PART V: Heuristic Search

Combinatorial Optimization

- Complete methods
  - Guarantee to find for every finite size instance an (optimal) solution in bounded time.
  - E.g., constructive tree search in CP and branch & bound, branch & cut in ILP, other tree search methods.
  - Might need exponential computation time.
- Approximate methods
  - Cannot guarantee optimality, nor termination in case of infeasibility.
  - Obtain good-quality solutions in a significantly reduced amount of time.

Approximate Methods

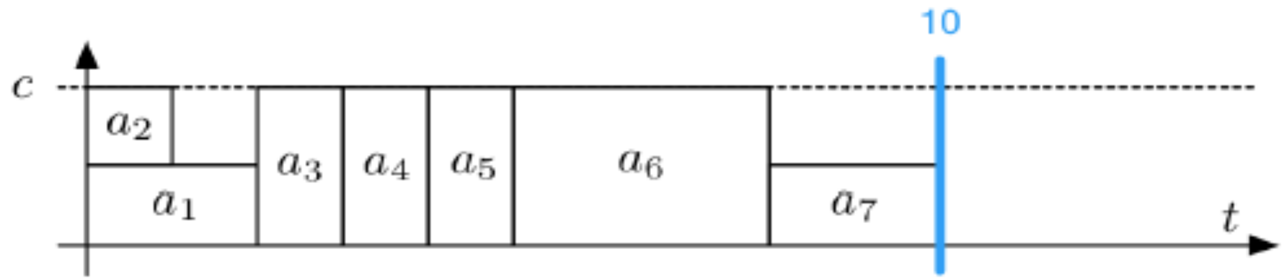
1. Constructive heuristics
2. Local search
3. Metaheuristics

Constructive Heuristics

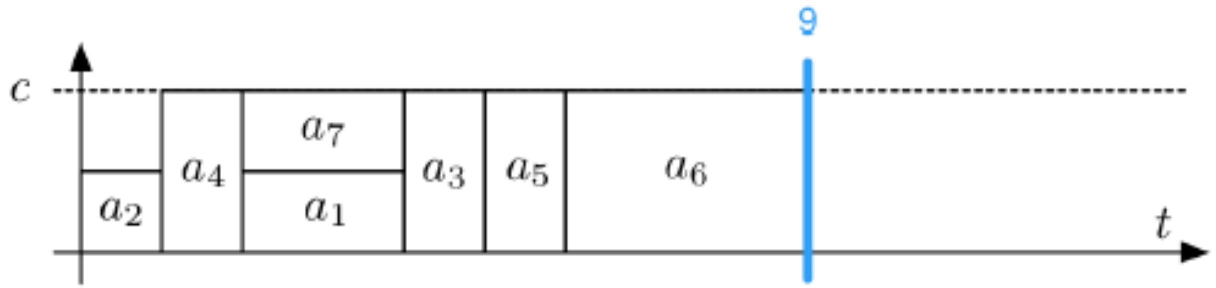
- Fastest approximation methods.
- Generate solutions from scratch by repeatedly extending the current partial assignments until a solution is found or stopping criteria are satisfied.
- Use problem-specific knowledge (heuristic) to construct a solution.
- A well-know class is greedy heuristics.
  - Make the locally optimal choice at each stage!

Priority Rule-Based Scheduling

- Schedule next the activity **i** with the minimum EST_i, breaking ties with the minimum LET_i.
- May not give the optimal solution.
  - A PRB solution.

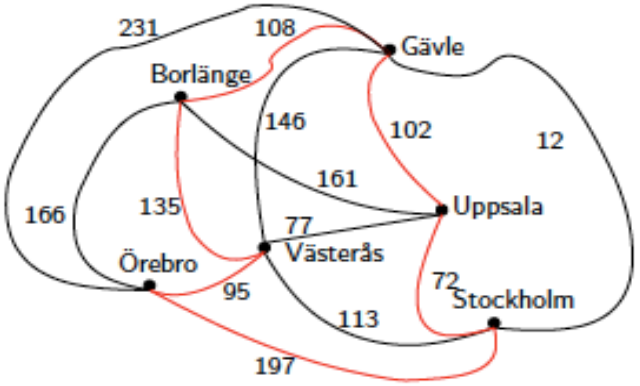


  - An optimal solution.



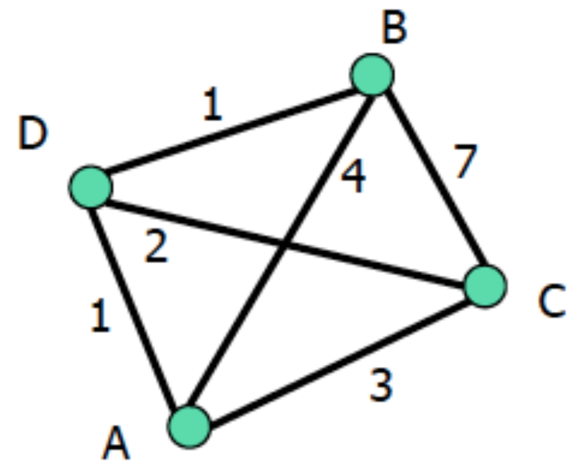
Travelling Salesman Problem (TSP)

- Given a list of connected cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
- When cities are seen as vertices V and the connections with distances as weighted edges E in a graph G = (V,E):
  - TSP is the minimum cost (i.e., total distance) Hamiltonian tour in G.



A Greedy Heuristic for TSP

- Visit next the unvisited city nearest to the current city.
- Nearest neighbour from A
  - A-D-B-C-A
  - Distance: 1+1+7+3 = 12
- Not necessarily optimal!
  - A-C-D-B-A
  - Distance: 3+2+1+4=10



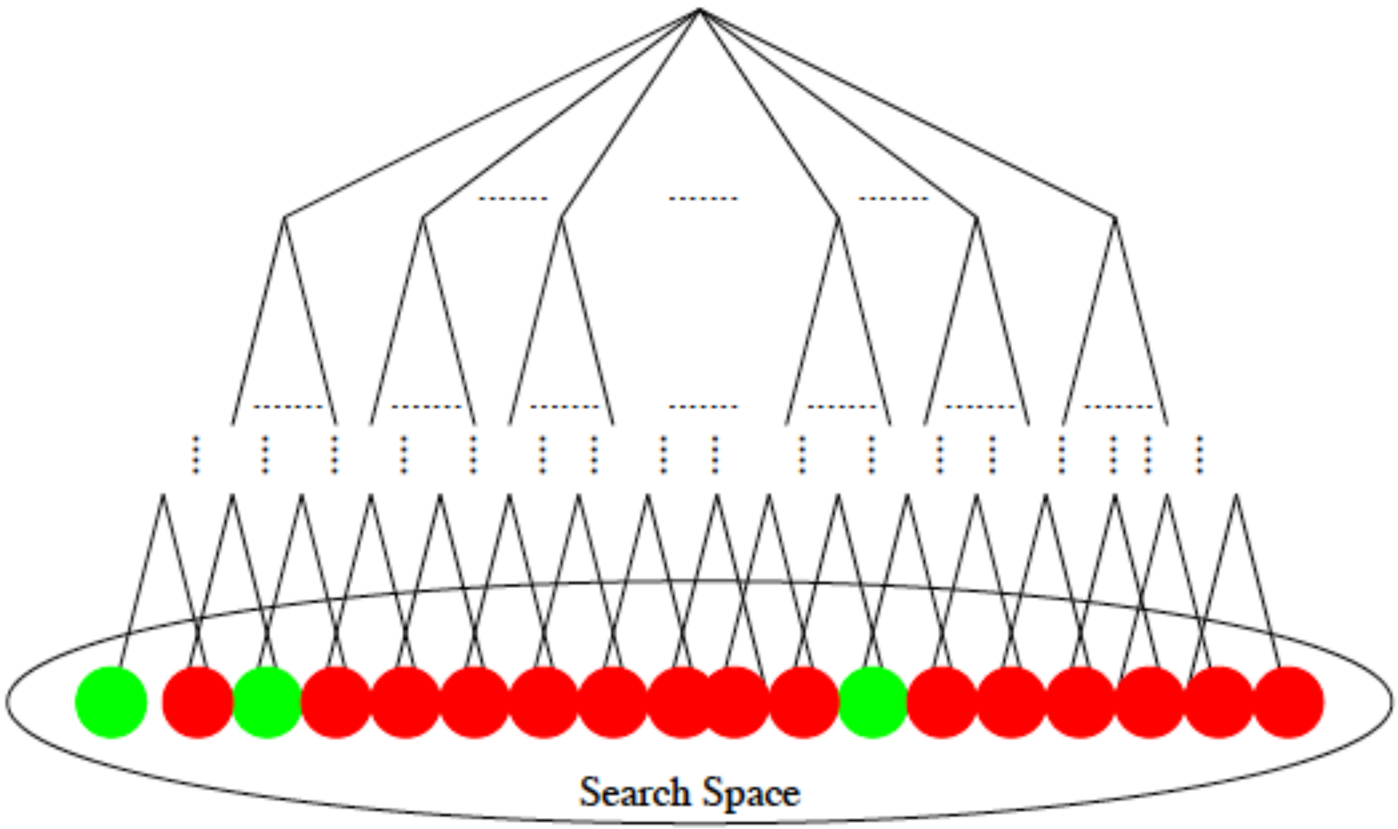
Constructive Heuristics

- Simple, quick and often give good approximations.
- Solutions maybe far from optimal!
  - Commit to certain choices too early.
- Widely used together with other methods.
  - E.g., for initialization for local search and metaheuristics.

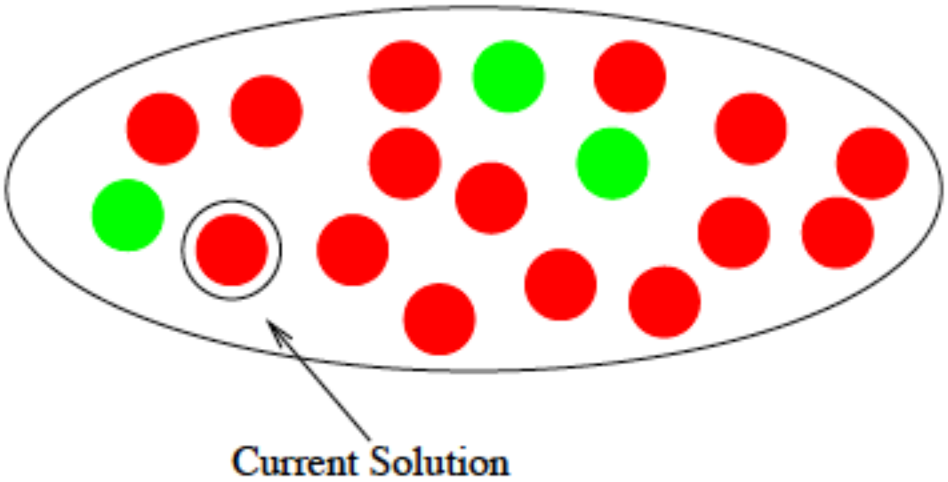
Local Search

- Often returns solutions of superior quality when compared to constructive heuristics.
- Starts from some initial solution and iteratively tries to replace the current solution with a better one in an appropriately defined neighbourhood by applying small (local) modifications.
- Can also start from an unfeasible assignment of all the variables.

Search Space in Constructive Tree Search



Search Space in Local Search



Combinatorial Optimization

- Given **<X,D,C,f>**, find a feasible solution s^* $\in S$ such that $f(s^*) \leq f(s)$ for all $s \in S$.

Neighbourhood Structure

- A function $\mathbf{N} : S \rightarrow 2^S$ that assigns to every $\mathbf{s} \in S$ a set of neighbours $N(\mathbf{s}) \subseteq S$. $N(s)$ is called the neighbourhood of **s**.
- Often implicitly defined by specifying the modifications that must be applied to **s** in order to generate its neighbours N(**s**).
- The application of such an operator to **s** that produces a neighbour is commonly called a move.

Local Minimum

- A locally minimal solution (or local minimum) with respect to a neighbourhood structure N is a solution s' such that $f(s') \leq f(s)$ for all $s \in N(s')$.

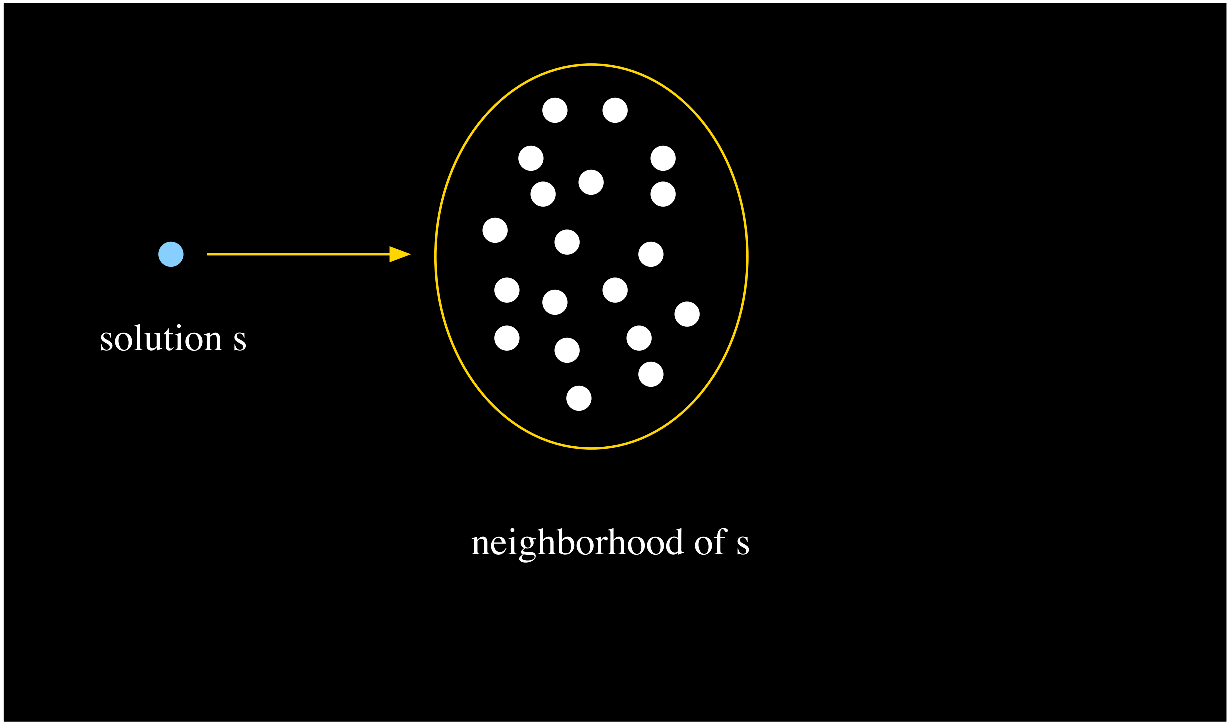
A Simple Local Search Algorithm

Algorithm 1 Iterative improvement local search

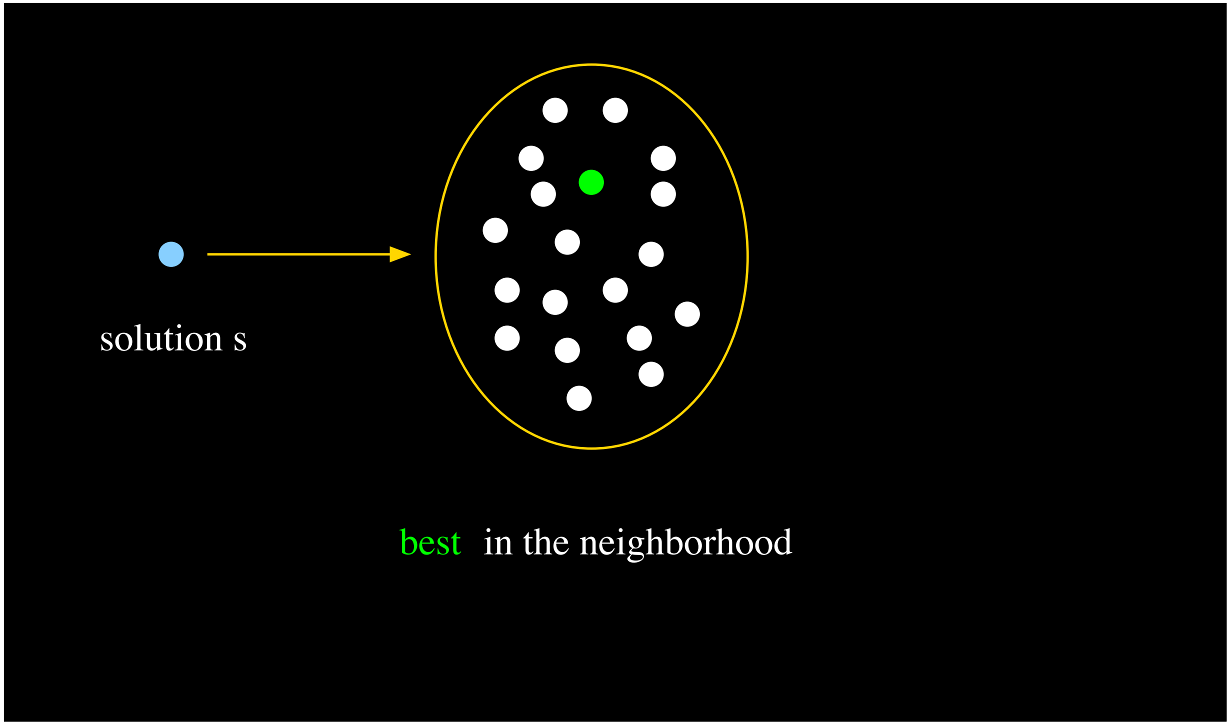
1: $s \leftarrow \text{GenerateInitialSolution}()$
2: **while** $\exists\, s' \in \mathcal{N}(s)$ such that $f(s') < f(s)$ **do**
3: $\quad s \leftarrow \text{ChooseImprovingNeighbor}(\mathcal{N}(s))$
4: **end while**

- Initial solution can be generated randomly or heuristically.
- A move is only performed if the resulting solution is better than the current solution (also called hill climbing).
- ChooseImrovingNeighbor: first improvement, best improvement.
- Stops as soon as it reaches a local minimum.
  - Performance highly depends on the neighbourhood structure.

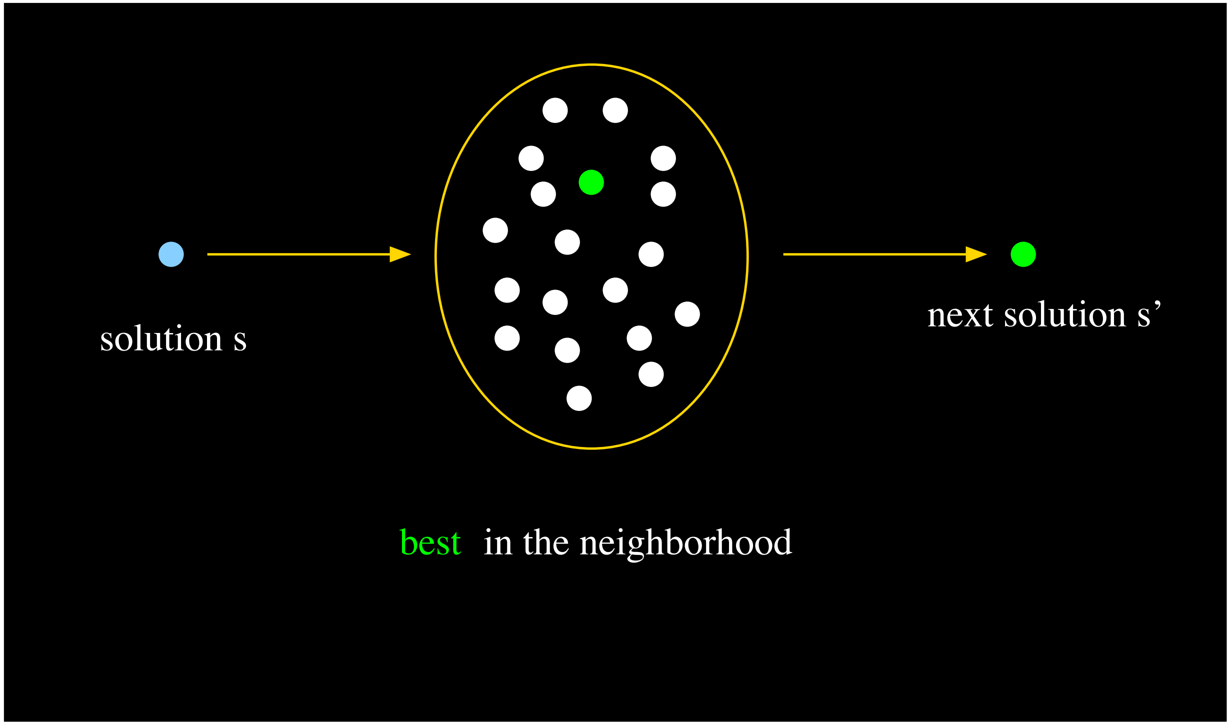
Iterative Improvement



Iterative Improvement



Iterative Improvement



Iterative Improvement for TSP

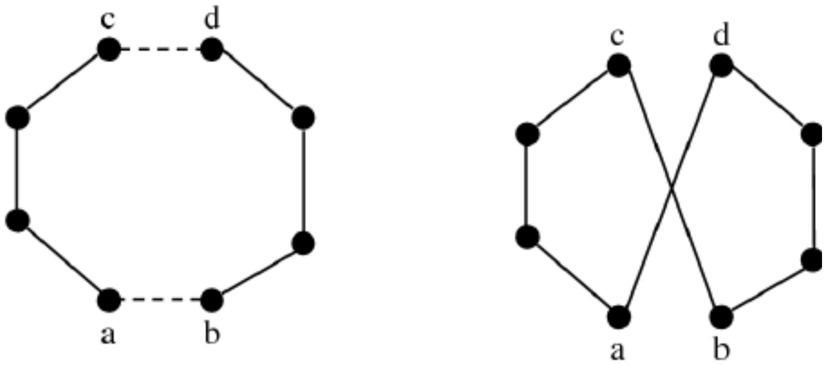
- Initial solution
  - Any Hamiltonian tour.
  - Can be generated easily, e.g., by using the nearest neighbour constructive heuristic.
- Neighbourhood structure?

Iterative Improvement for TSP

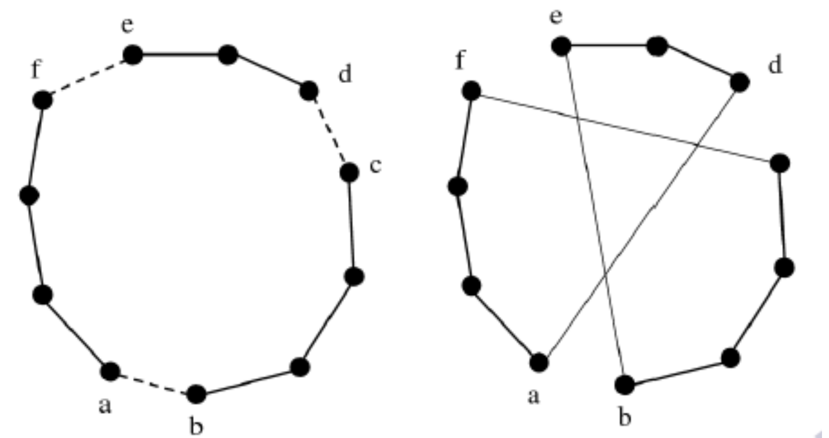
- Initial solution
  - Any Hamiltonian tour.
  - Can be generated easily, e.g., by using the nearest neighbour constructive heuristic.
- Neighbourhood structure
  - Arc exchanges.

K-exchange Neighbourhood

2-
exchange

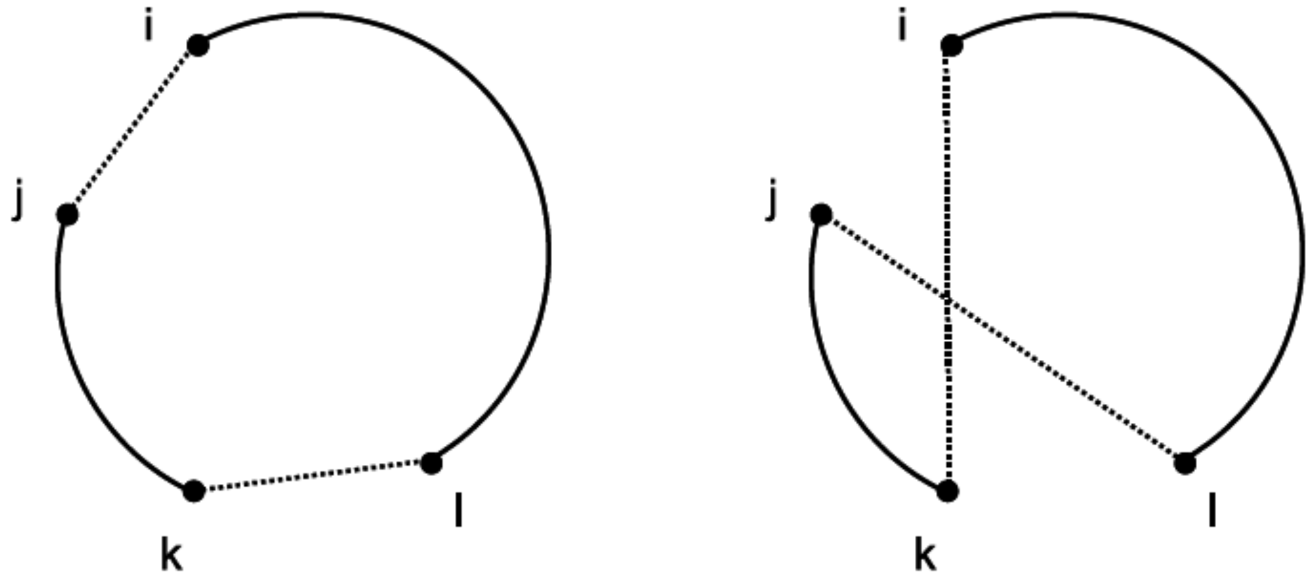


3-
exchange



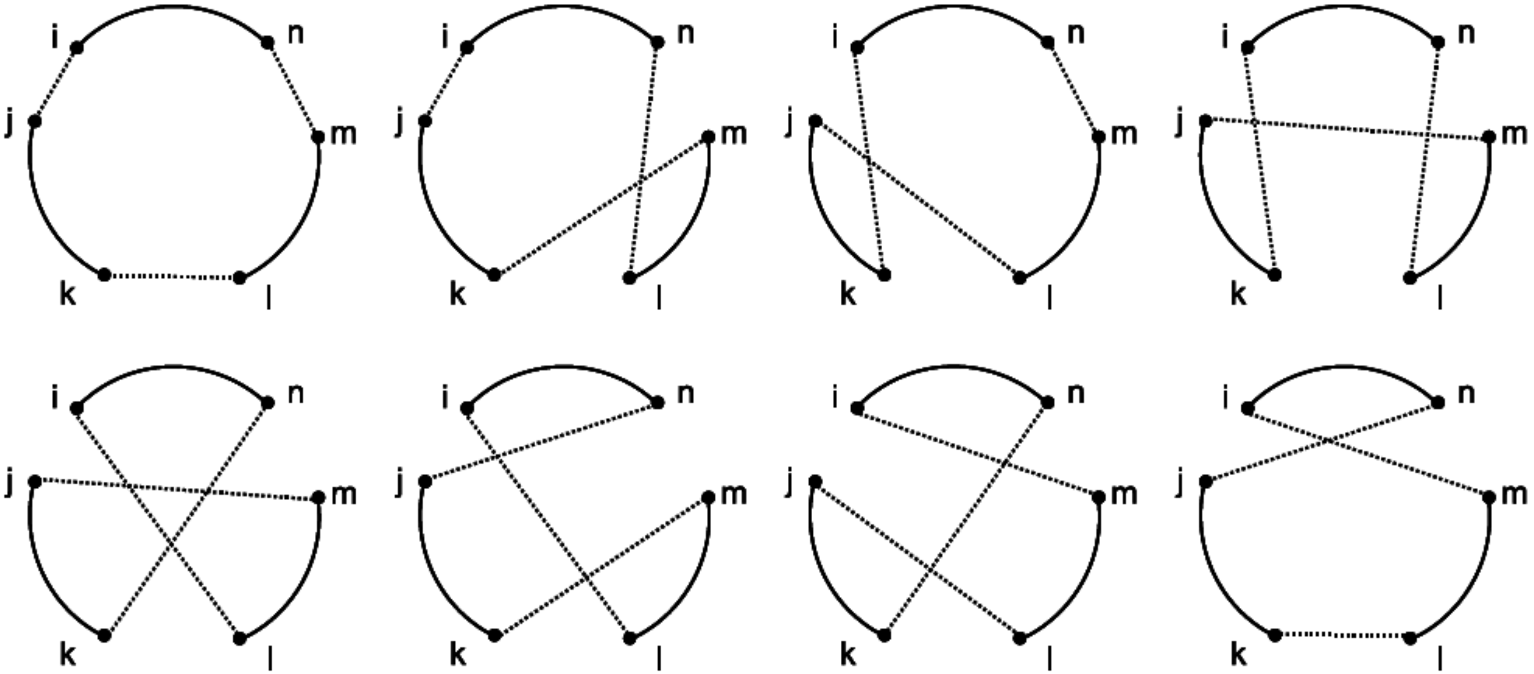
2-exchange

- For a pair of edges, only one alternative.



3-exchange

- For a triple of edges, 2^3-1 alternatives.

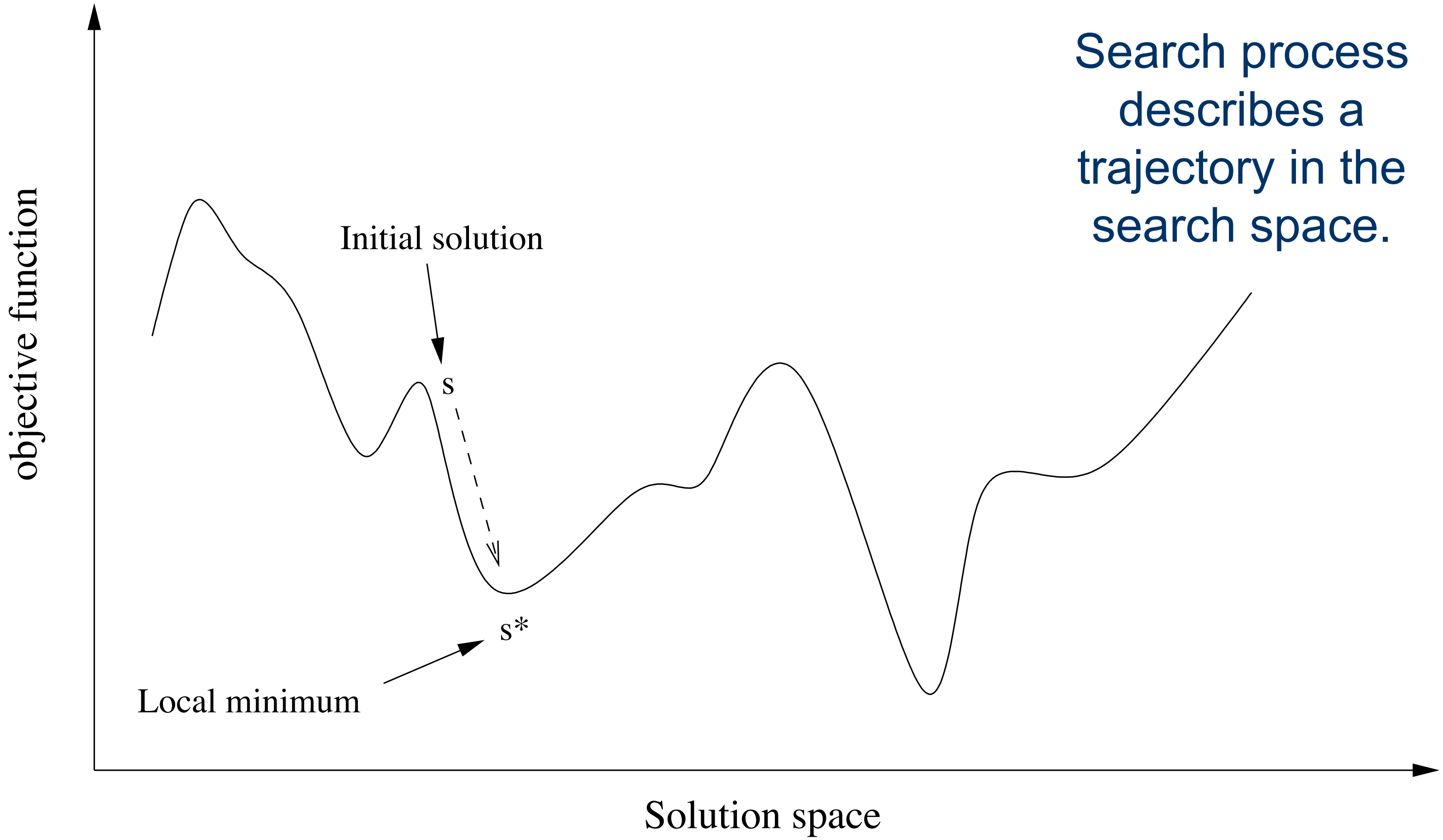


An Iterative Improvement Algorithm for TSP

1. Build an initial tour using the nearest neighbourhood heuristic.
2. Select randomly an edge from that tour.
3. Make a 2-e move with all the other edges of the tour and select the best tour therefore generated.
4. If it is better than the current tour then make it the current tour and go to 2.
5. Else, STOP. The local minimum is reached.

→ Different neighbourhood structures result in different algorithms.

A Pictorial View of Iterative Improvement



Search process describes a trajectory in the search space.

Metaheuristics

- High level strategies to increase performance.
  - Use of a priori knowledge (heuristics).
  - Exploitation of search history – adaptation.
  - General strategies to balance intensification and diversification.
  - Randomness and probabilistic choices.
- Aim: not to get trapped in local minima and search for better and better local minima.

Intensification & Diversification

- 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.

Intensification & Diversification

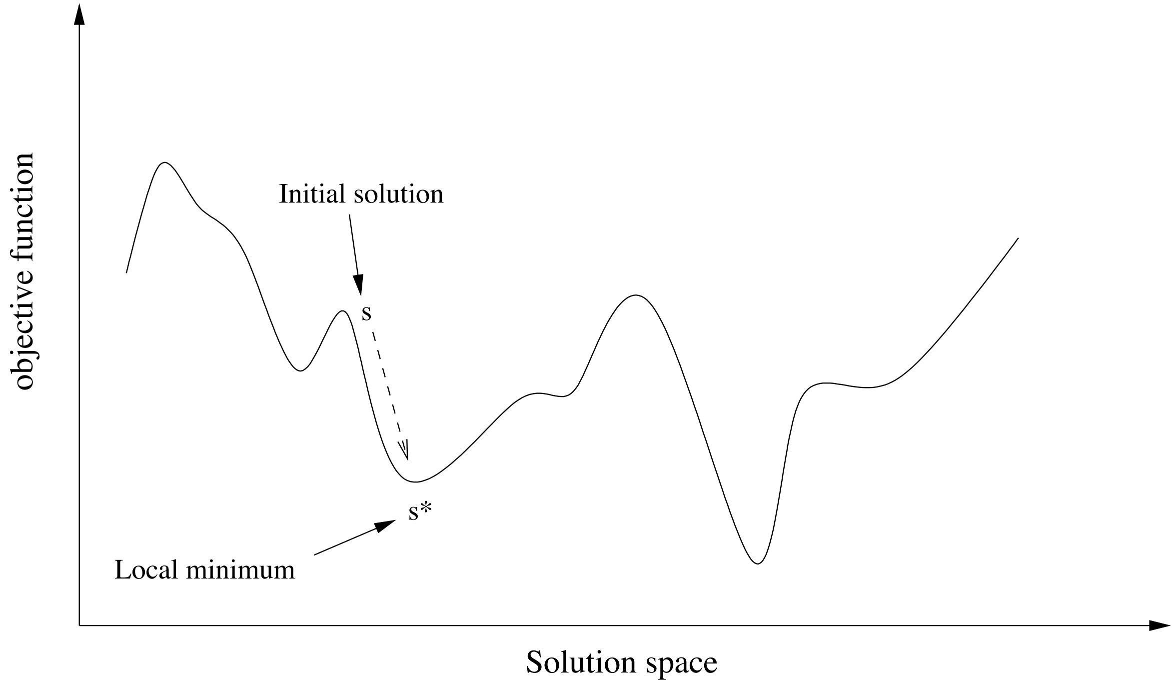
- Contrary and complementary:
  - need to quickly identify regions in the search space with high quality solutions, without wasting too much time in the regions already explored or not containing high quality solutions;
  - their dynamical balance determines the effectiveness of the metaheuristics.

Metaheuristics

- Encompass and combine:
  - constructive methods (e.g. random, heuristic, adaptive, etc);
  - local search-based (trajectory) methods;
  - population-based methods.

LS-based Methods

- Similarly to LS:
  - A single solution is used at each algorithm iteration.
  - Search process describes a trajectory in the search space.



LS-based Methods

- Differently from LS:
  - Add a diversification component to iterative improvement for escaping from local minima.
    - Allow worsening moves.
    - Change neighbourhood structure during search.
    - Change the objective function during search.
  - Termination criteria: maximum CPU time, maximum number of iterations (without improvement), a solution of sufficient quality, etc.

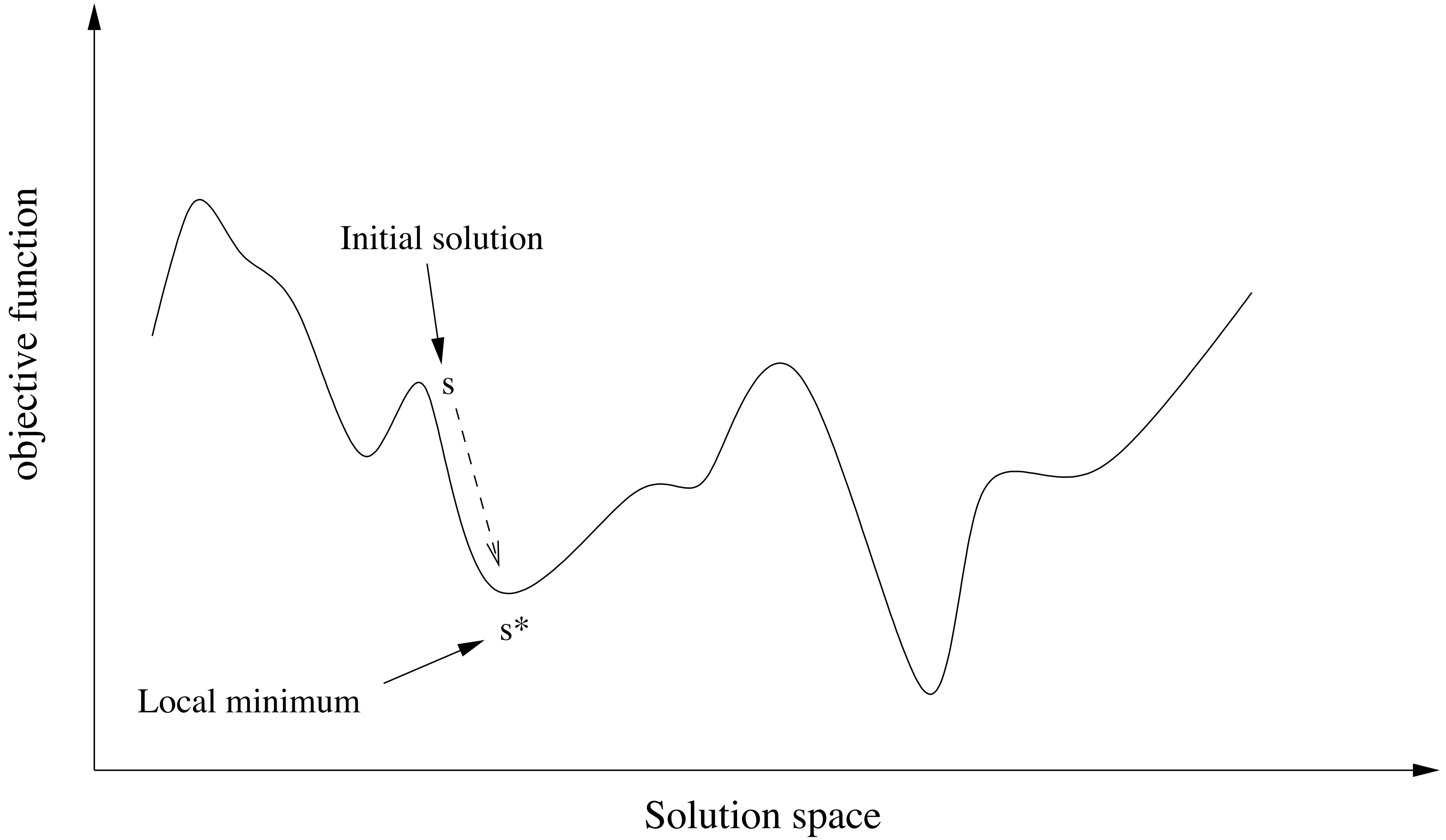
Some LS-based Methods

- Simulated Annealing (SA)
- Variable Neighbourhood Search (VNS)
- Tabu Search (TS)
- Guided Local Search (GLS)
- Iterated Local Search (ILS)
- Greedy Randomized Adaptive Search Procedure (GRASP).

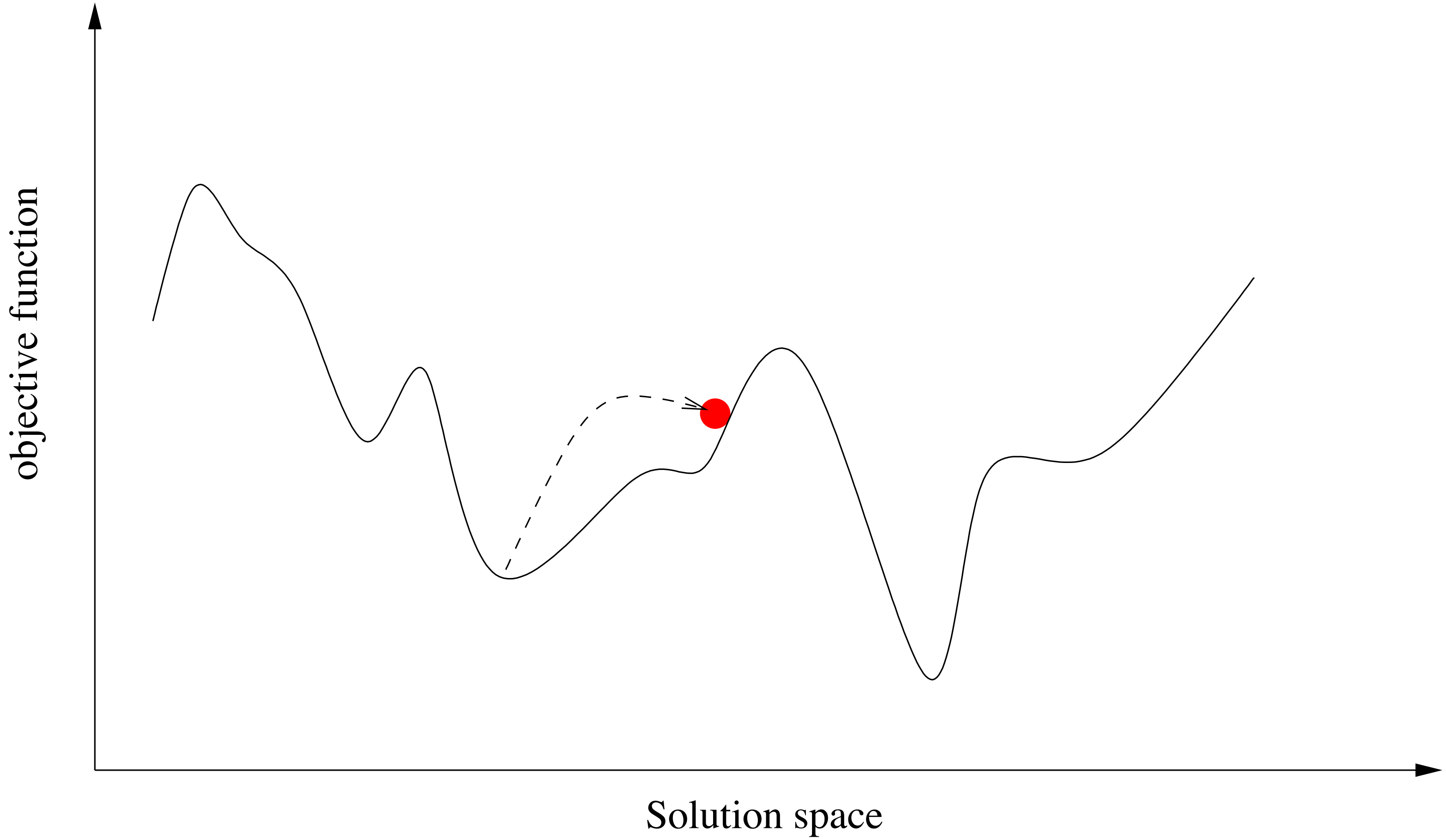
Simulated Annealing

- Accept worsening (up-hill) moves, i.e., the search moves toward a solution with a worse objective function value.
- Intuition: climb the hill and go downward in another direction in the same search landscape.
- The probability of doing such a move is decreased during search, favouring intensification to diversification.

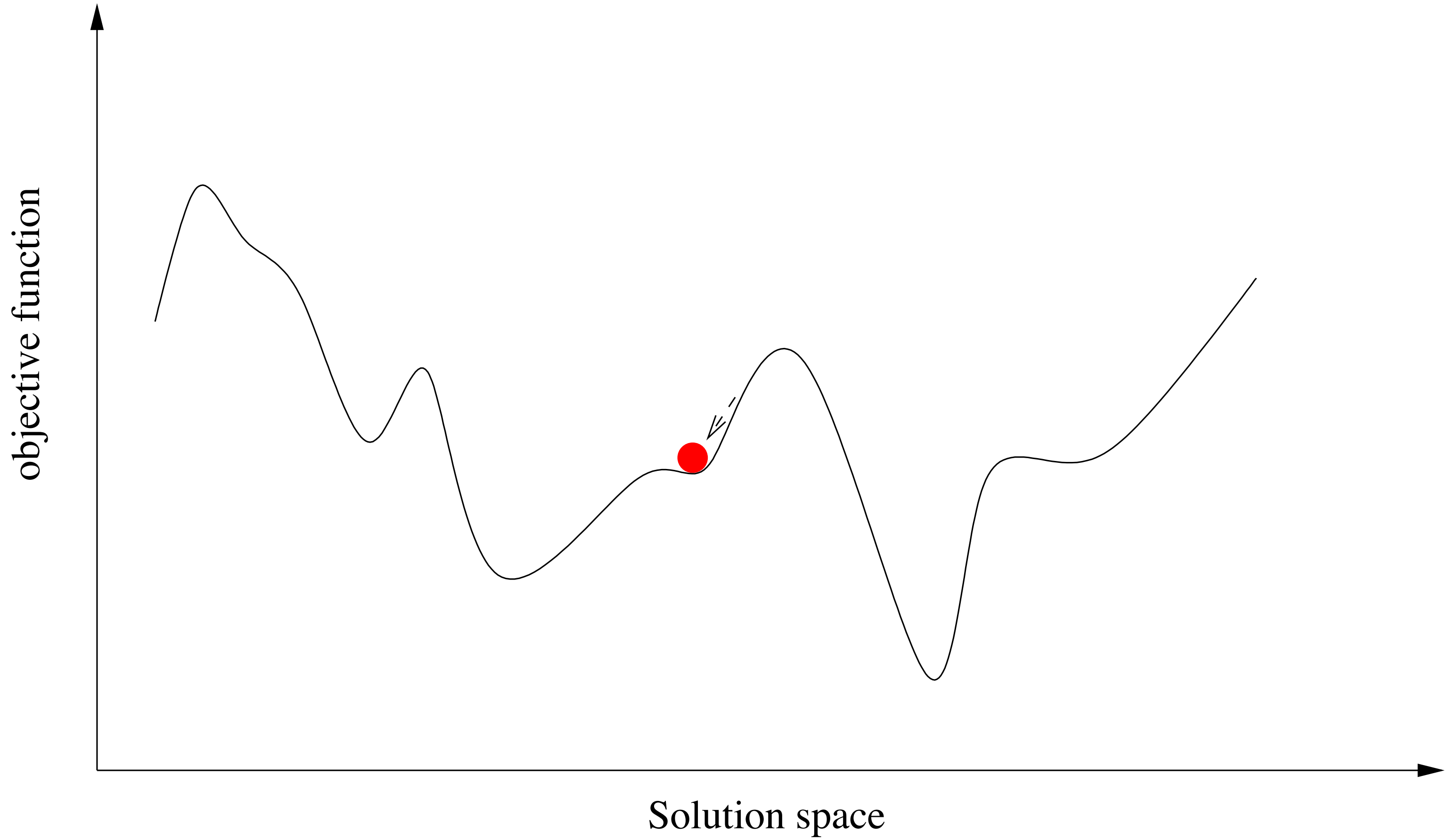
A Pictorial View of SA



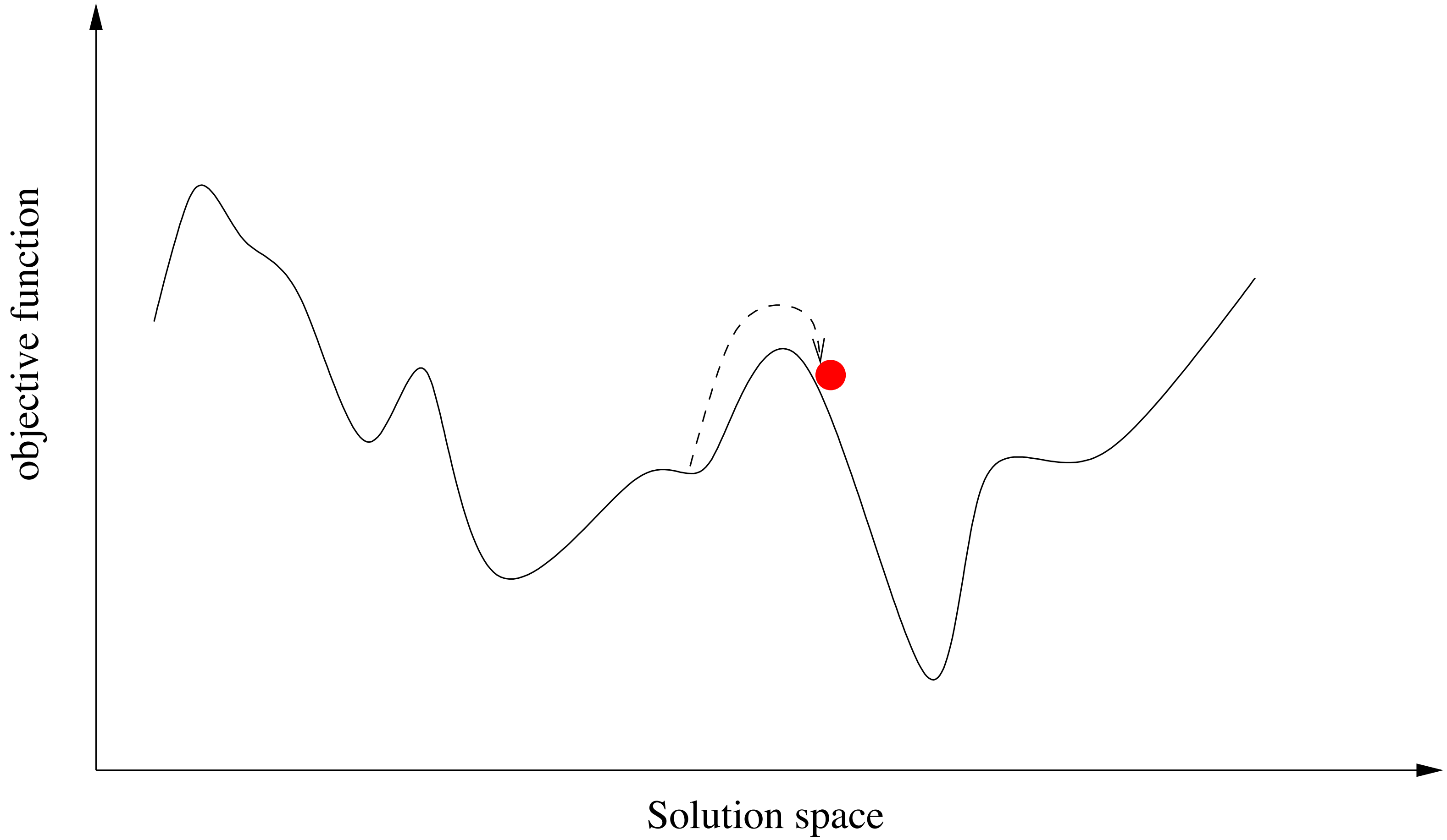
A Pictorial View of SA



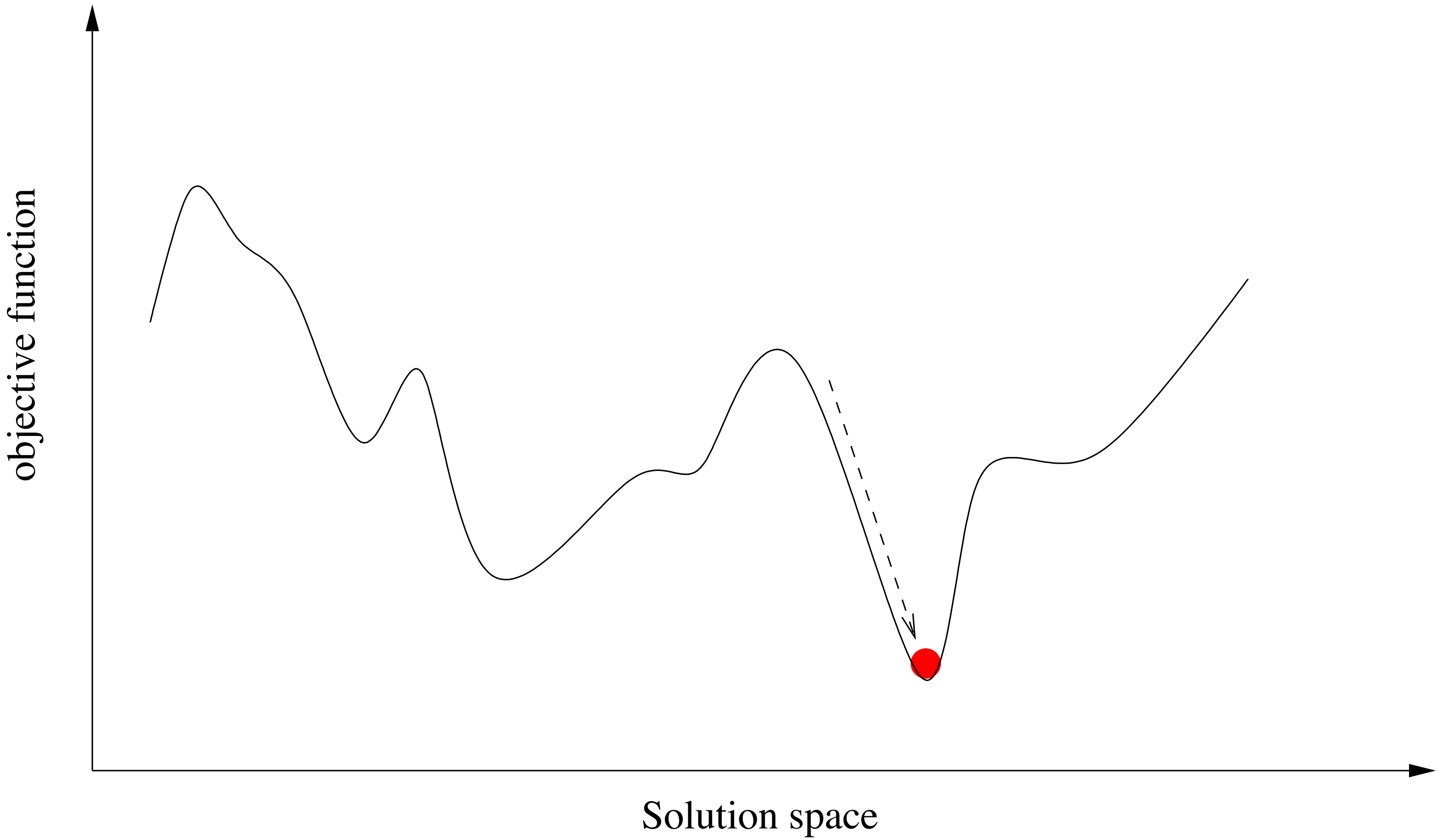
A Pictorial View of SA



A Pictorial View of SA



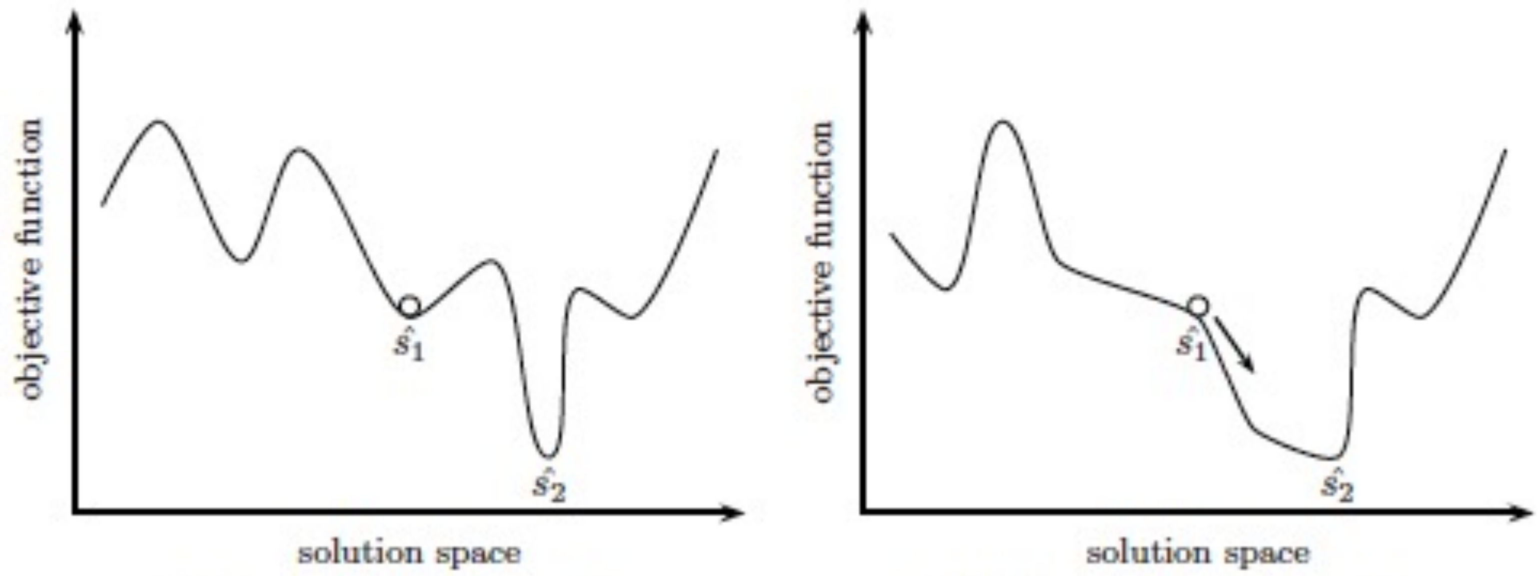
A Pictorial View of SA



Variable Neighbourhood Search

- Change neighbourhood structure during search.
- Intuition: different neighbourhoods generate different search landscapes.
- A neighbourhood N_i is substituted by a neighbourhood N_j as soon as local minima is reached.

Variable Neighbourhood Search

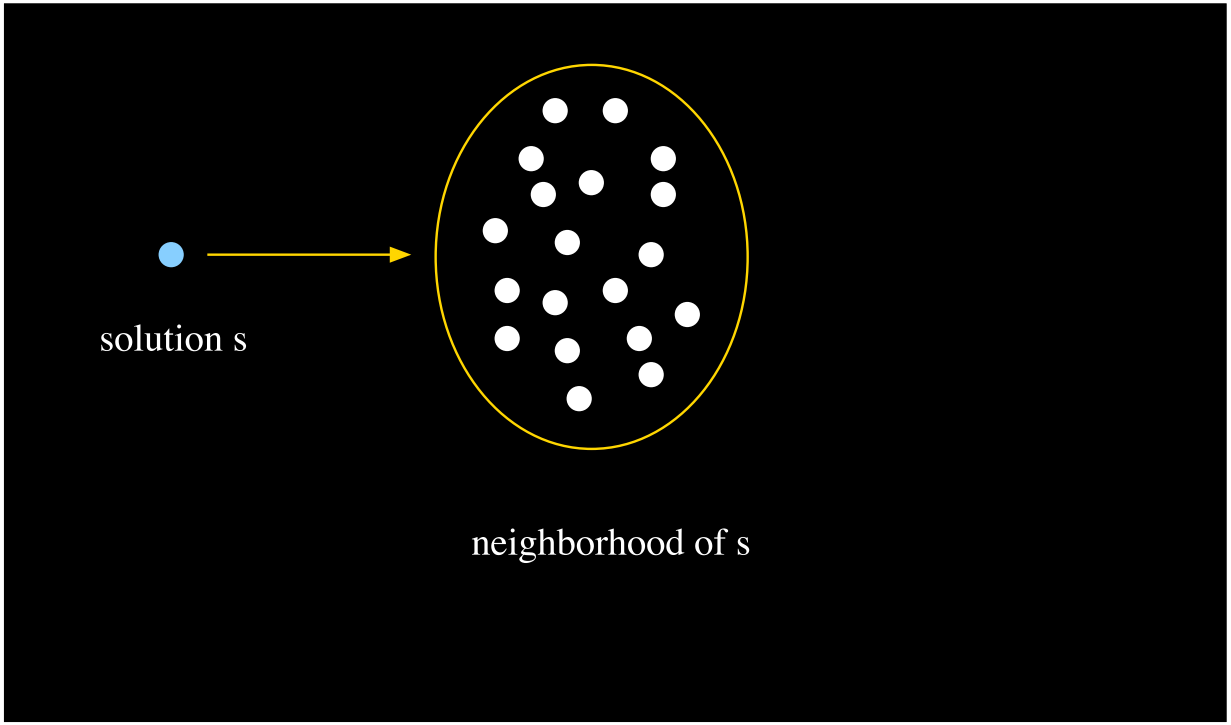


(a) Search landscape 1 (b) Search landscape 2

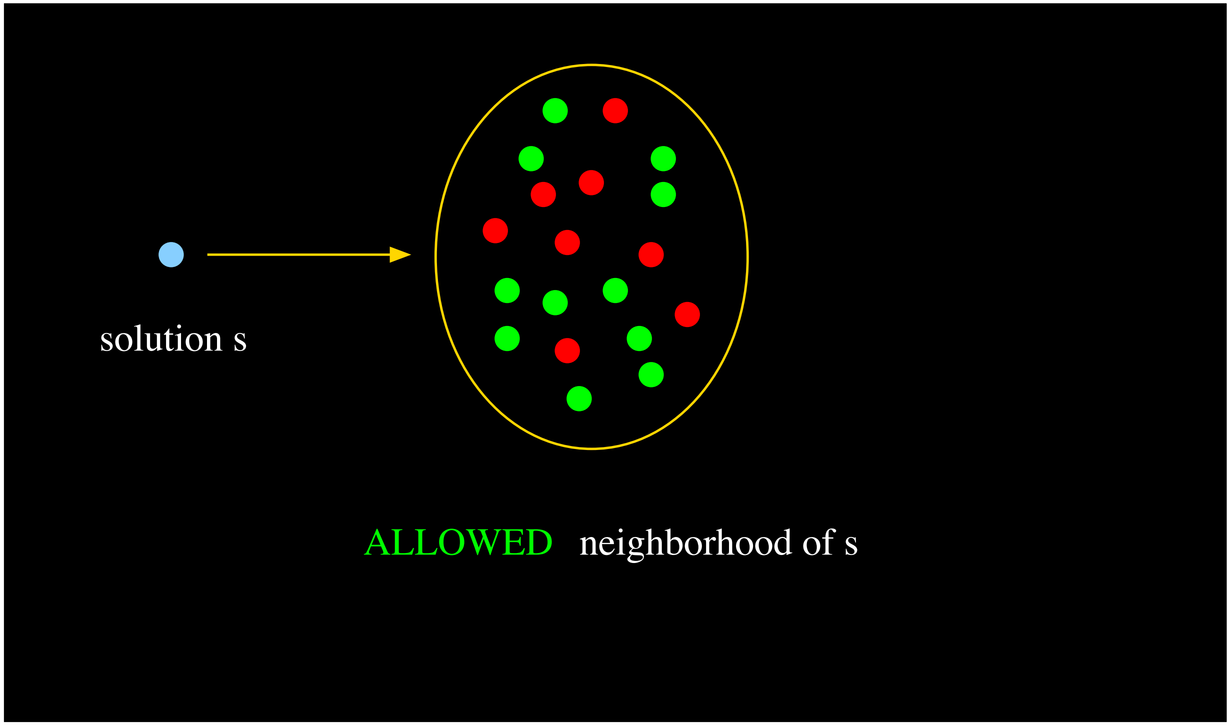
Tabu Search

- Change neighbourhood structure during search by exploiting the search history.
- Tabu list: keeps track of recently visited solutions/moves and forbids them.

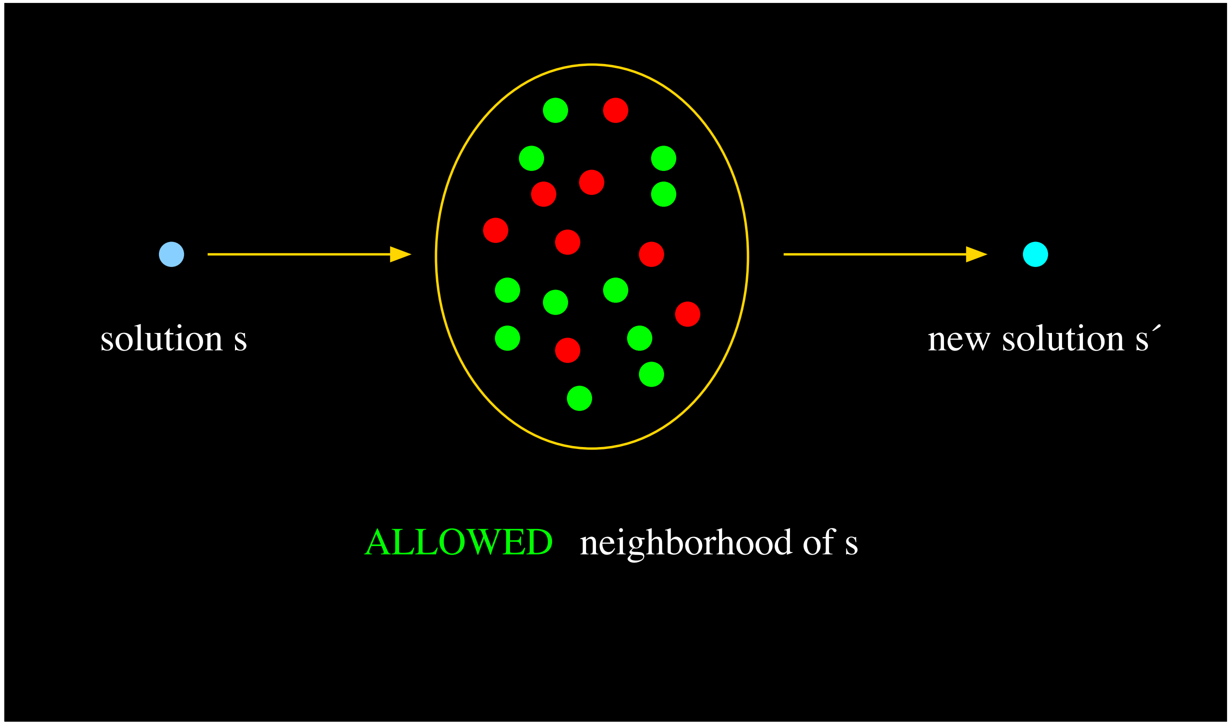
A Pictorial View of TS



A Pictorial View of TS



A Pictorial View of TS



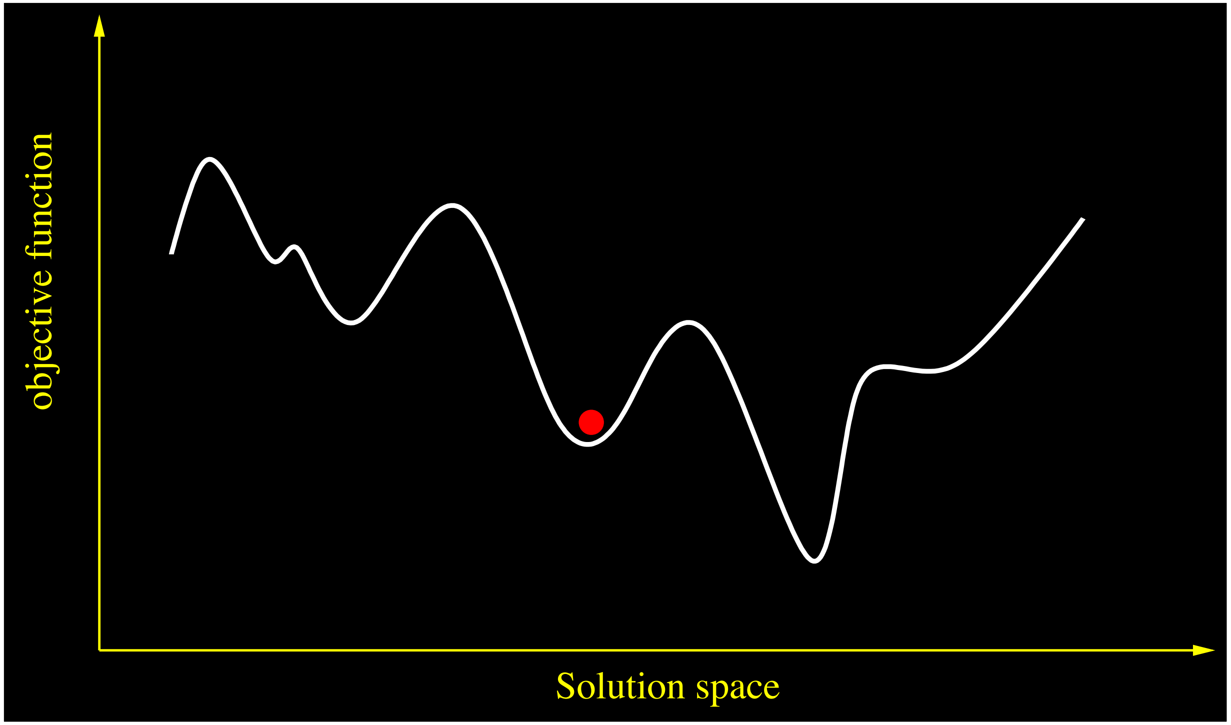
Tabu Search

- Storing solutions is often inefficient:
  - better store the moves;
  - but that could eliminate good yet not visited solutions.
- Aspiration criteria: accept a forbidden move towards a solution better than the current one.
- Tabu list size determines the size of exploration, favouring diversification to intensification as the size increases.
  - Dynamic tabu size is of interest!
  - Increase in case of repetitions (thus diversification is needed).
  - Decrease in case of no improvements (thus intensification should be boosted).

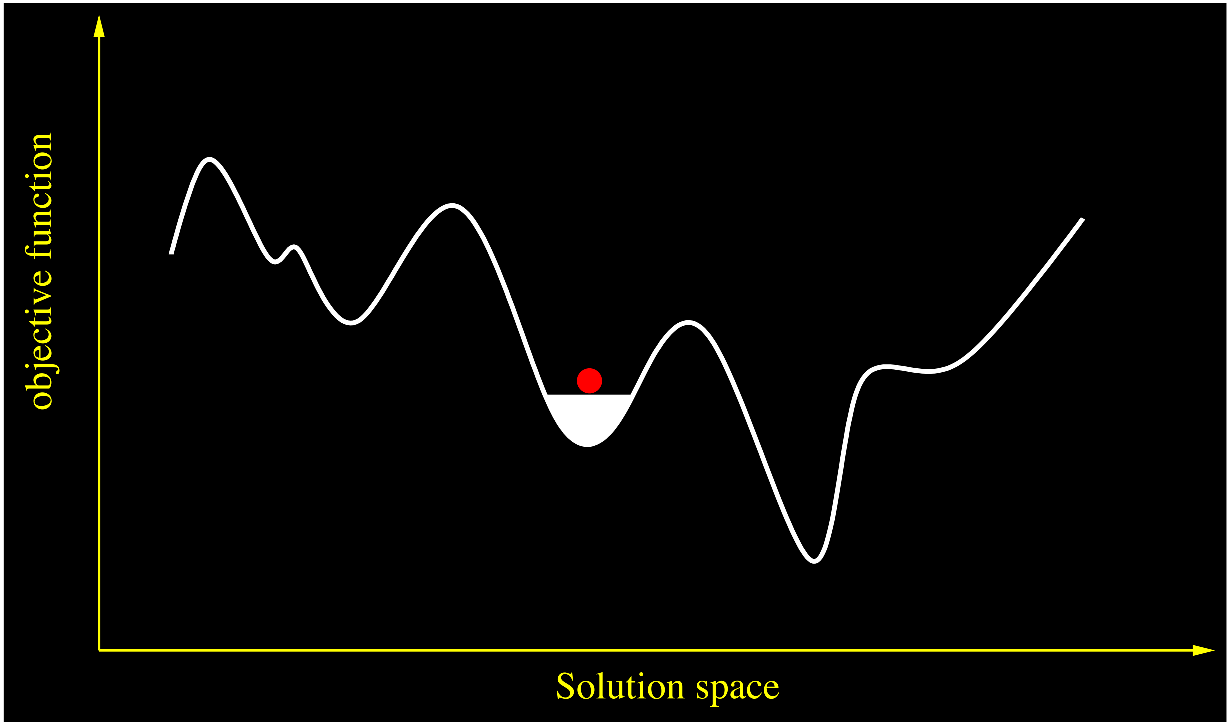
Guided Local Search

- Change the objective function during search so as to “fill in” local minima.
- Intuition: modify the search landscape with the aim of making the current local optimum less desirable.
- Penalize solution features that occur frequently in visited solutions.
  - E.g., certain arcs in a tour in TSP.
- New objective function takes into account these penalties.

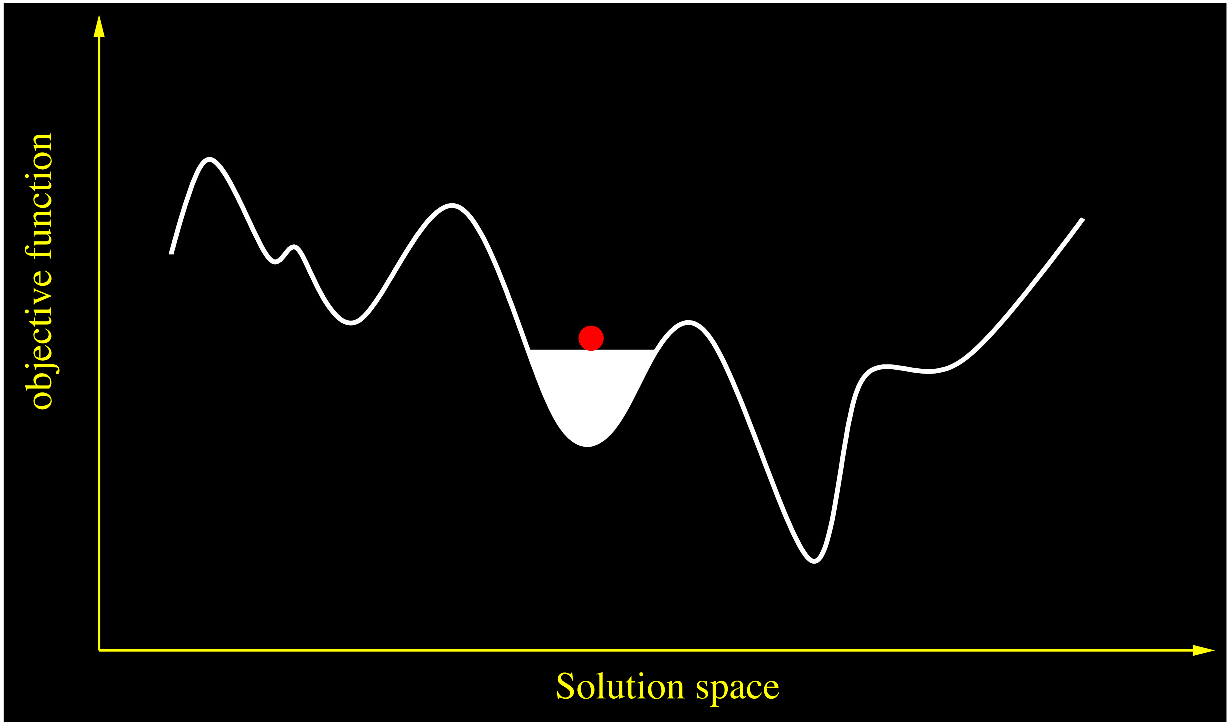
A Pictorial View of GLS



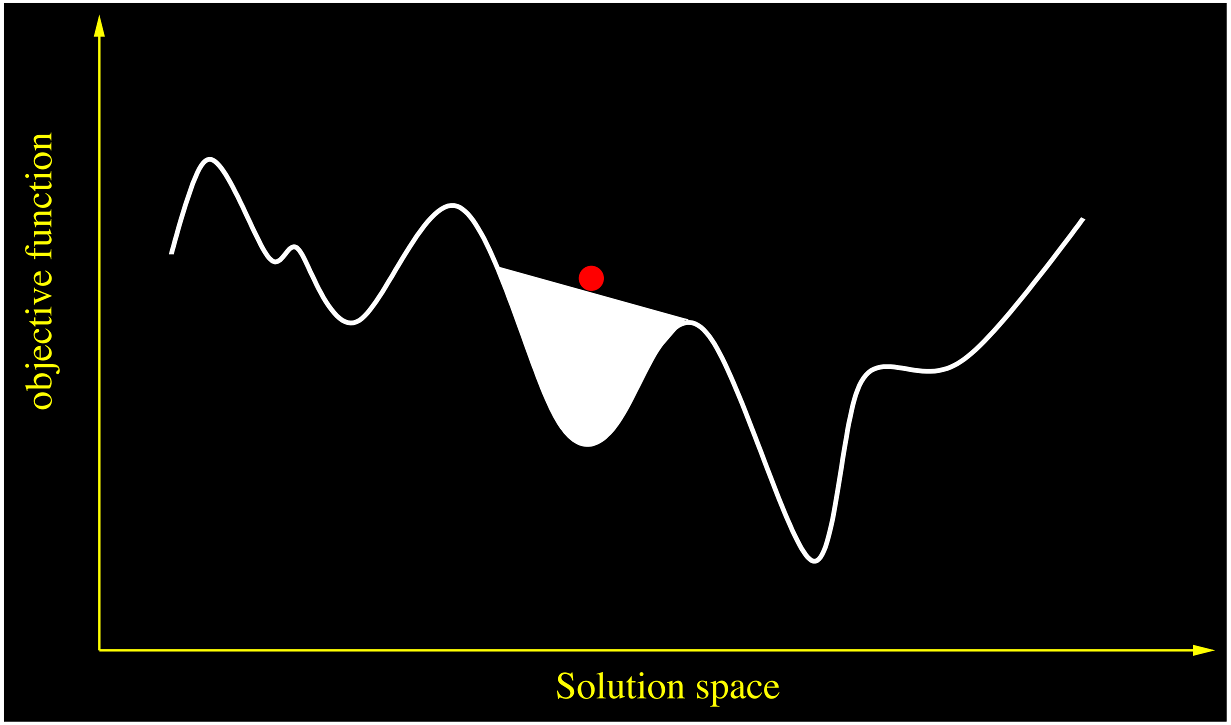
A Pictorial View of GLS



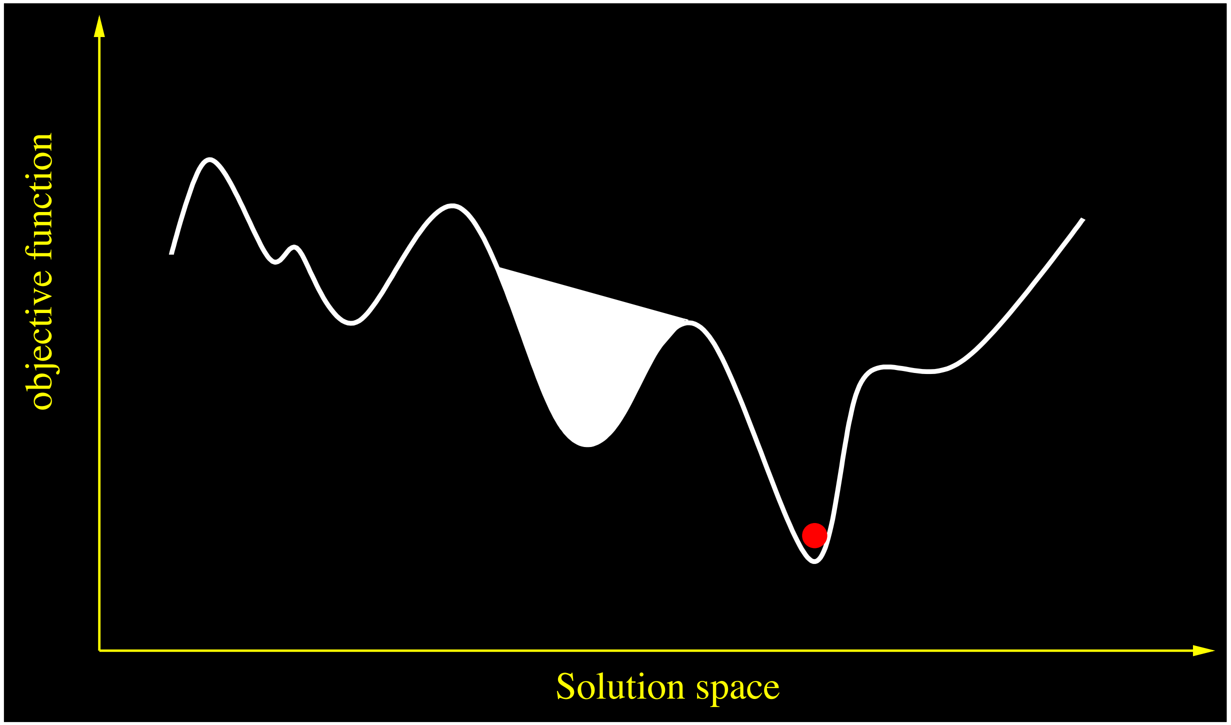
A Pictorial View of GLS



A Pictorial View of GLS



A Pictorial View of GLS



Metaheuristics

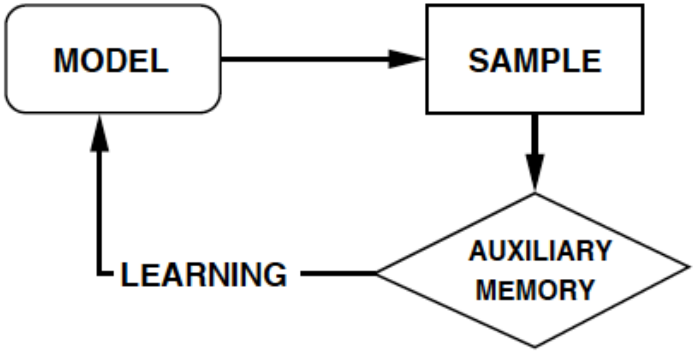
- Encompass and combine:
  - constructive methods (e.g. random, heuristic, adaptive, etc);
  - trajectory (local search-based) methods;
  - population-based methods.

Population-based Methods

- At each algorithm iteration, a set – population – of solutions are used.
- Search process is the evolution of a set of points or a probability distribution in the search space.
- Majority are inspired by natural processes, such as natural evolution and social insects foraging behaviour.
- Basic principle: learn correlations between good solution components.

Basic Principle

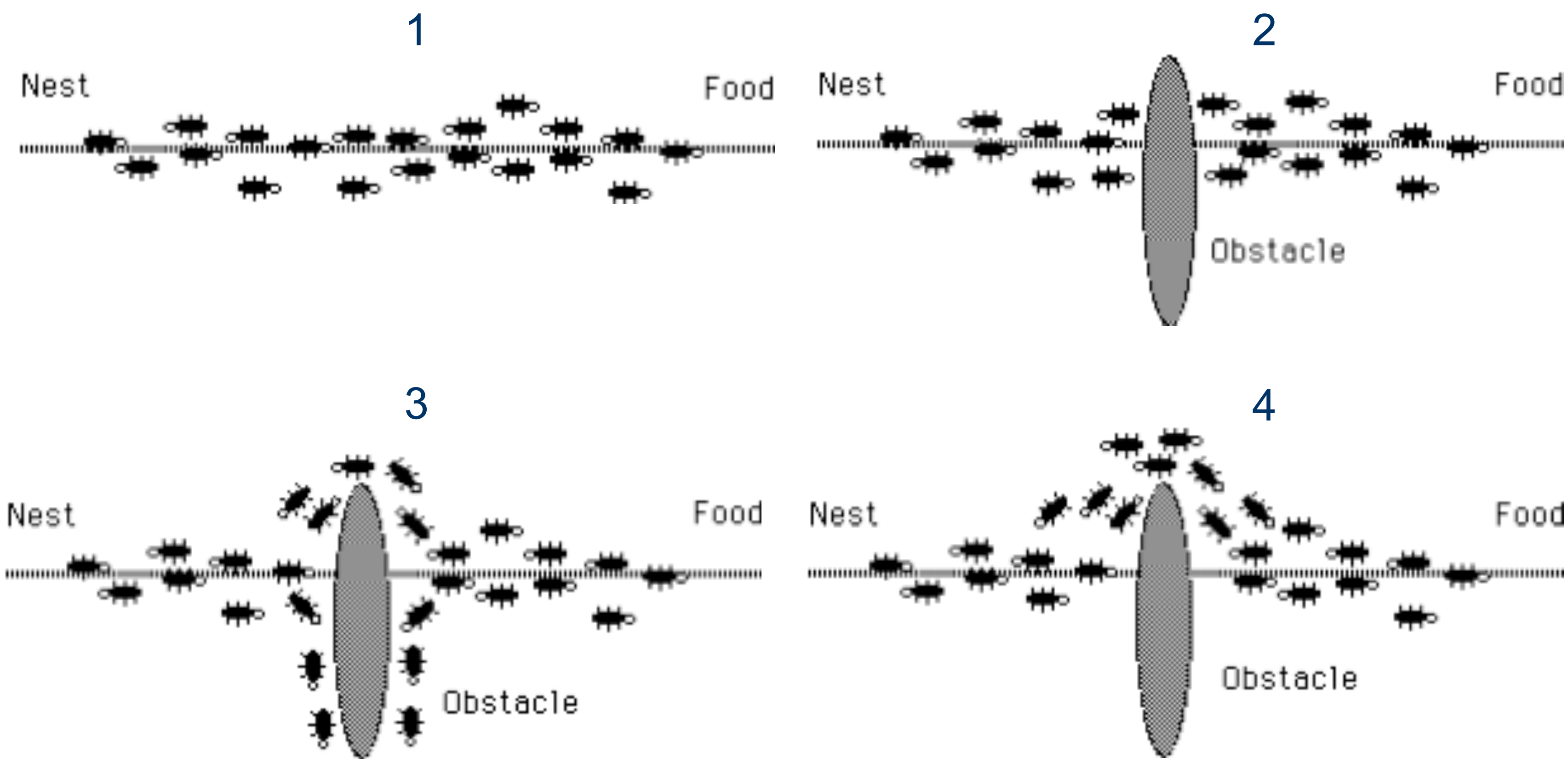
- Candidate solutions are generated using a parametrized probabilistic model.
- The model is updated using the previously seen solutions in such a way that the search will concentrate in the regions containing high quality solutions.
- E.g., Evolutionary Computation (EC), Ant Colony Optimization (ACO).



Ant Colony Optimization

- Inspired by the foraging behaviour of ants which enables them to find the shortest path between the nest and a food source.
- See the video.
  - 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 behaviour is the basis for a cooperative interaction which leads to the emergence of shortest paths.

Ant Foraging Behaviour



Ant Colony Optimization

- Pheromone trails are simulated by a parametrized probabilistic model – pheromone model.
  - Consists of a set of parameters whose values are called pheromone values.
  - Pheromone values act as the memory to keep track of the search process so as to intensify search around the best solution components.
  - E.g., a pheromone value $\tau(X_i,v_i)$ for all $X_i \in X$ and $v_i \in D(X_i)$ can represent the desirability of assigning v_i to X_i.
  - Bounding pheromone values between τ_{min} and τ_{max} can balance intensification and diversification.
  - Initially pheromone values are all set to τ_{max}.

Ant Colony Optimization

- Artificial ants employ constructive heuristics for probabilistically constructing solutions using the pheromone values.
  - Iteratively choose a variable X_i according to the heuristic and a value $v_i \in D(X_i)$ according to the probability:

Heuristic factor

$$p(x_i, v_i) = \frac{[\tau(x_i, v_i)]^\alpha \cdot [\eta(x_i, v_i)]^\beta}{\sum_{v_j \in D(x_i)} [\tau(x_i, v_j)]^\alpha \cdot [\eta(x_i, v_j)]^\beta}$$

  - Parameters α and β help to balance the influence of pheromone and heuristic factors.

Ant Colony Optimization

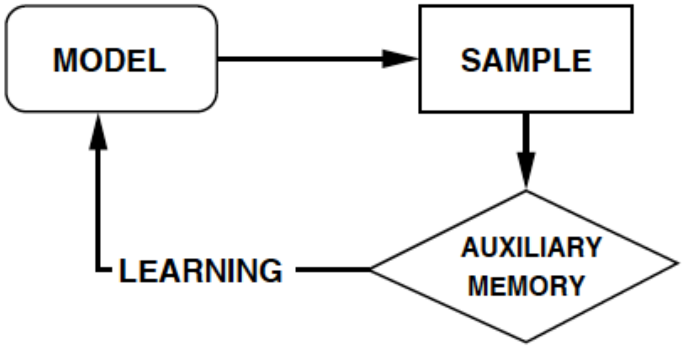
- Pheromone values are updated.
  - All pheromone values are decreased by an evaporation factor.

    → Allows ants to progressively forget older solutions and to emphasize to more recent ones (diversification).
  - Pheromone values associated to the assignments taking part of good solutions are increased proportionally to the quality of the solutions.

    → The goal is to increase the probability of selecting such assignments in the future constructions (intensification).

ACO Algorithm

1. Initialize pheromone values.
2. Ants construct solutions using heuristics and a pheromone model.
3. The constructed solutions are used to update the pheromone values in a way to bias the future sampling towards high quality solutions.



Termination criteria: maximum CPU time, maximum number of iterations (without improvement), a solution of sufficient quality, etc.

LS or Population-based Metaheuristics?

- LS-based methods are preferable when:
  - neighbourhood structures create a correlated search graph;
  - inventing moves is easy;
  - computational cost of moves is low.

LS or Population-based Metaheuristics?

- Population-based methods are preferable when:
  - solutions can be encoded as composition of good building blocks;
  - computational cost of moves in LS is high;
  - it is difficult to design effective neighbourhood structures;
  - coarse grained exploration (e.g., huge search spaces) is preferable.

Complementary Strengths

- LS-based methods
  - A promising area in the search space is searched in a more structured way.
  - Danger of being close to good solutions but “missing” them is low.
  - Intensification ability!
- Population-based methods
  - New solutions are obtained by recombining earlier solutions.
  - Search process performs a guided sampling of the search space, usually resulting in a coarse grained exploration.
  - Diversification ability!

Hybrid Metaheuristics

- Goal: Exploit complementary strengths of the individual strategies (synergy).
- The use of LS-based methods inside population-based methods.
  - E.g., application of local search to solutions constructed by ACO.
- Population-based iterated local search.
- Multilevel techniques.

Combinatorial Optimization

- Complete methods
  - Guarantee to find for every finite size instance an (optimal) solution in bounded time.
  - Might need exponential computation time.
  - E.g., constructive tree search in CP and branch & bound, branch & cut in ILP, other tree search methods.
- Approximate methods
  - Cannot guarantee optimality, nor termination in case of infeasibility.
  - Obtain good-quality solutions in a significantly reduced amount of time.

Complementary Strengths

- CP (a complete method)
  - Focus on constraints and feasibility.
  - Easy modelling and control of search.
  - Poor in optimization with loose bounds on the objective function.
- Metaheuristics (an approximation method)
  - Effective in finding good-quality solutions quickly.
  - Constraints are handled inefficiently, i.e. often by penalizing infeasible assignments in the objective function.

Metaheuristics + Complete Methods

- Main approaches
  - Metaheuristics are applied before complete methods providing a valuable input, or vice versa.
  - A complete method method applies a metaheuristic in order to improve a solution.
  - Metaheuristics use a complete method to efficiently explore the neighbourhood.
  - Metaheuristic concepts can also be used to obtain incomplete but efficient tree exploration strategies.

Metaheuristics + Complete Methods

- Main approaches
  - Metaheuristics are applied before complete methods providing a valuable input, or vice versa.
  - A complete method method applies a metaheuristic in order to improve a solution.
  - Metaheuristics use a complete method to efficiently explore the neighbourhood.
    - Large Neighbourhood Search
    - ACO + CP
  - Metaheuristic concepts can also be used to obtain incomplete but efficient tree exploration strategies.

Key Issues in LS-based methods

- Defining an appropriate neighbourhood structure.
- Choosing a way to examine the neighbourhood of a solution.

Small vs Large Neighbourhoods

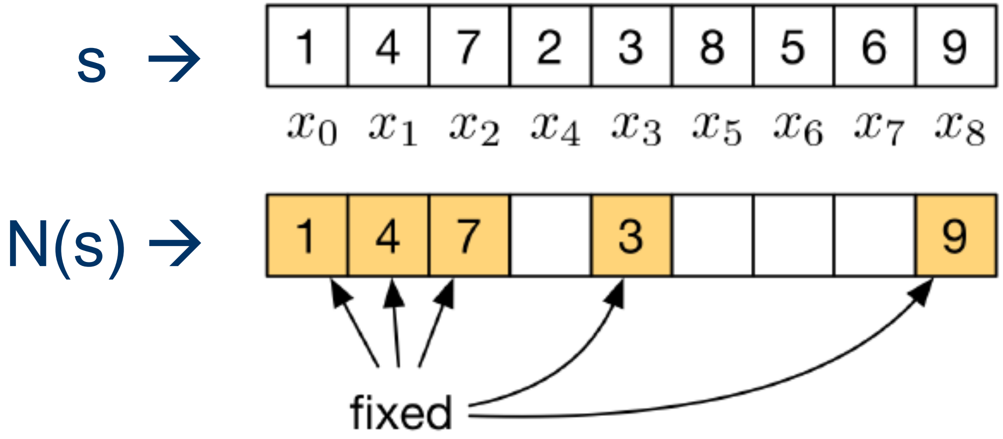
- Small neighbourhoods
  - PRO: it is fast to find an improving neighbour (if any).
  - CONS: the average quality of the local minima is low.
- Large neighbourhoods
  - PRO: the average quality of the local minima is high.
  - CONS: finding an improving neighbour might be difficult.

Large Neighbourhood Search

- Use a generic and large neighbourhood, and explore it with a complete method like CP!
- Core idea:
  - view the exploration of a neighbourhood as the solution of a sub-problem;
  - use tree search to exhaustively but quickly explore it.

Neighbourhood in LNS

- Given a solution s:
  - fix part of the variables to the values they have in s (called fragment);
  - relax the remaining variables.



Advantages over LS and CP

- Efficient neighbourhood exploration.
  - Thanks to propagation and advanced search techniques of CP.
- LNS is easier to develop than LS.
  - Easy and problem-independent neighbourhood definition.
  - No need to ensure that complicated constraints are satisfied.
- More scalable than using only CP on the problem.
  - Subproblems are typically much smaller.
  - We can control the subproblem size.
  - The fixed-variables reduce the domain sizes.
  - Propagation works best when domains are small.

Design Decisions

- Complete vs incomplete neighbourhood exploration.
  - Often incomplete.
- How many variables to fix?
  - The neighbourhood size should be large enough to diversify the search, but the complexity of solving it should be rather low.
  - Often hand-tuned for custom applications.
  - Automatic/adaptive techniques.
- Which variables to fix?
  - Random (ensures diversification).
  - Problem specific approaches.
  - Automatic/adaptive techniques.

ACO + CP

- Constructive techniques with complementary strengths:
  - ACO is characterized by a learning capability;
  - CP is efficient in handling constraints.
- Typical ACO + CP approaches:
  - Use CP as solution construction for artificial ants.
  - Use ACO for variable and value ordering heuristics in CP.

CP in ACO

- Artificial ants employ ~~constructive heuristics~~ CP for probabilistically constructing solutions using the pheromone values.
  - Iteratively choose a variable X_i according to the heuristic and a value $v_i \in D(X_i)$ according to the probability:

$$p(x_i, v_i) = \frac{[\tau(x_i, v_i)]^\alpha \cdot [\eta(x_i, v_i)]^\beta}{\sum_{v_j \in D(x_i)} [\tau(x_i, v_j)]^\alpha \cdot [\eta(x_i, v_j)]^\beta}$$

Heuristic factor

- Pheromone values are updated as usual.

ACO+CP followed by CP

- ACO+CP is performed and the final pheromone matrix is saved.
- The resulting solution provides an upper bound to CP.
- CP performs a complete search and uses the pheromone matrix as a heuristic information for value selection.