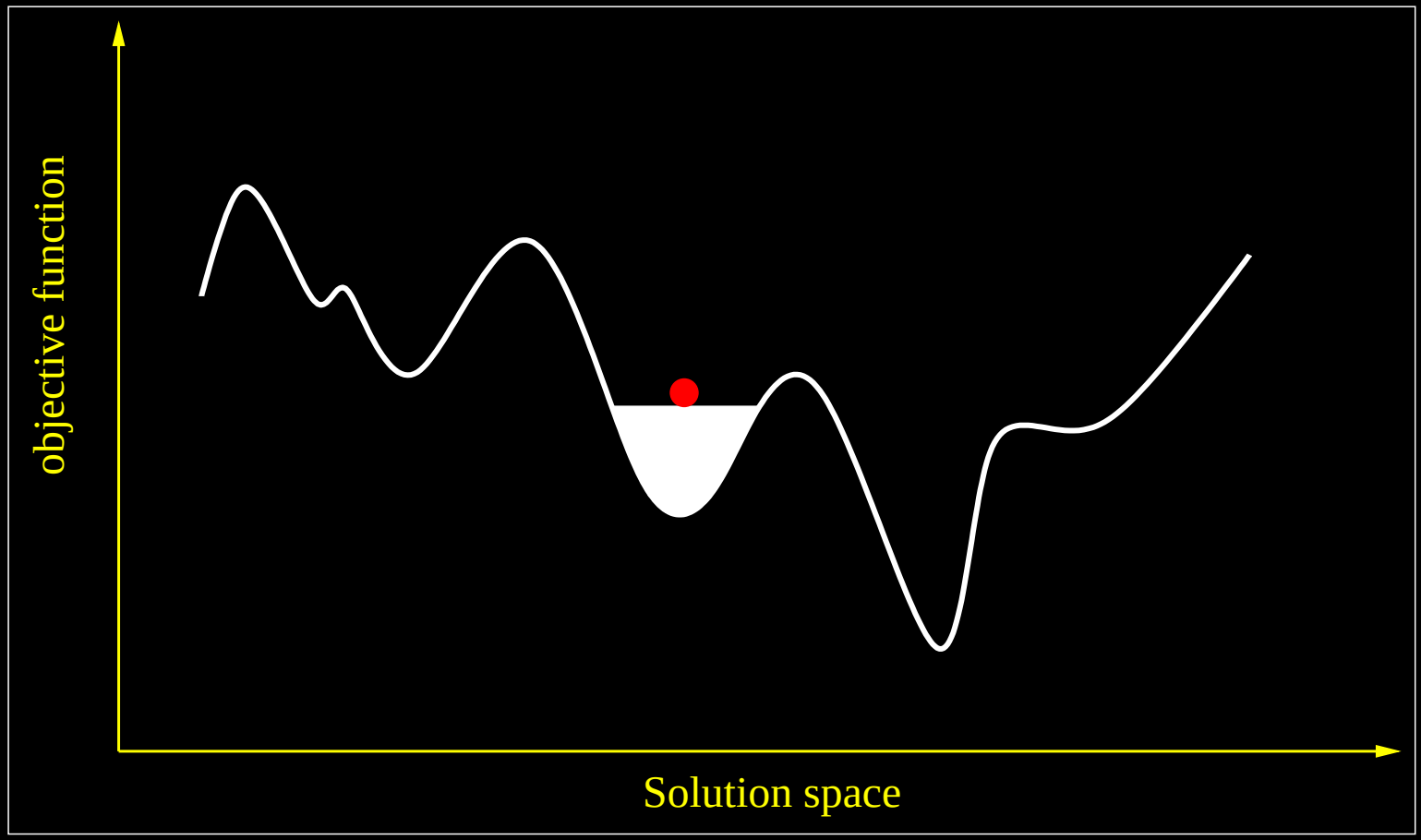
A pictorial view



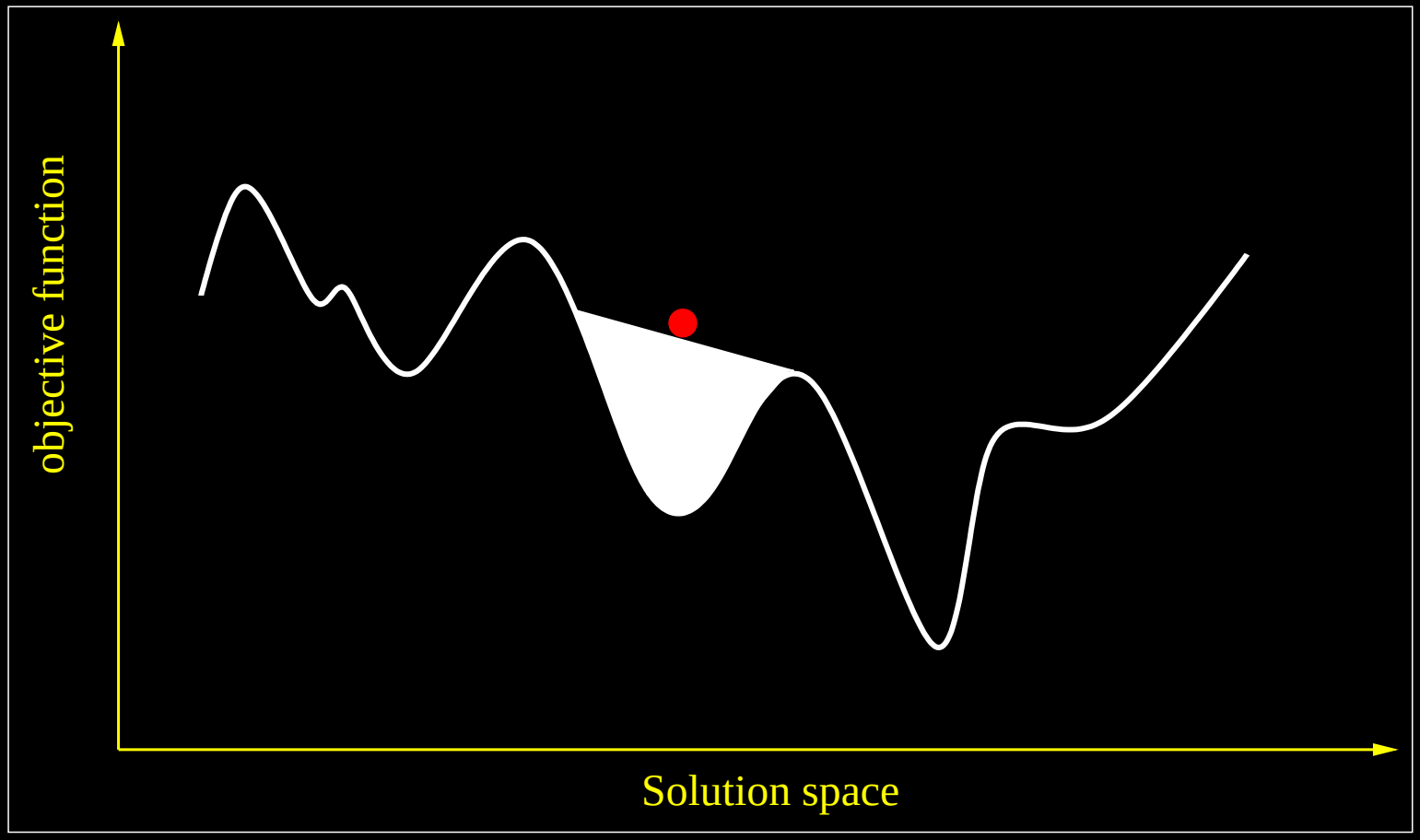
A pictorial view



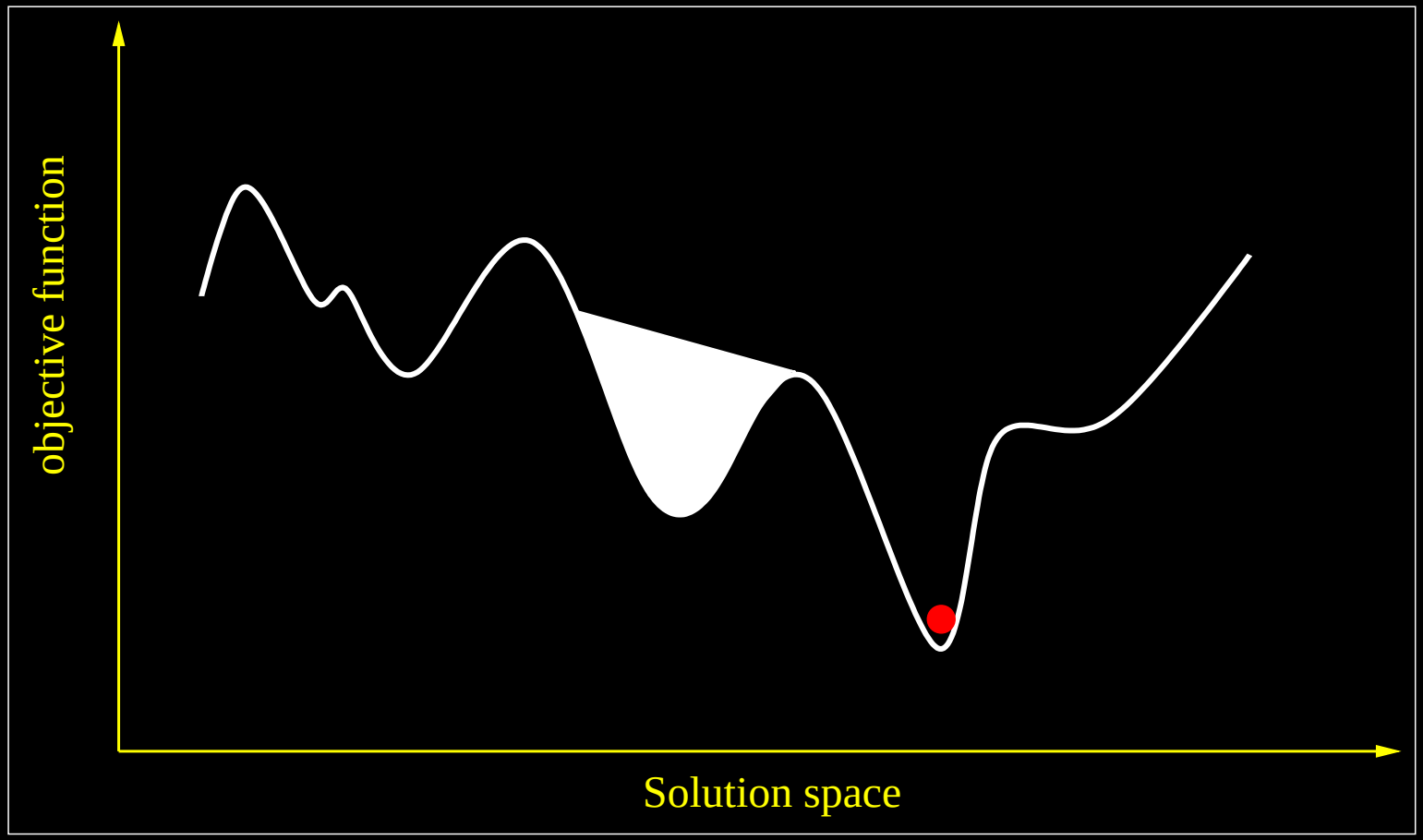
A pictorial view



A pictorial view



A pictorial view



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

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.

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

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

GLS: The 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

GLS: Applications

Crucial: optimally tune parameters and penalty updating procedure

GLS: Applications

Crucial: optimally tune parameters and penalty updating procedure

Successfully applied to the weighted MAXSAT, the VR problem, the TSP and the QAP

References:

- C. Voudouris and E. Tsang, Guided Local Search, European Journal of Operational Research, 1999.

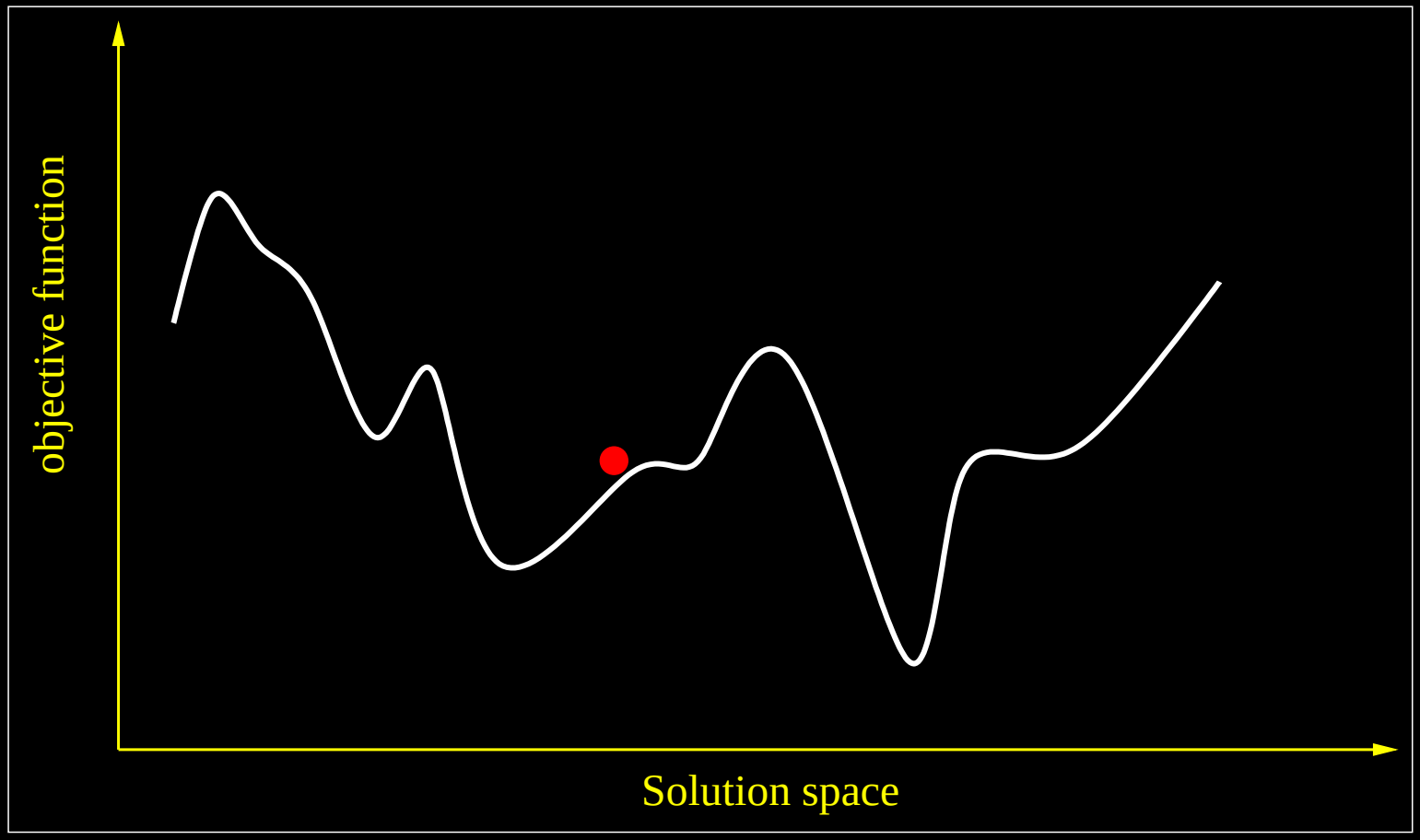
Iterated Local Search

- ILS can be seen as a general trajectory method framework
- Three basic blocks:
  - Local Search
  - Perturbation
  - Acceptance criteria

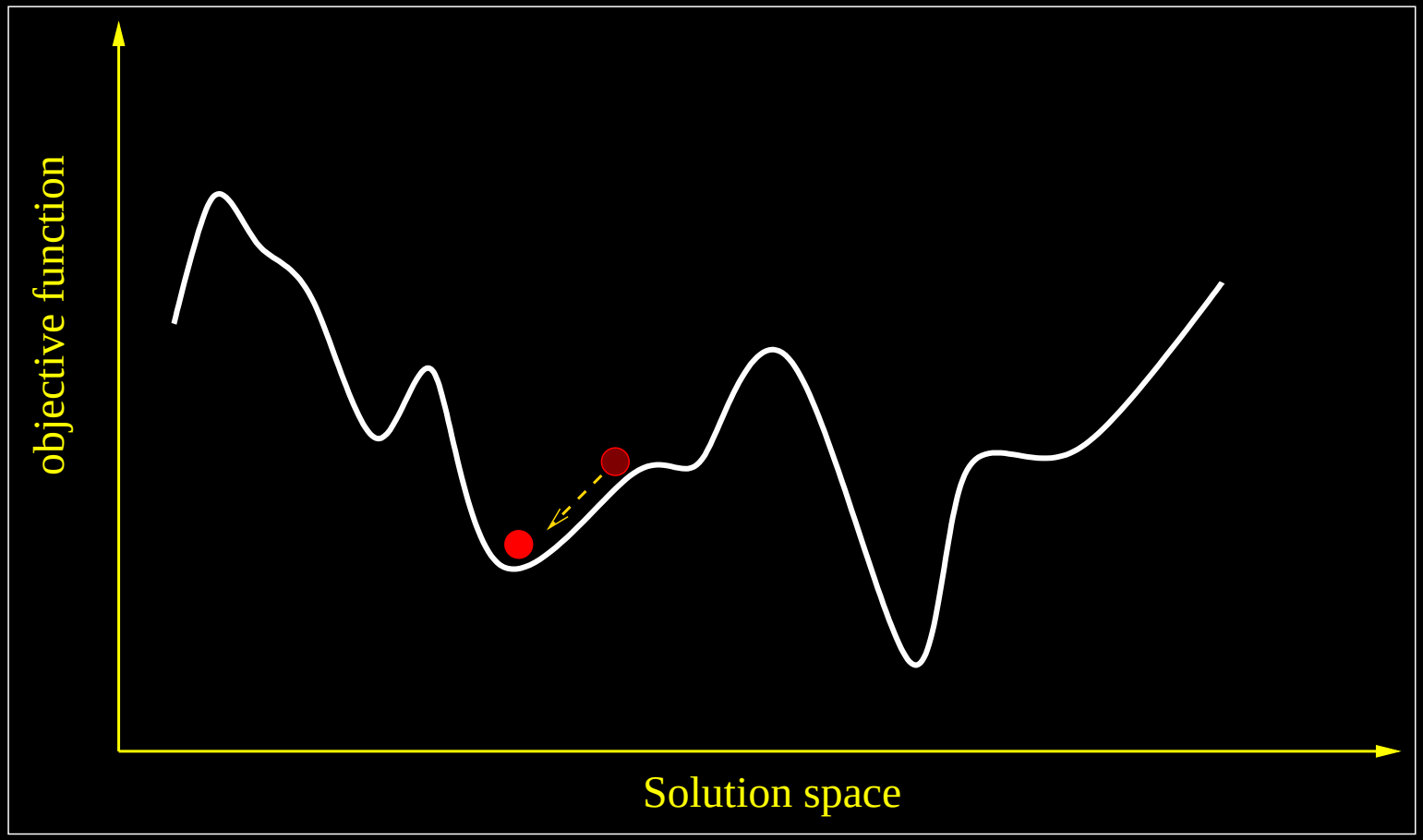
ILS: basic scheme

1. Generate an initial solution
2. Apply local search (e.g., SA, TS, etc.)
3. Perturb (i.e., *slightly* change) the obtained solution
4. Apply again local search with the perturbed solution as starting solution
5. Decide whether to accept the new solution or not
6. Go to step 3

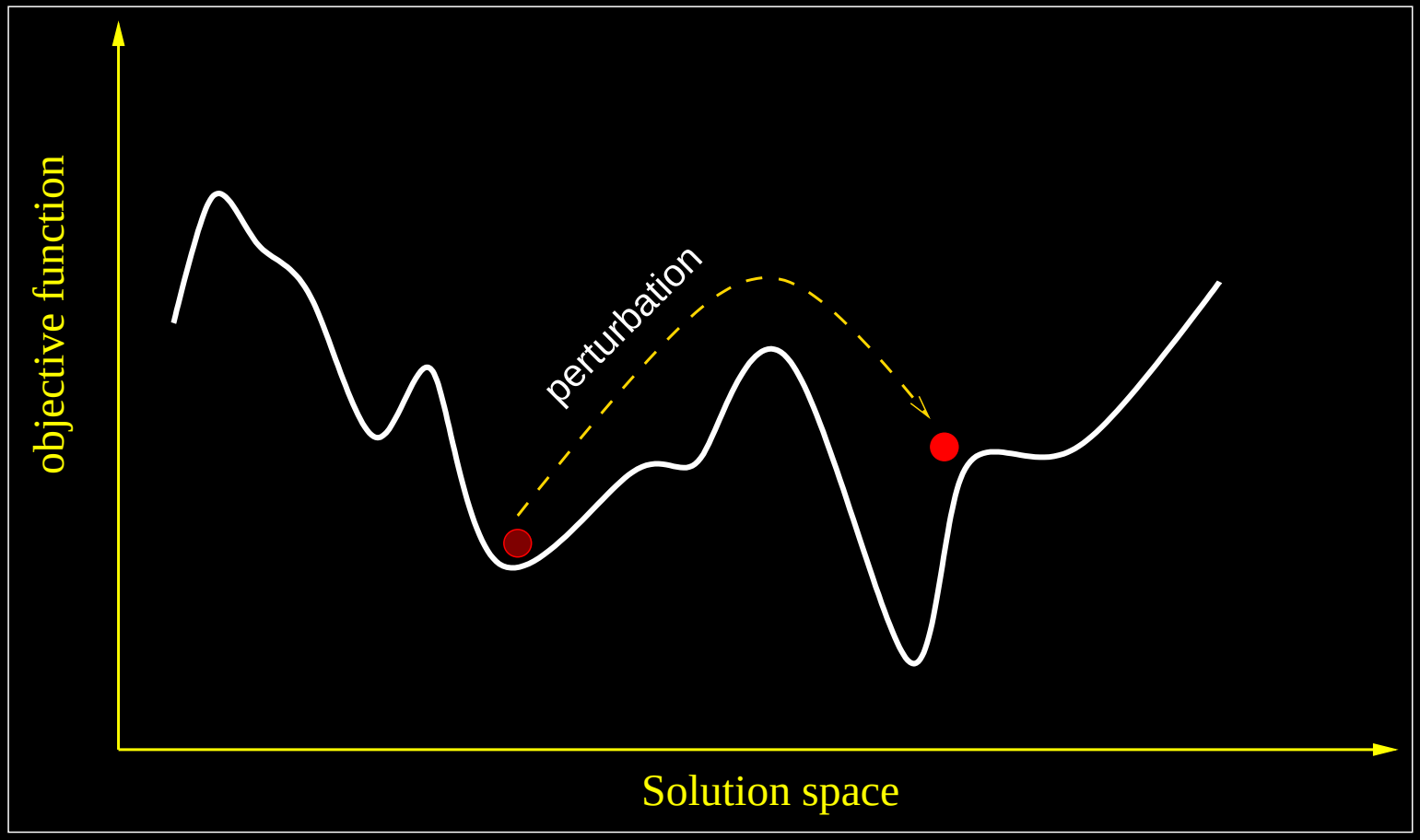
A pictorial view



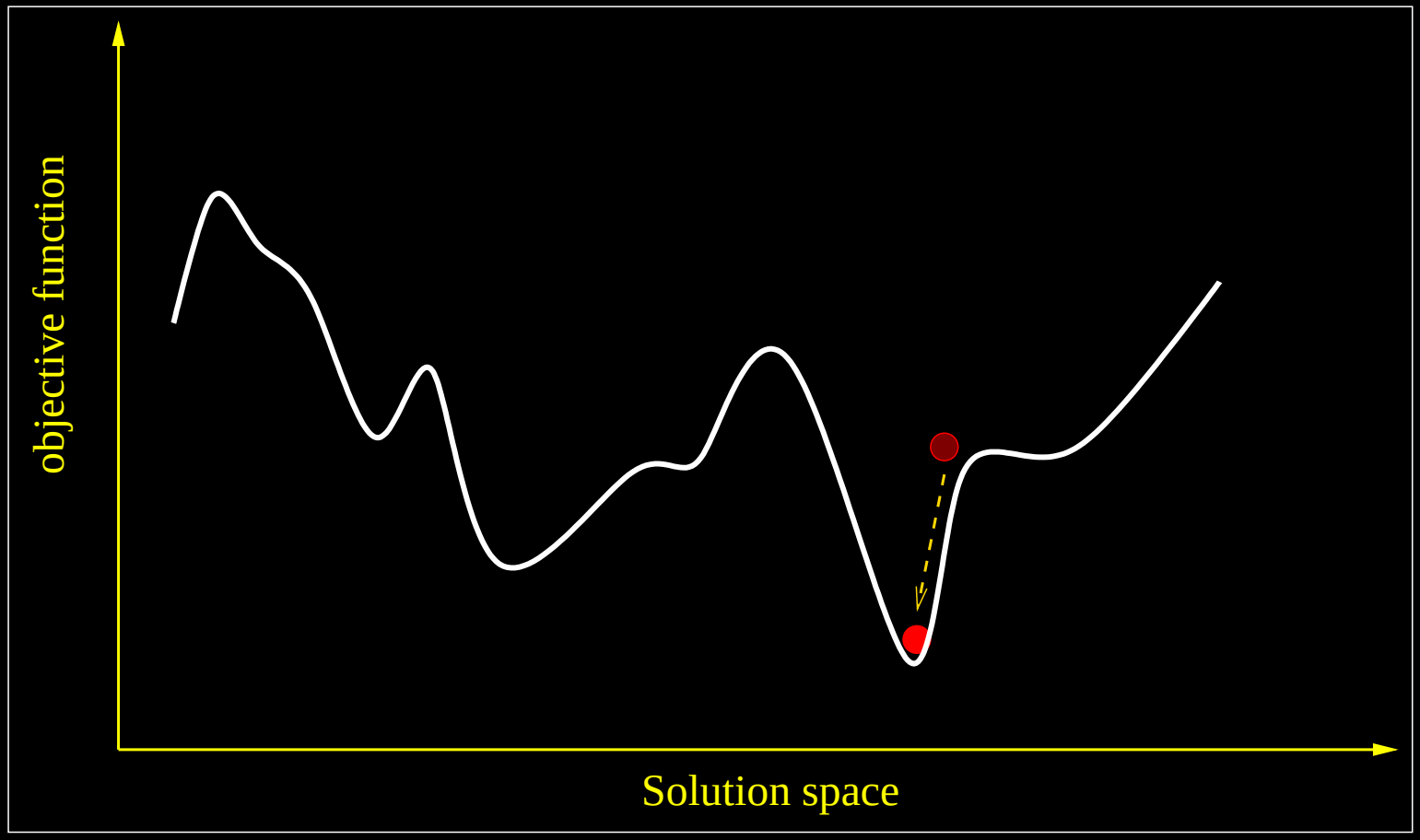
A pictorial view



A pictorial view



A pictorial view



ILS: the algorithm

$s_0 \leftarrow$ GenerateInitialSolution()
$s^* \leftarrow$ LocalSearch(s_0)
while termination conditions not met **do**
  $s' \leftarrow$ Perturbation($s^*, history$)
  $s^{*'} \leftarrow$ LocalSearch(s')
  $s^* \leftarrow$ ApplyAcceptanceCriterion($s^*, s^{*'}, history$)
end while

Design principles

- ’Local search‘ can be any kind of trajectory method.
- The perturbation should be *strong* enough to move the staring point to another local minimum basin of attraction, but it should keep some parts of the current solution.
- The acceptance criteria can range from very simple (e.g., accept the new solution if better than the current one) to more complex (e.g., with probabilistic acceptance)

Applications

- TSP, QAP, Single Machine Total Weighted Tardiness (SMTWT) problem, Graph Coloring Problem

References:

- H. R. Lourenço and O. Martin and T. Stützle, Iterated Local Search, in *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2002.

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.

Population-based methods

- Evolutionary Algorithms
  - Evolutionary Programming
  - Evolution Strategies
  - Genetic Algorithms
- Ant Colony Optimization

But also (not coverd by this introduction): Scatter Search, Population-Based Incremental Learning and Estimation of Distribution Algorithms.

Evolutionary Algorithms

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

Evolutionary Algorithms

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

They include:

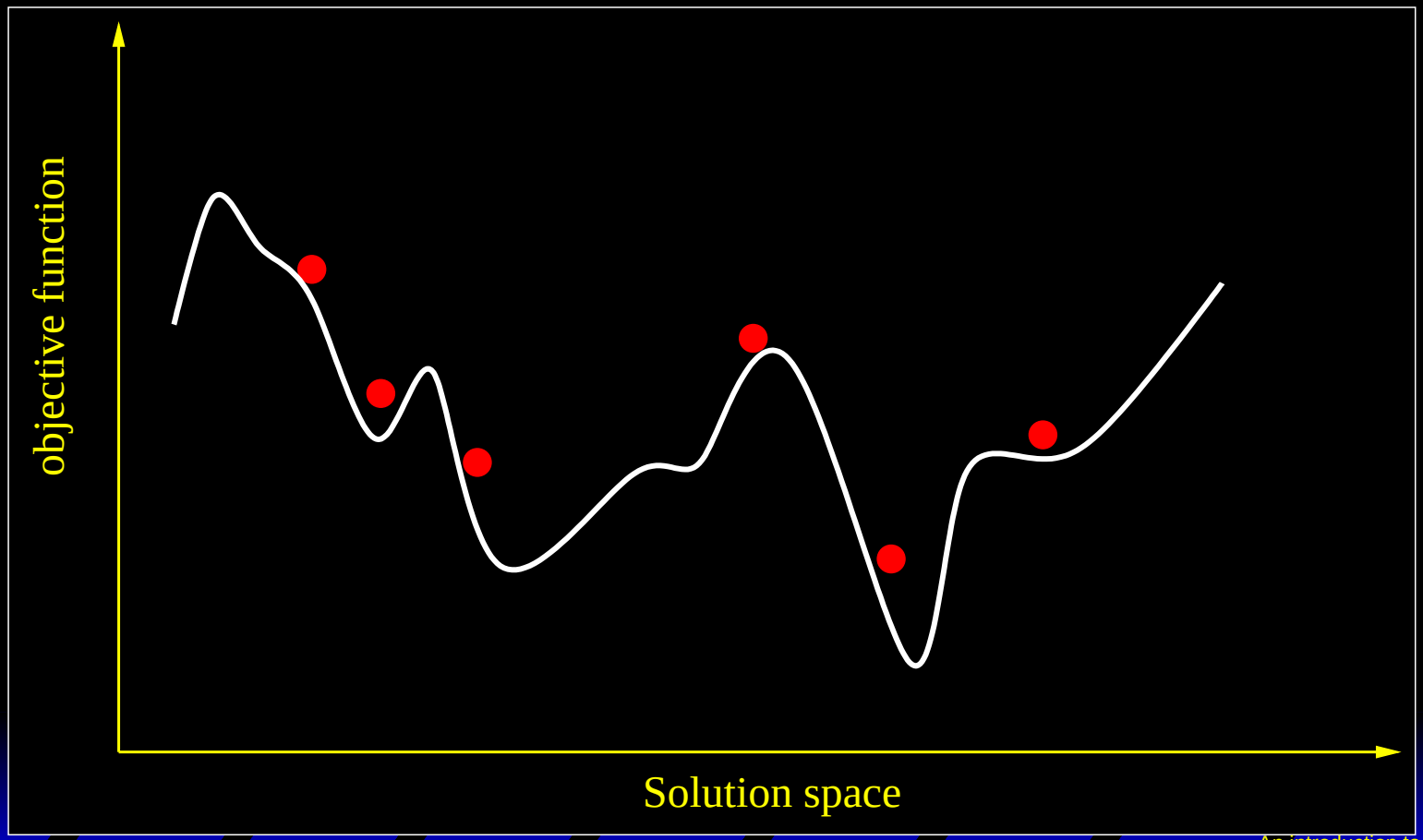
- Evolutionary Programming
- Evolution Strategies
- Genetic Algorithms

Evolutionary Algorithms

Basic principle: moving a population of solutions toward good regions of the search space.

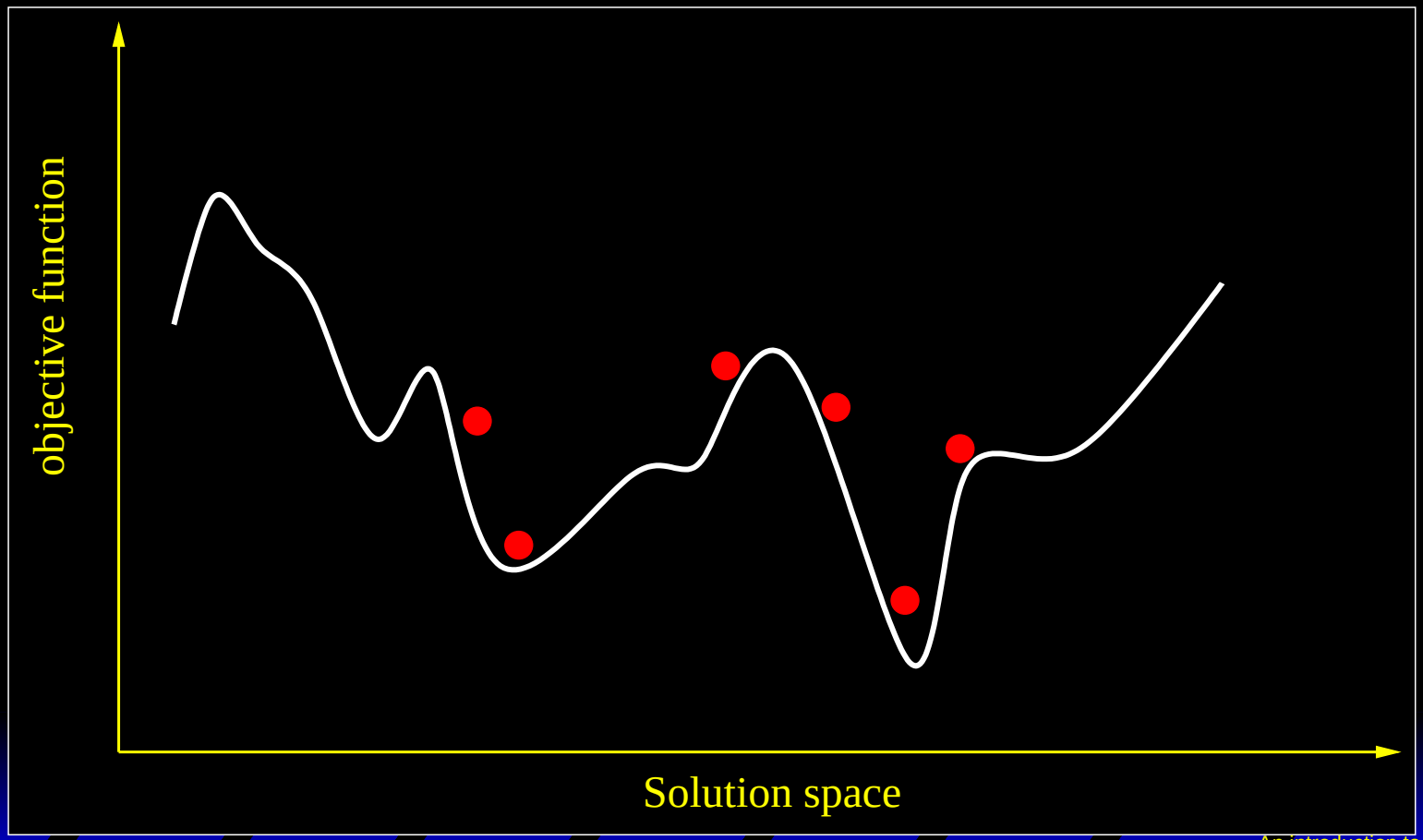
Evolutionary Algorithms

Basic principle: moving a population of solutions toward good regions of the search space.



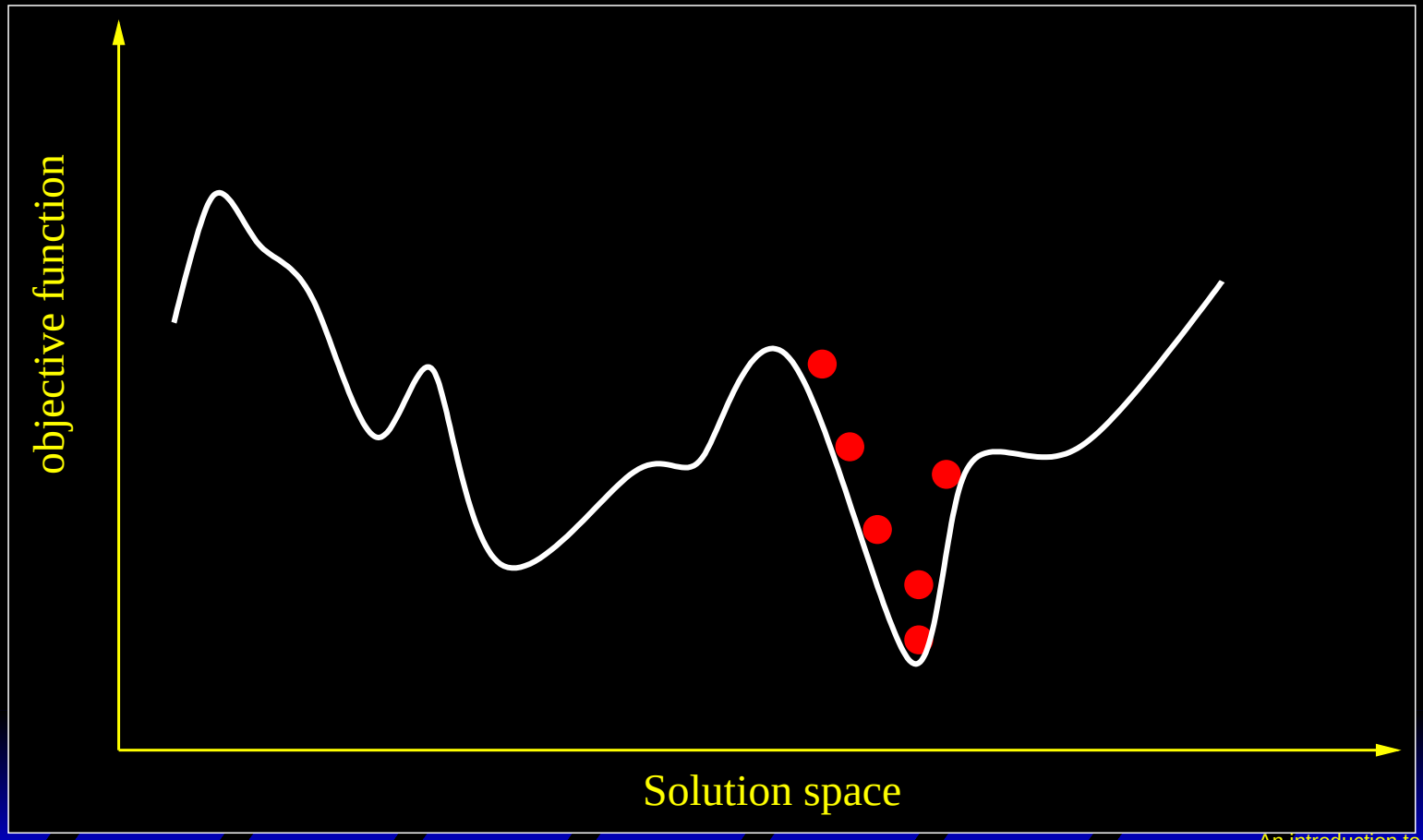
Evolutionary Algorithms

Basic principle: moving a population of solutions toward good regions of the search space.



Evolutionary Algorithms

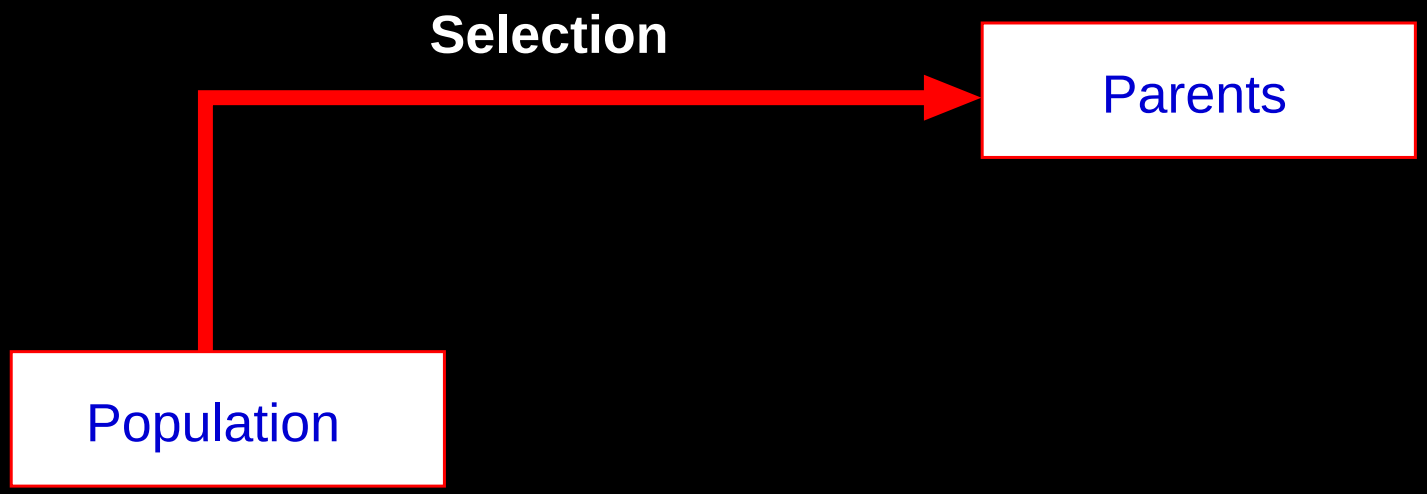
Basic principle: moving a population of solutions toward good regions of the search space.



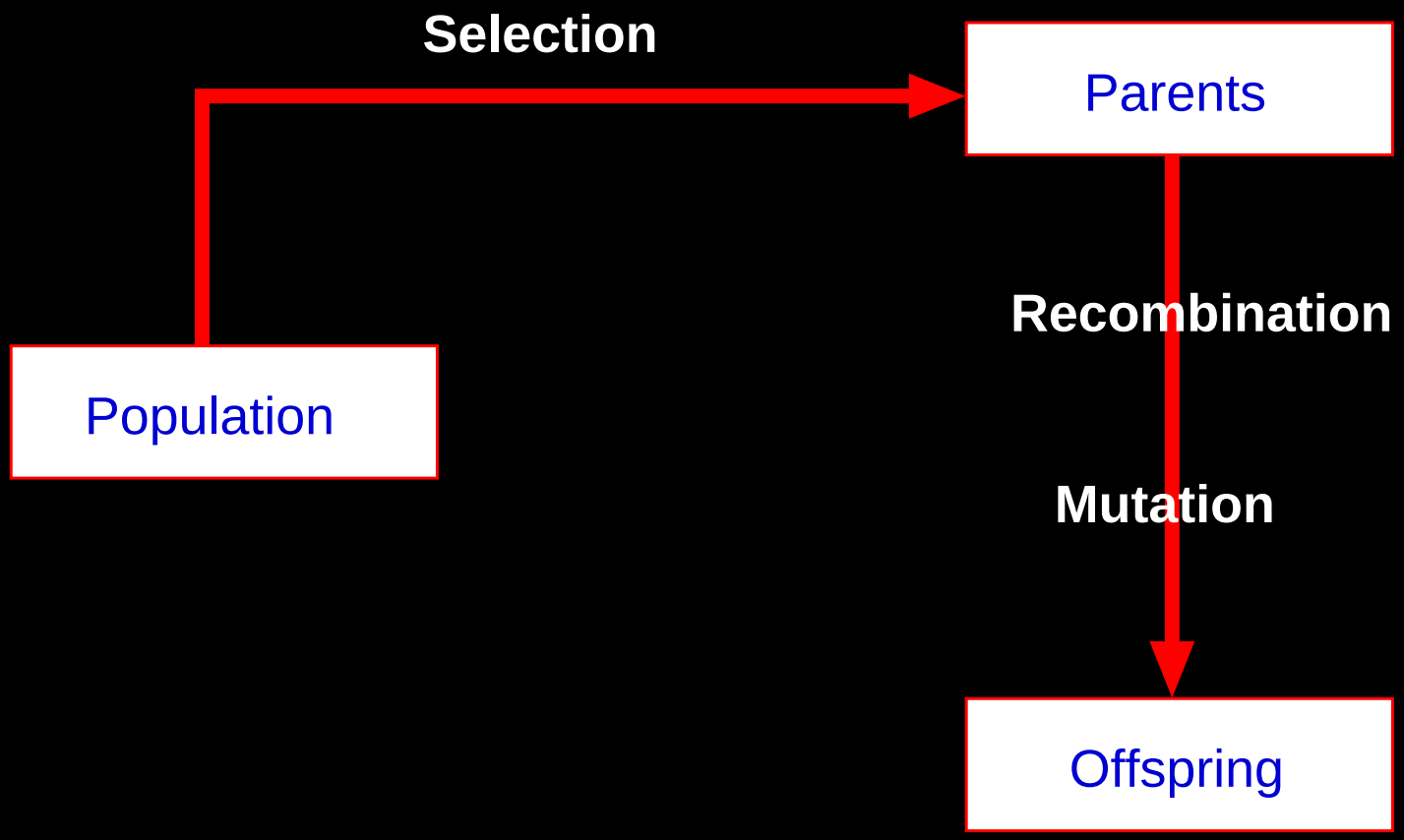
The Evolutionary Cycle

The Evolutionary Cycle

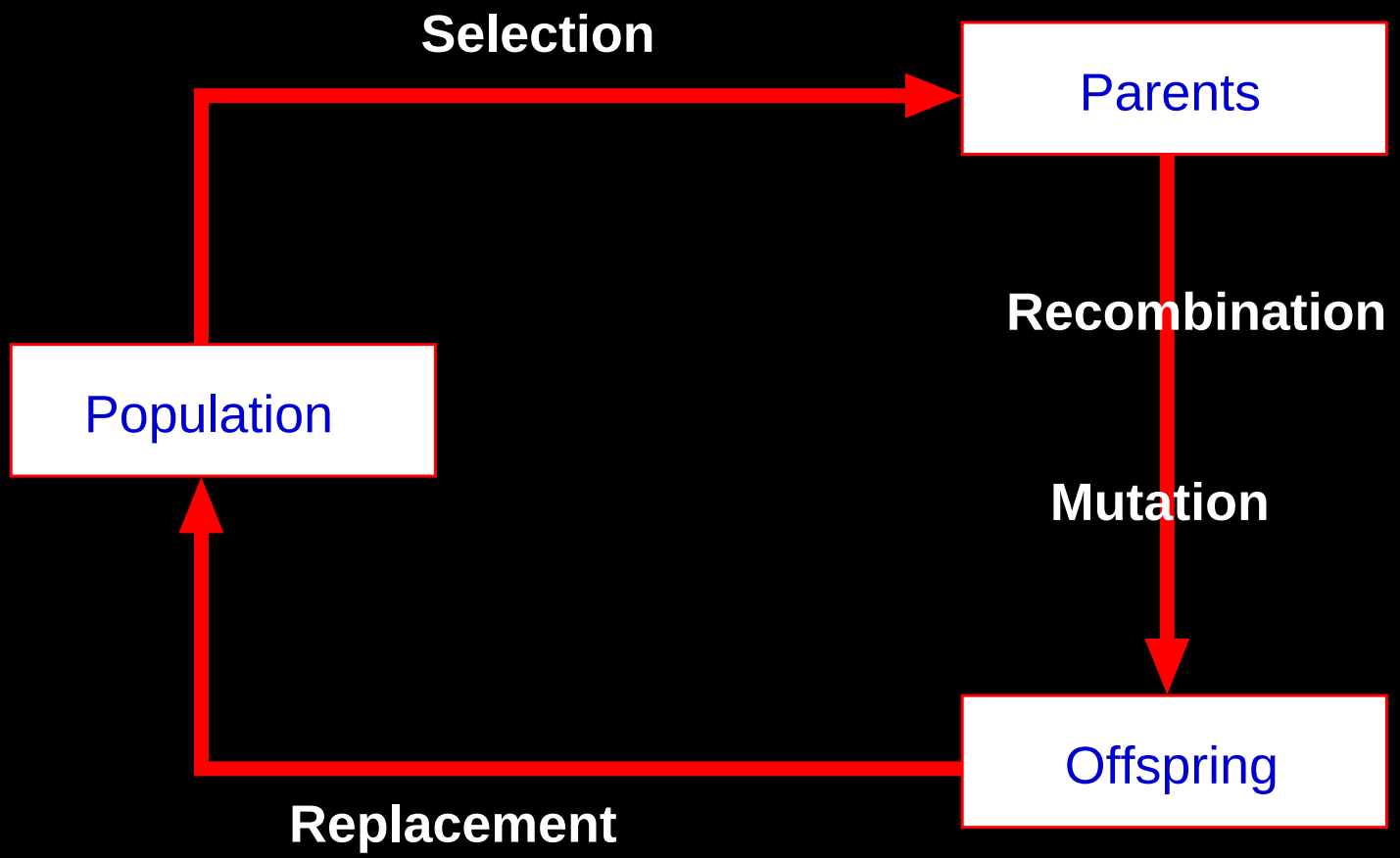
The Evolutionary Cycle



The Evolutionary Cycle



The Evolutionary Cycle



EA: the 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

The *Seven Features*

1. Description of the individuals
2. Evolution process
3. Neighborhood structure
4. Information sources
5. Infeasibility
6. Intensification strategy
7. Diversification strategy

Description of the individuals

Solutions can be represented in many ways:

- bit-strings
- integer/real arrays
- tree structures or more complex data structures

► the representation is crucial for the success of the algorithm

Evolution process

The evolution process determines which individuals will enter the population at each iteration.

- *Generational replacement*: the offspring entirely replaces the old population
- *Steady state*: some new indivuduals are inserted in the old population
- The population size can be constant or varying

Neighborhood structure

The neighborhood function $\mathcal{N}_{\mathcal{EC}} : \mathcal{I} \rightarrow 2^{\mathcal{I}}$ defines, for every individual $i \in \mathcal{I}$, the set of individuals $\mathcal{N}_{\mathcal{EC}}(i) \subseteq \mathcal{I}$ which can be recombined with it.

- *Unstructured population*: every individual can be recombined with any other one (e.g., Simple Genetic Algorithm)
- *Structured population*: otherwise (e.g., Parallel Genetic Algorithm with “islands”)

Information sources

Several kinds of recombination are possible:

- two-parent crossover
- multi-parent crossover
- population statistics-based recombination operators

Infeasibility

The result of a recombination could be an individual violating some constraints.

Three possible ways of dealing with infeasibility:

- *Reject*: discard infeasible solutions
- *Penalize*: decrease the fitness of individuals violating constraints
- *Repair*: apply some operators to change the solution trying to obtain a feasible one

Intensification strategy

It is possible to apply operators or algorithms to improve the fitness of single individuals.

For example:

- Before the replacement, apply local search to every individual of the population ($\rightarrow$ *memetic algorithms*.
- Apply mutation operators based on local improvements (e.g., some steps of local search).

Diversification strategy

To avoid *premature convergence* of the search, diversification techniques are introduced.

For example:

- Random mutation (most often adopted)
- Introduce into the population new individuals ’coming‘ from not yet explored areas of the search space

Applications

- Applied to nearly any COPs and optimization problems
- Particularily effective in robotics applications

References:

- T. Bäck and D. Fogel and M. Michalewicz eds., *Handbook of Evolutionary Computation*, Institute of Physics Publishing Ltd., 1997.

Ant Colony Optimization

Population-based metaheuristic inspired by the foraging behavior of ants, which enables them to 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 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

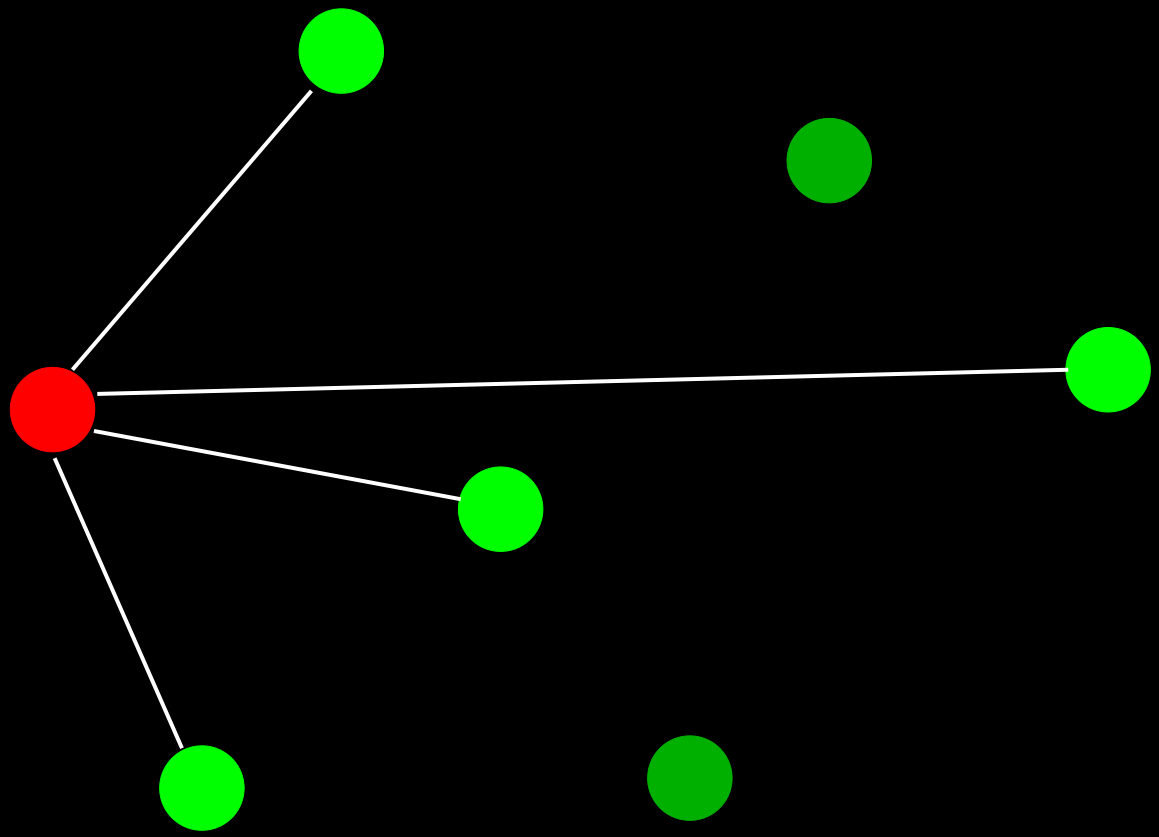
One possible TSP model for ACO:

- 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).

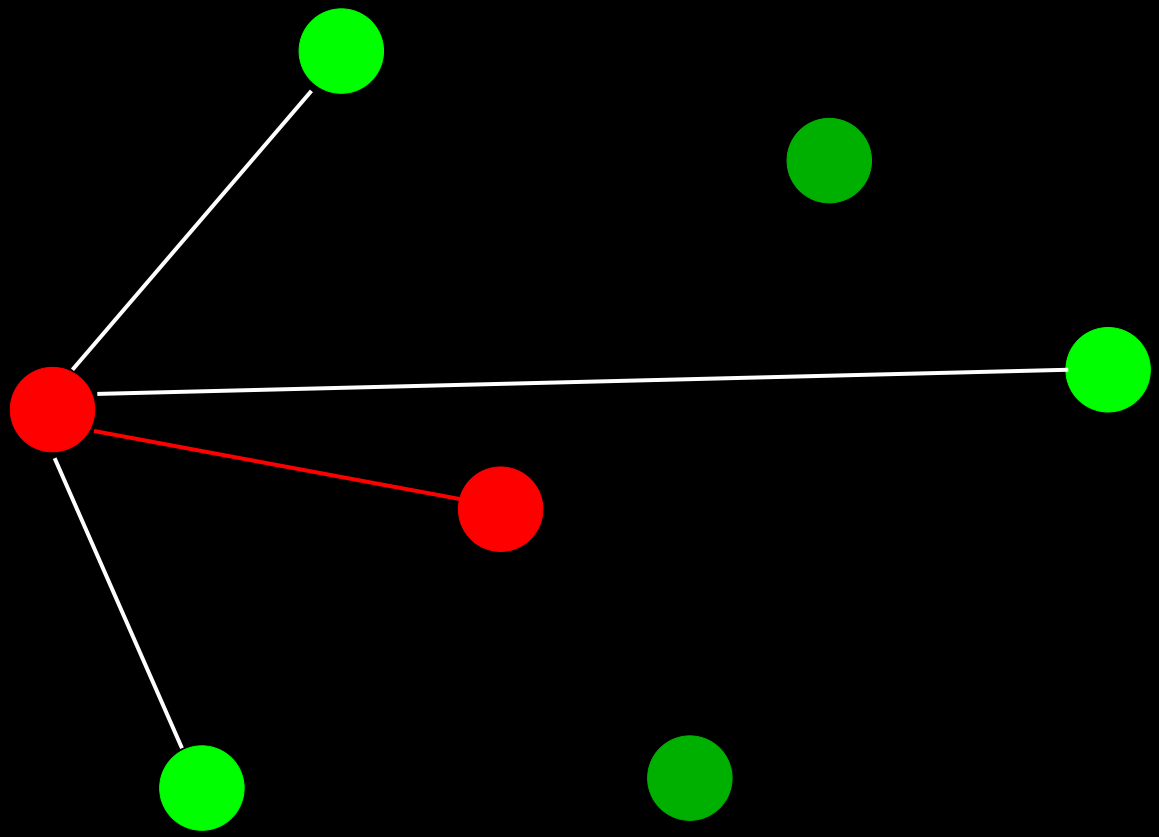
Sources of information

- 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).

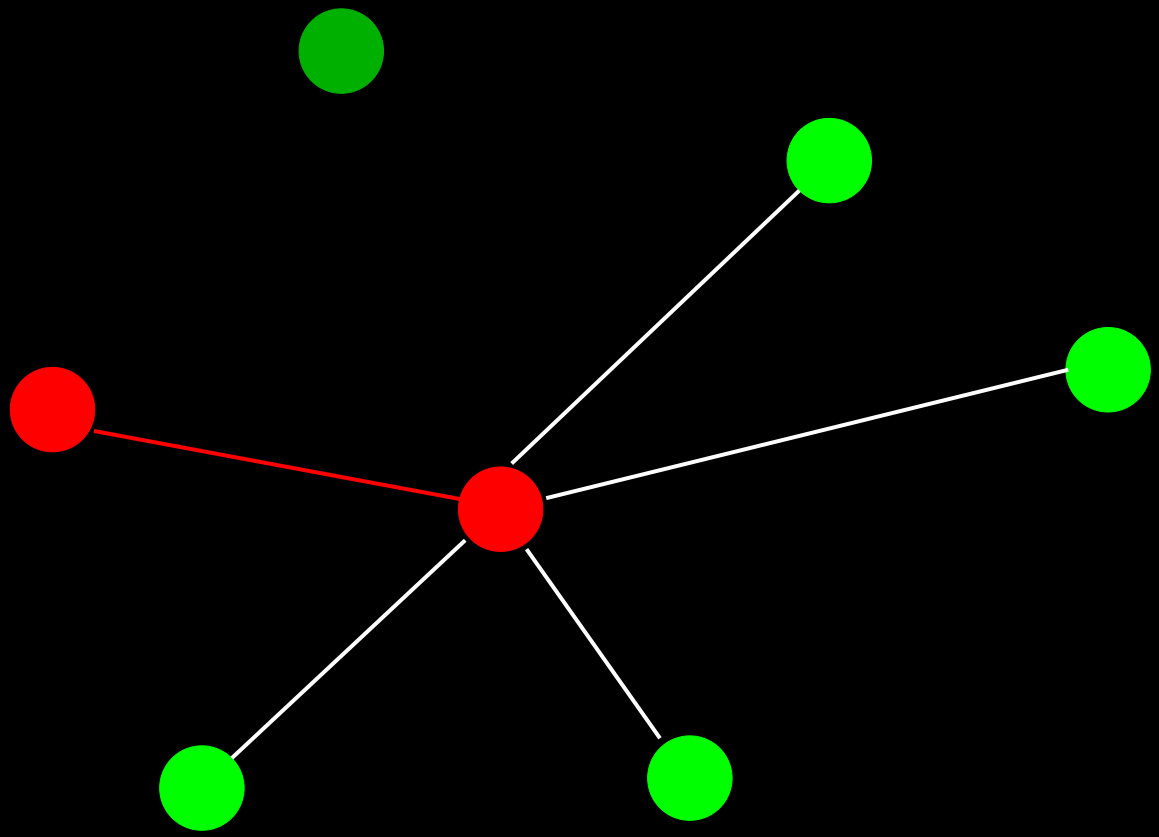
A pictorial view



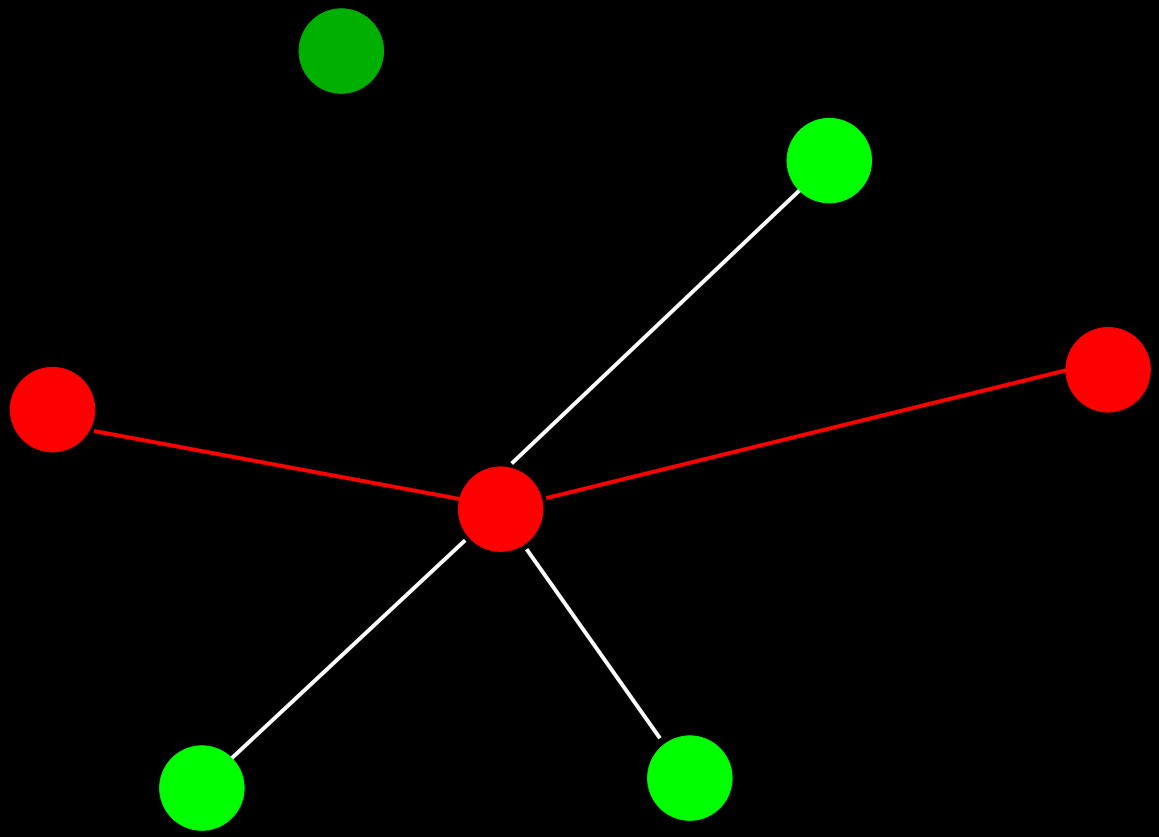
A pictorial view



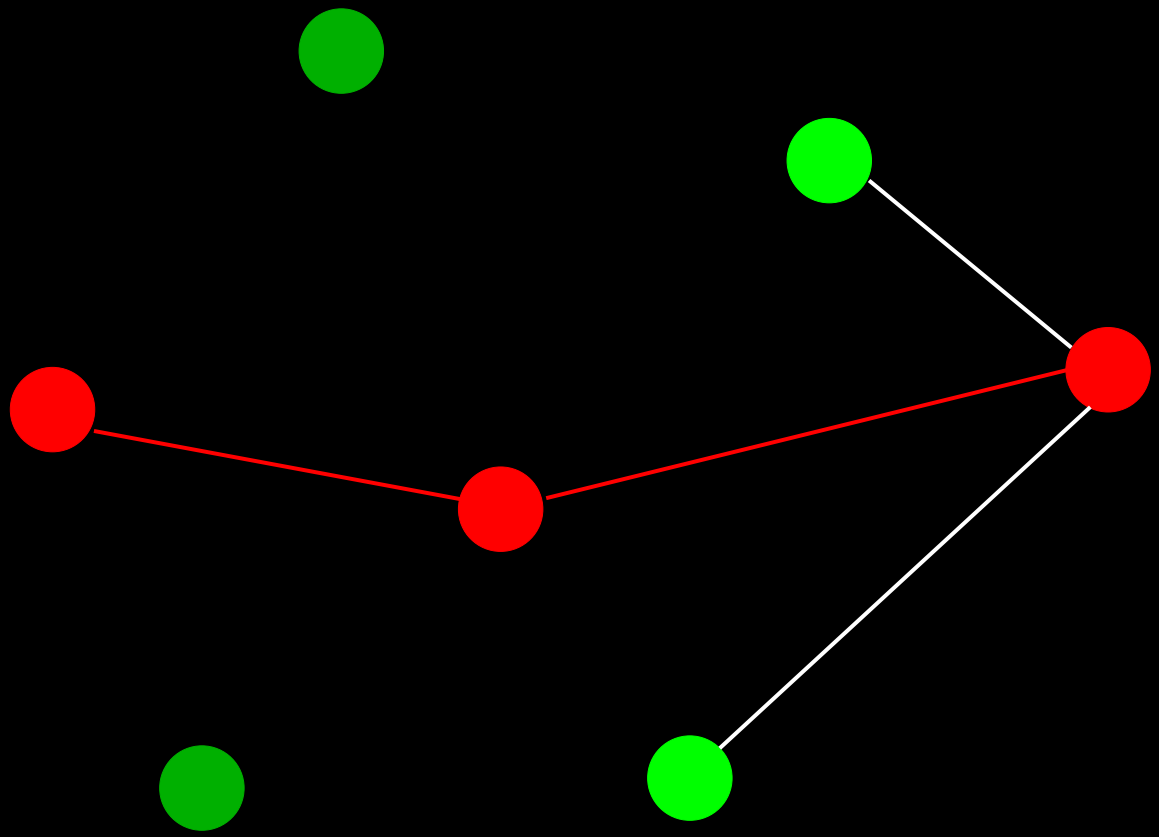
A pictorial view



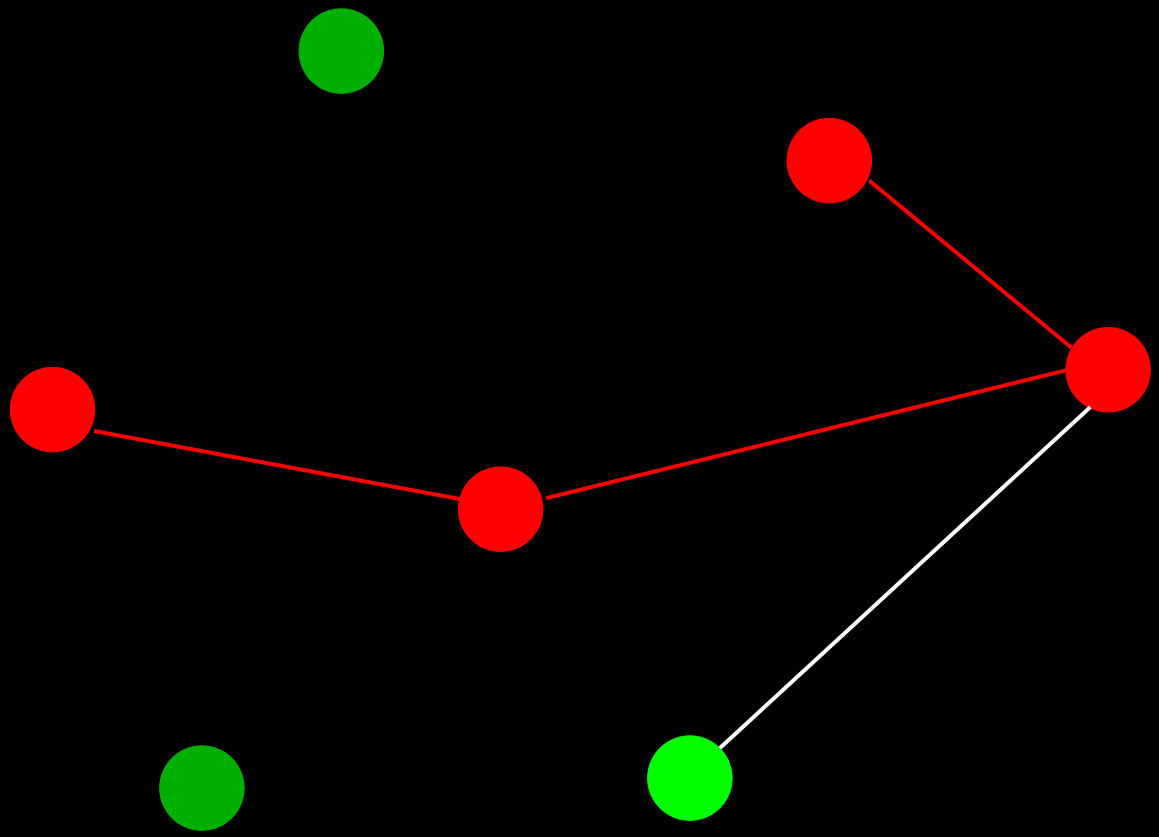
A pictorial view



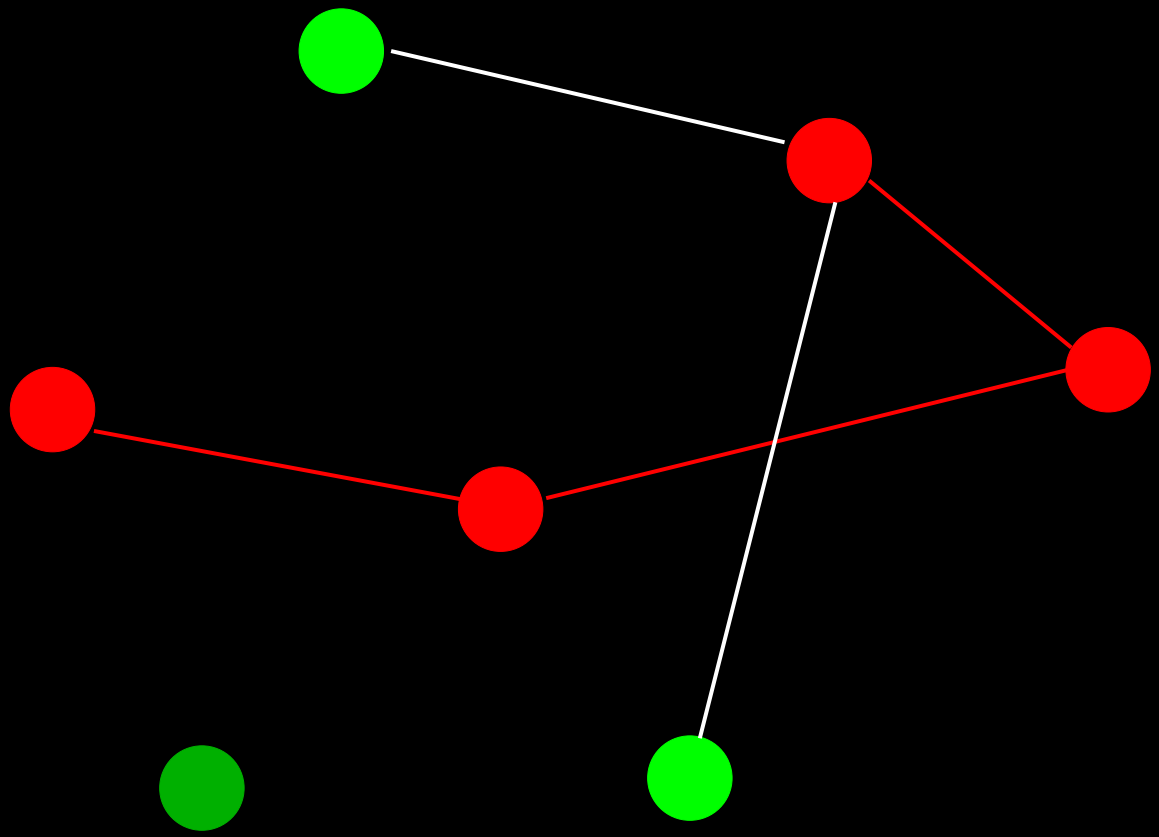
A pictorial view



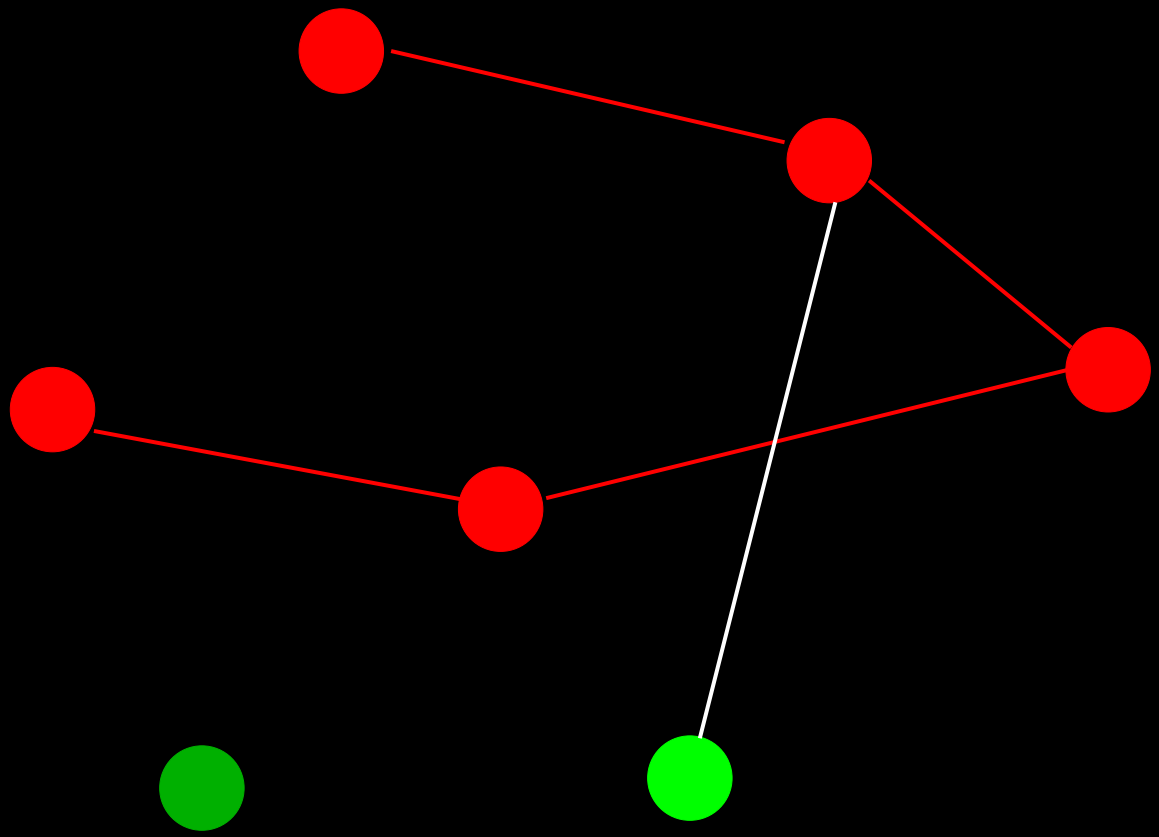
A pictorial view



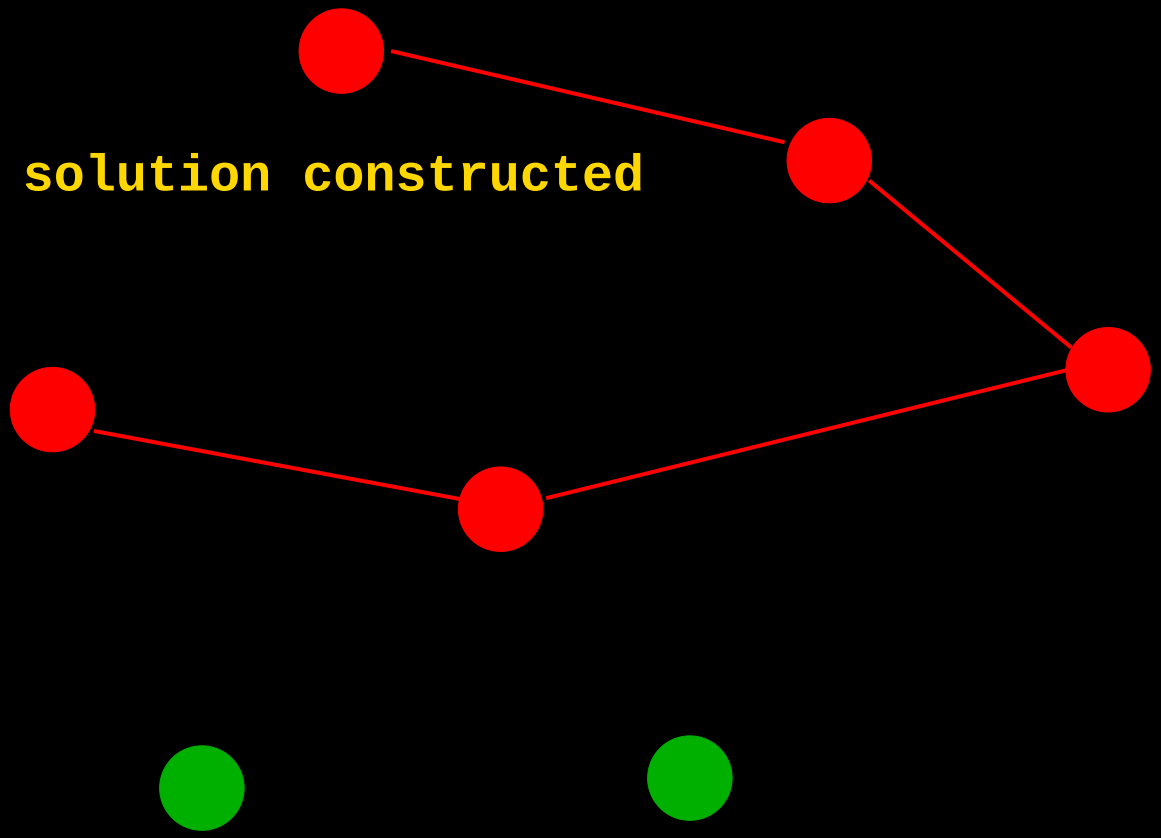
A pictorial view



A pictorial view



A pictorial view



ACO: the algorithm

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

Solution construction

- Ants move by applying a *stochastic local decision policy* that makes use of the pheromone values and the heuristic values on components of the construction graph.
- While moving, the ant keeps in memory the partial solution it has built in terms of the path it was walking on the construction graph.

Pheromone Update

- When adding a component c_j to the current partial solution, an ant can update the pheromone trail (*online step-by-step pheromone update*).
- Once an ant has built a solution, it can retrace the same path backward and update the pheromone trails of the used components according to the quality of the solution it has built (*online delayed pheromone update*).
- *Pheromone evaporation* is always applied $\rightarrow$ the pheromone trail intensity on the components decreases over time.

Daemon Actions

- Can be used to implement centralized actions which cannot be performed by single ants. E.g.,
  - local search procedure applied to the solutions built by the ants
  - collection of global information used to decide whether to deposit additional pheromone to bias the search process from an non-local perspective

Variants

- Ant Colony System
- $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System – among the most effective ACO implementations

Variants

- Ant Colony System
- $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System – among the most effective ACO implementations

- *Hyper-Cube Framework*: generalizes pheromone-based construction mechanisms

Applications

- Routing in communication networks, Sequential Ordering Problem, Resource Constraint Project Scheduling

► ACO is effective when combined with local search and/or tree-search strategies.

References:

- M. Dorigo and G. Di Caro, The Ant Colony Optimization Meta-Heuristic, in *New Ideas in Optimization*, McGraw-Hill, 1999. MIT Press, 2003.

Hybrid metaheurictics

- Component exchange among metaheuristics
- Cooperative search
- Integrating metaheuristics and systematic methods

Component exchange

- Integrate trajectory methods into population-based methods (e.g., memetic algorithms)
- Apply general I&D strategies of a metaheuristic into another one (e.g., TS restart strategy applied into ILS)

► The combination of population-based methods with trajectory methods produces hybrid algorithms which are often more efficient than single methods.

Cooperative search

- *Loose* form of integration
- The search is performed by different algorithms (possibly running in parallel)
- The algorithms exchange information during the search process
- Crucial:
  - kind of information exchanged
  - implementation

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.

Metaheuristics and systematic methods

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.

Metaheuristics and systematic methods

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.

References: F. Focacci and F. Laburthe and A. Lodi, Local Search and Constraint Programming, in Handbook of Metaheuristics, Kluwer Academic Publishers, 2002.

Overview references

- C.Blum, A.Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. Technical report TR/IRIDIA/2001-13, IRIDIA, Université Libre de Bruxelles, Belgium.
- F. Glover and G. Kochenberger eds., Handbook of Metaheuristics, Kluwer Academic Publishers, 2002.
- A.Roli, M.Milano. MAGMA: A Multiagent Architecture for Metaheuristics. LIA Technical Report 02-007, no. 60. To be published in IEEE Tran. of Systems, Men and Cybernetics – part.B.

Internet resources

- www.metaheuristics.net
- tew.ruca.ua.ac.be/eume/