U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence

Lecture 3a: Optimization and Local Search

(based on Løkketangen, 2019)

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence

Agenda

- Overview of, and introduction to, modern heuristic optimization methods suitable for solving practical optimization problems
- **Topics:**
  - Combinatorial Optimization Problem (COP)
  - Formulating COPs
  - Local Search
  - Heuristics and Meta-Heuristics
  - Individual/Population based Metaheuristics

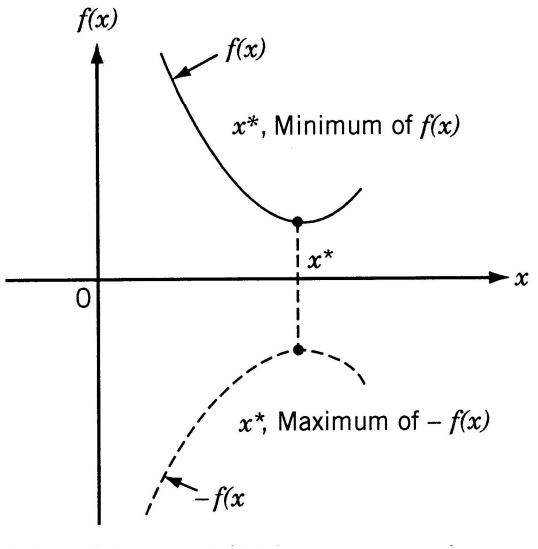
Optimization Problems

- **Many real-world problems involve getting a solution for maximizing or minimizing a value:**
  - How can a **car manufacturer** get the most parts out of a piece of sheet metal?
  - How can a **moving company** fit/transport the maximum amount of furniture in a truck of a given size?
  - How can a **telecommunications company** route communications to get the best use of its lines and connections?
  - How can a **university/school** schedule its classes to make the best use of teachers and classrooms without conflicts?

Optimization Problems

- **Optimization Problems (minimization):**

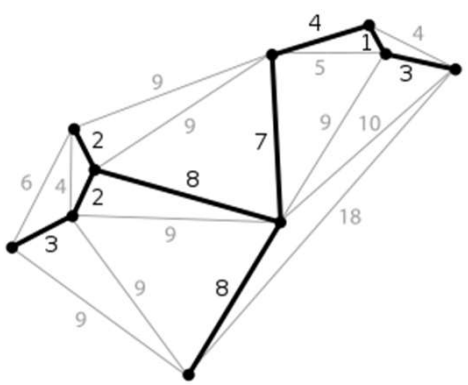
$$\text{minimize } f_0(x)$$
$$\text{subject to } g_i(x) \leq b_i, \quad i = 1,\ldots,m$$

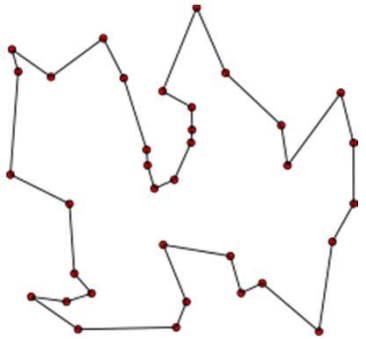


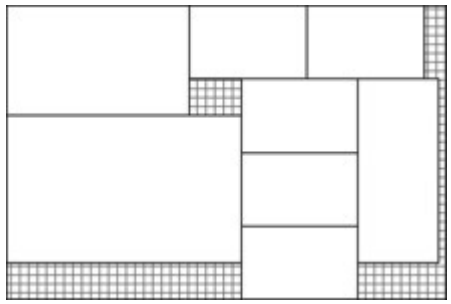
- ***$f_0 : R^n \rightarrow R$:***
  - Objective Function (calculates a value)
- ***$x = (x_1,\ldots,x_n)$:***
  - Controllable Variables (linearly independent)
- ***$g_i : R^n \rightarrow R$: ($I = 1,\ldots,m$):***
  - Constraints (may be of several other types)

Combinatorial Optimization Problems

- **Set of Feasible Solutions is Discrete**
- **Important applications in several fields:**
  - Artificial intelligence, machine learning, auction theory, software engineering, applied mathematics, theoretical computer science, …
- **About formal problems**
  - Formulations: COP, IP, MIP, …
  - Problems: TSP, VRP, Assignment, Set Covering, Cutting Stock, Minimum Spanning Tree…
- **How to solve problems**
  - Exact, Approximation, Heuristic algorithms
- **Why use heuristics?**
  - Complexity (**P** vs **NP**), combinatorial explosion







Practical Example: Choice of Projects

- A leading worldwide company may develop, during the next 3 months, a few projects, each with a known:
  - Total profit
  - Amount of resources needed
- The company has a fixed total amount of resources
- Which projects should the company select to maximize its total profit?
- **Strategic/Tactical Decision**
  - Optimization problem
  - May be modelled as: Knapsack Problem

COP Example: The Knapsack Problem

Knapsack Problem:

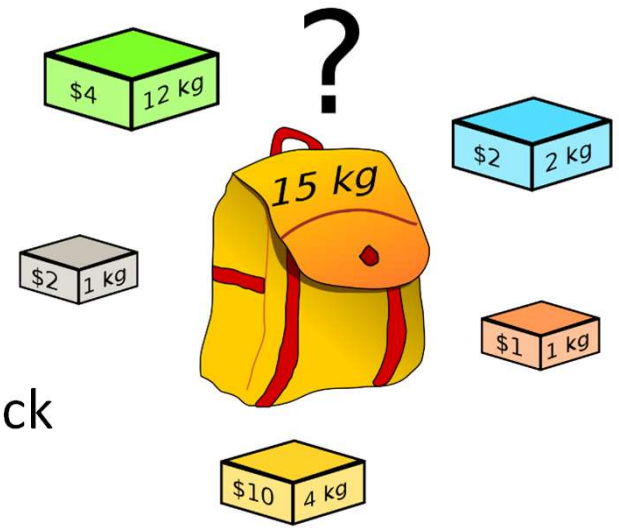
- ***n* items** *{1,...,n}* available, **weight a_i , value/profit c_i**
- A selection of items shall be packed in a knapsack with weight capacity *b*
- Find the selection of items that maximizes the total value/profit

$$\max \sum_{i=1}^{n} c_i x_i \quad \text{s.t.}$$

$$\sum_{i=1}^{n} a_i x_i \leq b$$

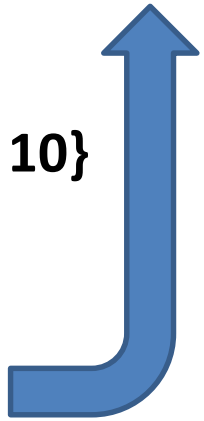
$$x_i = \begin{cases} 1 \\ 0 \end{cases}$$

If the item i is in the knapsack
otherwise



COP Example: Knapsack Problem

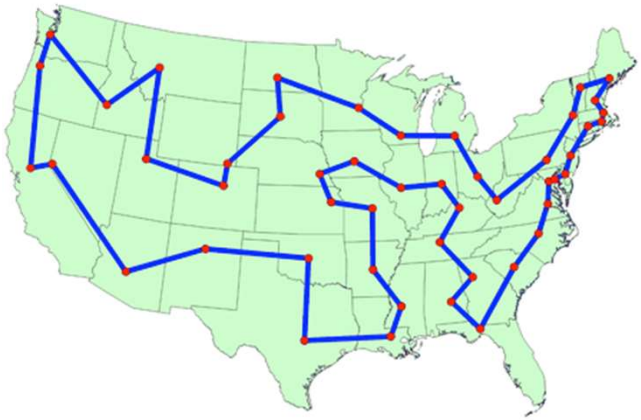
Numb. Item	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Weight/Size	85	26	48	21	22	95	43	45	55	52



- **Knapsack with 10 ”items”, capacity 101**
- **10 ”items” (e.g. projects, objects, ...) {1, 2, ..., 10}**
- **Trivial solution: empty backpack:**
  - [0000000000], Value: 0
- **Greedy solution, assign first the items with highest value:**
  - [0000010000], Value: 85
- **Better initial solutions?**

Practical Example: Delivery Company

- Consider a transportation/delivery company with one single truck
- You have a set of pickup/delivery jobs around the country/city to do in each day/month
- What sequence of trips should you choose to finish the jobs as early as possible?
- **Operational Decision**
  - Optimization problem
  - May be modelled as: **Travelling Salesman Problem (TSP**) or more generally, for several trucks as a **Vehicle Routing Problem (VRP)**



COP Example: Travelling Salesman

- **TSP: Possible Formulation with binary variables**

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \quad \text{subject to}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad ,i = 1,\ldots,n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad ,j = 1,\ldots,n$$

$$e_i \leq a_i \leq l_i \qquad ,i = 1,\ldots,n$$

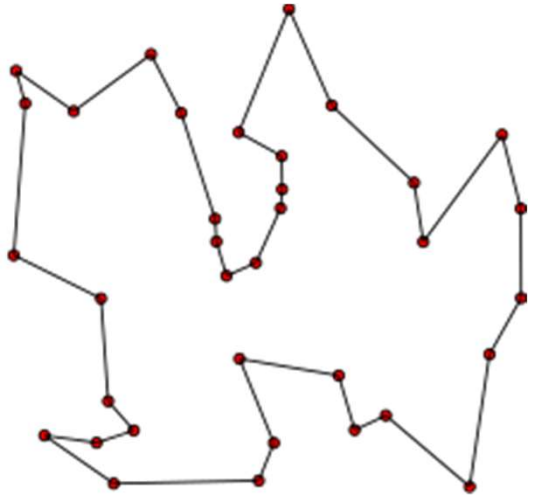
$$a_j \geq x_{ij}(a_i + c_{ij}) \quad ,i = 1,\ldots,n$$

$$,j = 1,\ldots,n$$

$$x_{i,j} = \begin{cases} 1 & \text{If city j follows right after city i} \\ 0 & \text{otherwise} \end{cases}$$

a_i Arrival time at city i

$c_{i,j}$ Cost goinf from i to j



COP Example: Travelling Salesman

Example TSP with 7 cities

Random solution:

- Random initial solution, example:
- [1 2 7 3 4 5 6 1] (Cost: 184)

Trivial solution:

- Visit cities in order
- [1 2 3 4 5 6 7 1] (Cost: 288)

Greedy construction:

- Choose to go to the closest, still unvisited city!
- [1 3 5 7 6 4 2 1] (Cost: 160)

Better initial solutions?

End / Start	1	2	3	4	5	6	7
1	0	18	**17**	23	23	23	23
2	**2**	0	88	23	8	17	32
3	17	33	0	23	**7**	43	23
4	33	**73**	4	0	9	23	19
5	9	65	6	65	0	54	**23**
6	25	99	2	**15**	23	0	13
7	83	40	23	43	77	**23**	0

Cost matrix for travelling start-> end cities

COP - Combinatorial Optimization Problem

- In a formal problem we usually find:
  - Data (parameters)
  - Decision variables
  - Constraints
- The problem is typically to find values for the variables that optimize some objective function subject to the constraints
  - Optimizing over some discrete structure gives a Combinatorial Optimization Problem

$$\max_{x \in \mathcal{F} \subseteq \mathcal{S}} f(x)$$

Optimal Solution:

$$i^* \in S : f(i^*) \leq f(i), \forall i \in S$$

Solution: often given by the values of the decision variables (x_1,…, x_n)

Where

- $\boldsymbol{x}$ is a vector of decision variables
- f($\boldsymbol{x}$) $\boldsymbol{S \rightarrow \mathcal{R}}$ is the objective function
- $\boldsymbol{S}$ is the solution space (often a subset in large state space (feasible + unfeasible)
- $\boldsymbol{\mathcal{F}}$ is the set of feasible solutions

Solving Optimization Problems

Exact and Heuristic methods for finding solutions to abstract problems:

(Exact) Algorithms

- Set of instructions that will result in the solution to a problem when followed correctly and is assumed to give the optimal solution to an optimization problem
- Typically very hard to implement for practical, large, real-world problems
- Very large running times, impossible in practice for large problems
- Sometimes link between the real-world problem and the formal problem is weak

Heuristic Algorithms

- Heuristic algorithms do not guarantee to find the optimal solution
- Heuristic algorithms do not have a bound on quality, that is, they can return a solution that is arbitrarily bad compared to the optimal solution
- However, in practice, heuristic algorithms (heuristics for short) have proven very successful in solving large real-world problems

Combinatorial Explosion

Number of possible solutions for different problems:

- Set Covering Problem:
  - Number of centers: *n* -> *N*umber of solutions: 2^n
- Traveling Salesman Problem:
  - A salesman must travel between *n* cities, visiting each once. The salesman can visit the cities in any order
  - Number of centers: *n* -> *N*umber of solutions: $\frac{1}{2}*(n-1)!$
- Number of solutions for hard optimization problems increase very quickly with problem/variables size

n	n^2	n^3	2^n	1/2(n-1)!
10	100	1000	1024	181440
100	10000	1000000	1.27E+30	4.7E+155
1000	1000000	1E+09	1.1E+301	#NUM!

Solving Optimization Problems

- **Exact methods**
  - Explicit enumeration
  - Implicit enumeration
    - Divide problem into simpler problems
    - Solve the simpler problems exactly
- **Trivial solutions**
- **Inspection of the problem instance**
- **Constructive methods**
  - Gradual construction with a greedy heuristic
  - Constraint Programming
- **Solve a simpler problem**
  - Remove/modify constraints
  - Modify the objective function
- **Starting with a initial solution and improving it**

Local Search

- Given a Solution: How to improve it?
- Small modification of a given solution gives a ”neighbour solution”
- A certain set of operations on a solution gives a set of neighbour solutions, a neighbourhood
- Evaluations of neighbours:
  - Objective function value?
  - Feasibility?

Example (Finding Better Solutions): Knapsack

Item Number	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Weight/Size	85	26	48	21	22	95	43	45	55	52
Solution	0	0	1	0	1	0	0	0	0	0

- **Given a solution 0010100000, with value 73**
- **Natural operator: ”Flip” a bit, i.e.**
  - If the item is in the knapsack, take it out
  - If the item is not in the knapsack, include it
- **Some Neighbours:**
  - 0110100000 value 105
  - 1010100000 value 152, not feasible
  - 0010000000 value 47

Search space is the set of solutions

Feasibility is with respect to the *constraint set*

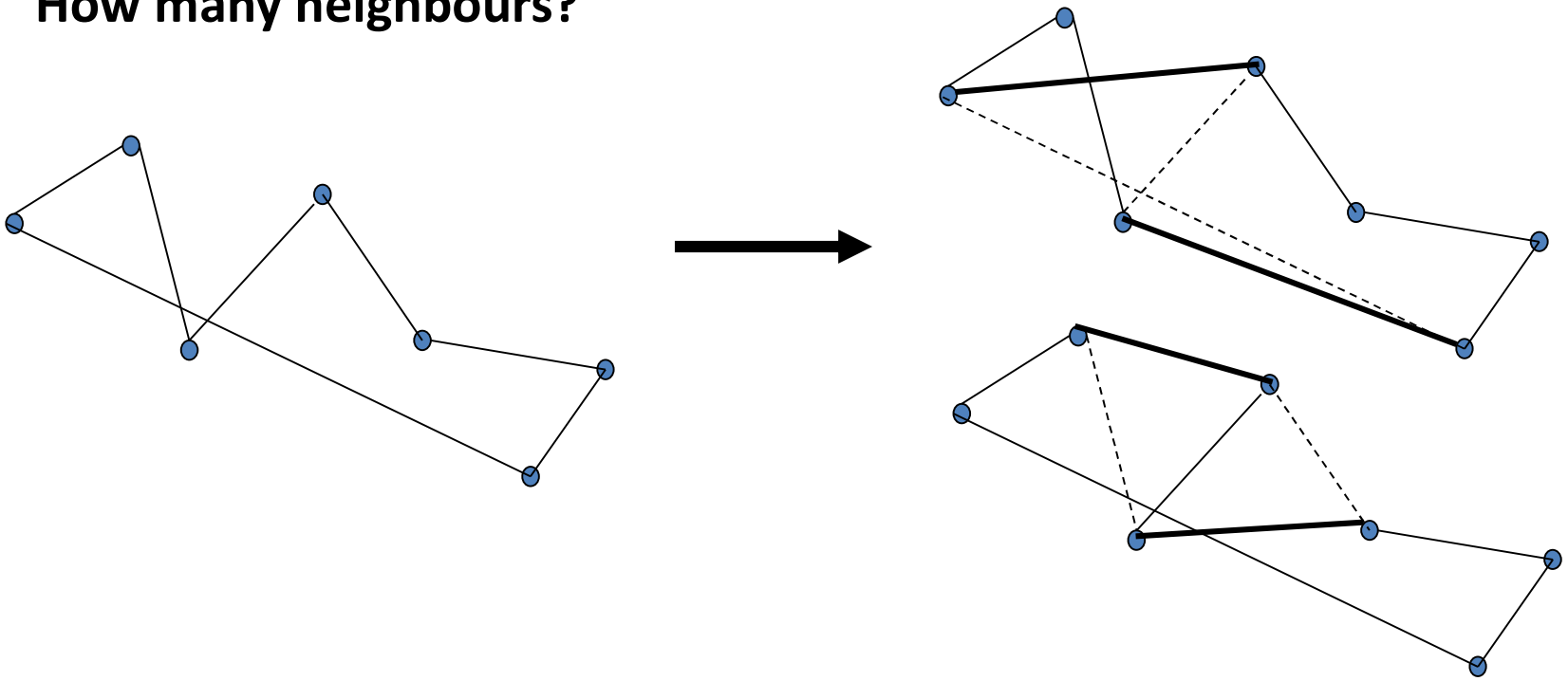
$$\sum_{i=1}^{n} a_i x_i \leq b$$

Evaluation is with respect to the *objective function*

$$\max \sum_{i=1}^{n} c_i x_i$$

Example: (Finding Better Solutions): Travelling Salesman

- **Operator: 2-opt**
- **How many neighbours?**



Definitions: Neighbour/Neighbourhood

- A neighbourhood function is a mapping from a solution to the set of possible solutions, reached by a move:
  - $N : S \rightarrow 2^S$
- For a given solution $s \in S$, *N* defines a neighbourhood of solutions, $t \in N(s)$, that in some sense is ”near” to s
- $N(s) \subseteq S$ is then a ”neighbour” of s
- Neighbourhoods are most often defined by a given operation on a solution
- Often simple operations:
  - Remove an element of the solution
  - Add an element to the solution
  - Interchange two or more elements of a solution

Global and Local Optima

- Assume we want to solve $\max_{x \in \mathcal{F} \subseteq \mathcal{S}} f(x)$
- Let $\boldsymbol{x}$ be our current solution in a local search
- If f($\boldsymbol{x}$) ≥ f($\boldsymbol{y}$) for all $\boldsymbol{y}$ in $\boldsymbol{\mathcal{F}}$, then we say that $\boldsymbol{x}$ is a global optimum (of f)
- Further assume that $\boldsymbol{\mathcal{N}}$ is a neighbourhood operator, so that $\boldsymbol{\mathcal{N}}(\boldsymbol{x})$ is the set of neighbours of $\boldsymbol{x}$
- If f($\boldsymbol{x}$) ≥ f($\boldsymbol{y}$) for all $\boldsymbol{y}$ in $\boldsymbol{\mathcal{N}}(\boldsymbol{x})$, then we say that $\boldsymbol{x}$ is a local optimum (of f, with respect to the neighbourhood operator $\boldsymbol{\mathcal{N}}$)
- All **global optima** are also **local optima** (with respect to any neighbourhood)

Local Search/Neighbourhood Search (1)

- Start with an *initial solution*
- Iteratively search in the neighbourhood for better solutions
- Sequence of solutions $s_{k+1} \in N_\sigma(s_k), k = 0, \ldots$
- Define a Strategy for which solution, in the neighbourhood, will be accepted as the next solution
- Define a Stopping Criteria
- Define what happens when the neighbourhood does not contain a better solution (local optimum):
  - If a solution *x* is ”better” than all solutions in its neighbourhood, *N(x)*, we say that *x* is a local optimum
  - Local optimality is defined relative to a particular neighbourhood
  - The way that the neighbours are defined is very important for the state space exploration

Local Search/Neighbourhood Search (2)

- Heuristic method
- Iterative method
- Small changes to a given solution
- Alternative search strategies:
  - Accept first improving solution (”First Accept”)
  - Search the full neighbourhood and go to the best improving solution
    - ”Steepest Descent”
    - ”Hill Climbing”
    - ”Iterative Improvement”
- Strategies with randomization
  - Random neighborhood search (”Random Walk”)
  - ”Random Descent”
- Other strategies?

Local Search/Neighbourhood Search (3)

Defining a local search solving method, needs the following:

- A Combinatorial Optimization Problem (COP)
- A starting solution (e.g. random, constructive)
- A defined search neighbourhood (neighbouring solutions)
- Definition of possible moves: e.g. changing a variable from 0 → 1 or 1 → 0, going from one solution to a neighbouring solution
- A solution evaluation function and, possibly, a move evaluation function – an evaluation of the possible changes
- A neighbourhood evaluation strategy
- A move selection strategy
- A stopping criterion: e.g. a local optimum, maximum time, maximum iterations

Local Search 1 : Best Accept

```
1: input: starting solution, s_0
2: input: neighborhood operator, N
3: input: evaluation function, f
4: current ⇐ s_0
5: done ⇐ false
6: while done = false do
7:    best_neighbor ⇐ current
8:    for each s ∈ N(current) do
9:       if f(s) < f(best_neighbor) then
10:         best_neighbor ⇐ s
11:      end if
12:   end for
13:   if current = best_neighbor then
14:      done ⇐ true
15:   else
16:      current ⇐ best_neighbor
17:   end if
18: end while
```

Local Search 2 : First Accept

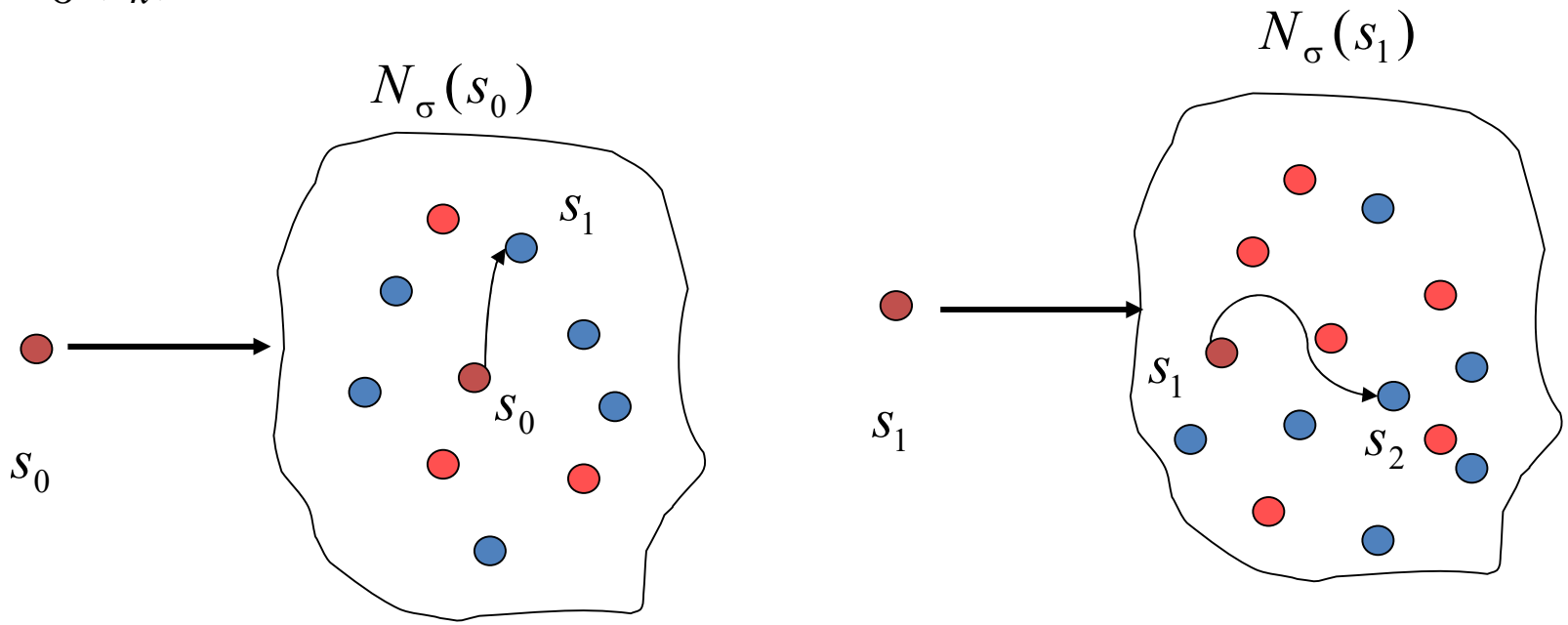
```
1: input: starting solution, s_0
2: input: neighborhood operator, N
3: input: evaluation function, f
4: current ⇐ s_0
5: done ⇐ false
6: while done = false do
7:    best_neighbor ⇐ current
8:    for each s ∈ N(current) do
9:       if f(s) < f(best_neighbor) then
10:         best_neighbor ⇐ s
11:         exit the for-loop
12:      end if
13:   end for
14:   if current = best_neighbor then
15:      done ⇐ true
16:   else
17:      current ⇐ best_neighbor
18:   end if
19: end while
```

”Best Accept” and ”First Accept”

- ”Best Accept” and ”First Accept” stops in a local optimum
- ”Best Accept”: Selects the best neighbour
- ”First Accept”: Selects the first better neighbour found
- Local Search can be regarded as a traversal in a directed graph (the *neighbourhood graph),* where the nodes are the members of *S*

Local Search: Traversal of the Neighbourhood Graph

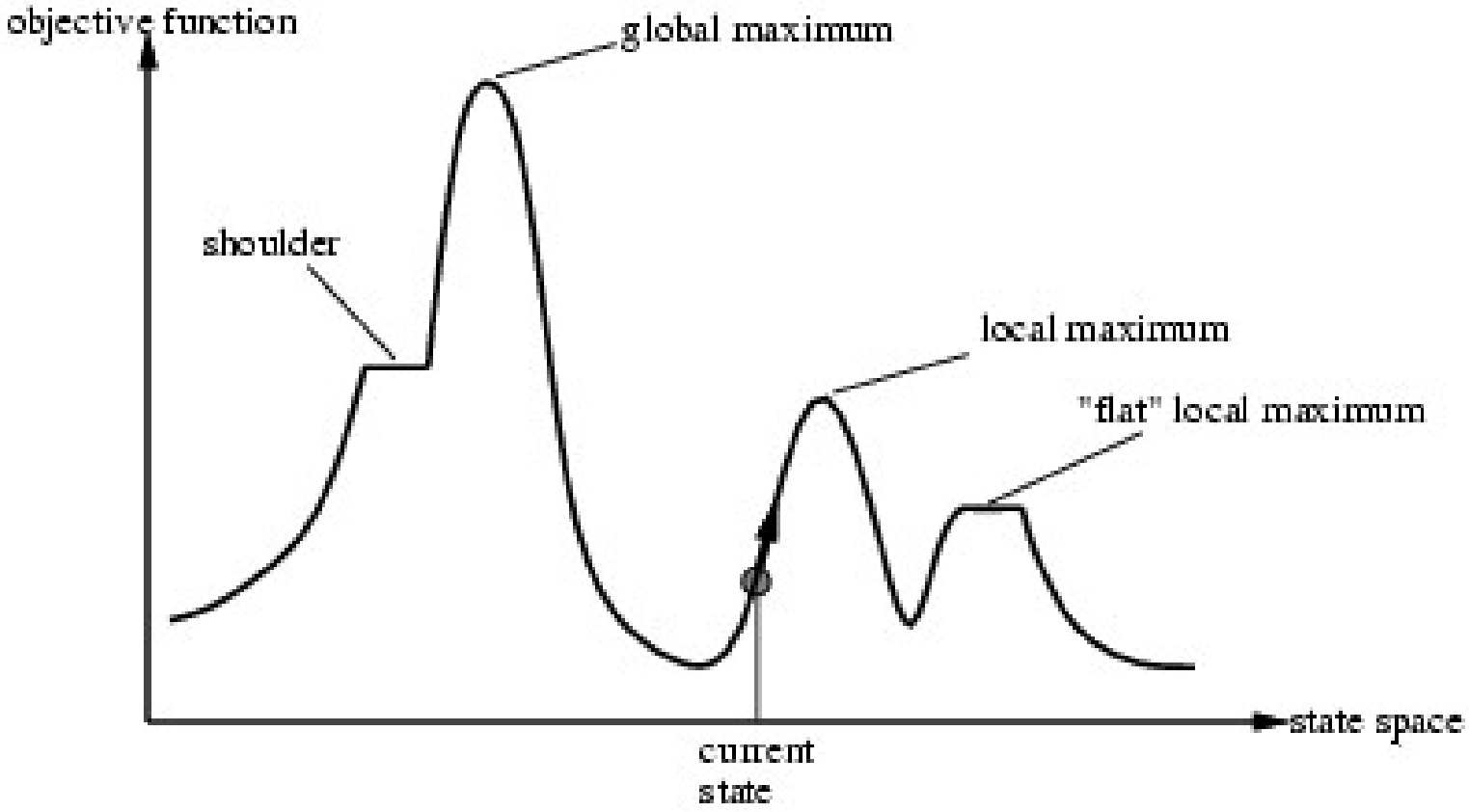
$$s_{k+1} \in N_\sigma(s_k), k = 0, \ldots$$



A *move* is the process of selecting a given solution in the neighborhood of the current solution to be the current solution for the next iteration

Local and Global Optima

Example for Maximization Problem

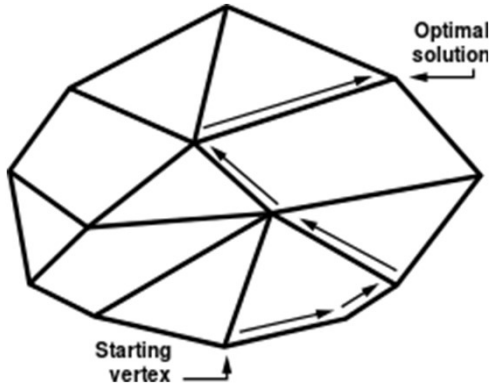


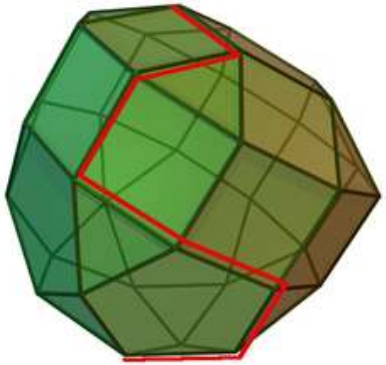
Example of Local Search: Simplex

- Simplex algorithm for Linear Programming
  - Simplex Phase I gives an initial (feasible) solution
  - Phase II gives iterative improvement towards the optimal solution (if it exists)
- The *Neighbourhood* is defined by the simplex polytope
- The *Strategy* is ”Iterative Improvement”
- The moves are determined by pivoting rules
- The neighbourhood is exact. This means that the Simplex algorithm finds the global optimum (if it exists)

maximize $\mathbf{c}^{\mathrm{T}}\mathbf{x}$

subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$



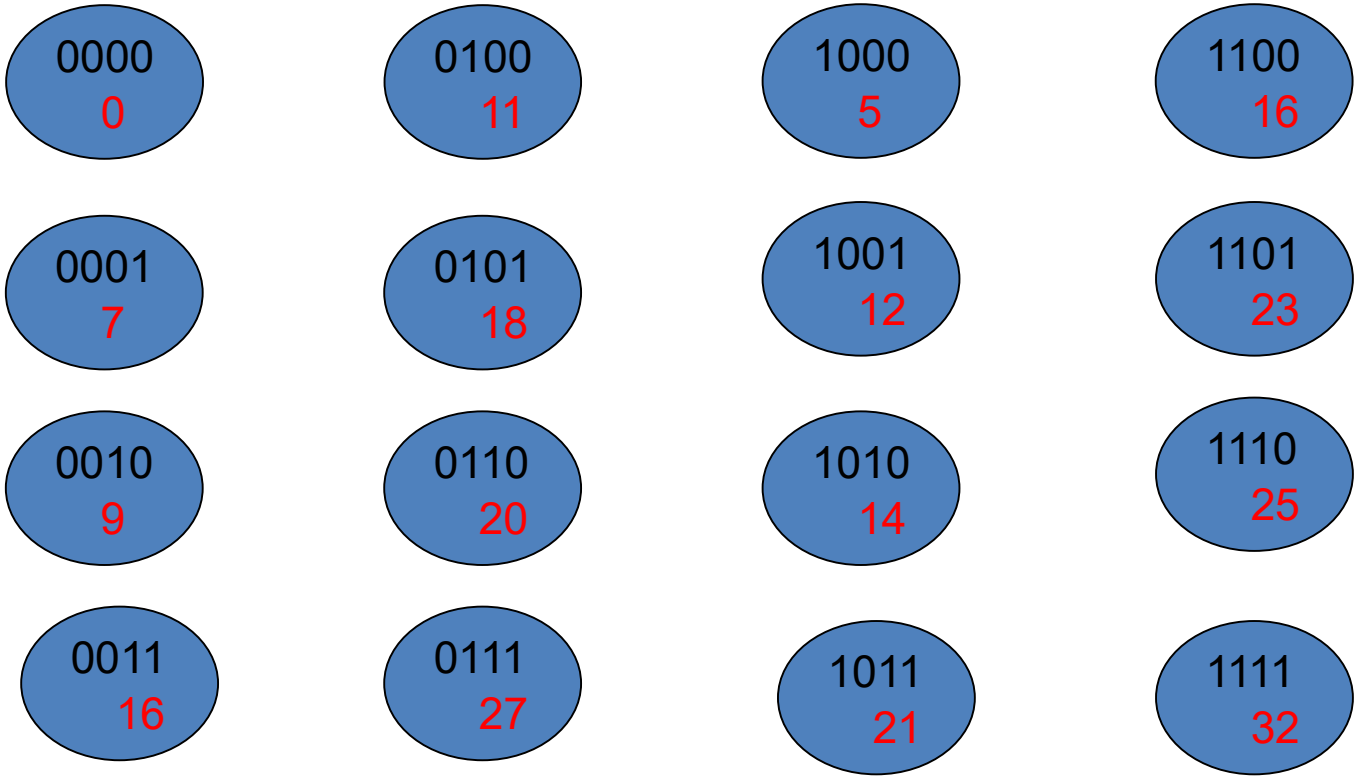


Example Knapsack Search Space

Knapsack with four items **Solution $\Rightarrow$ xxxx**

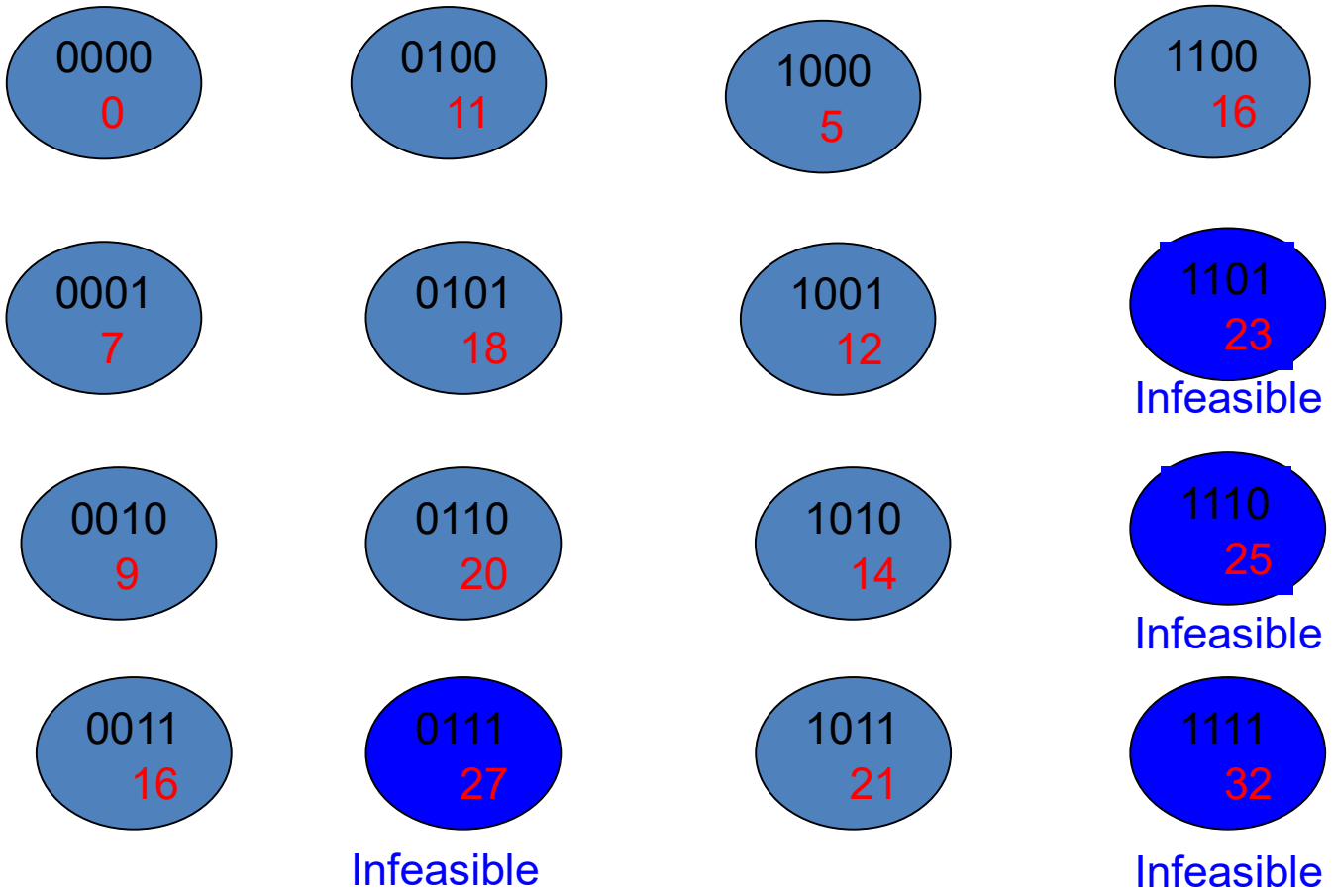
Search space is the set of solutions (feasible and infeasible)

Objective Function Value = Total value of selected items

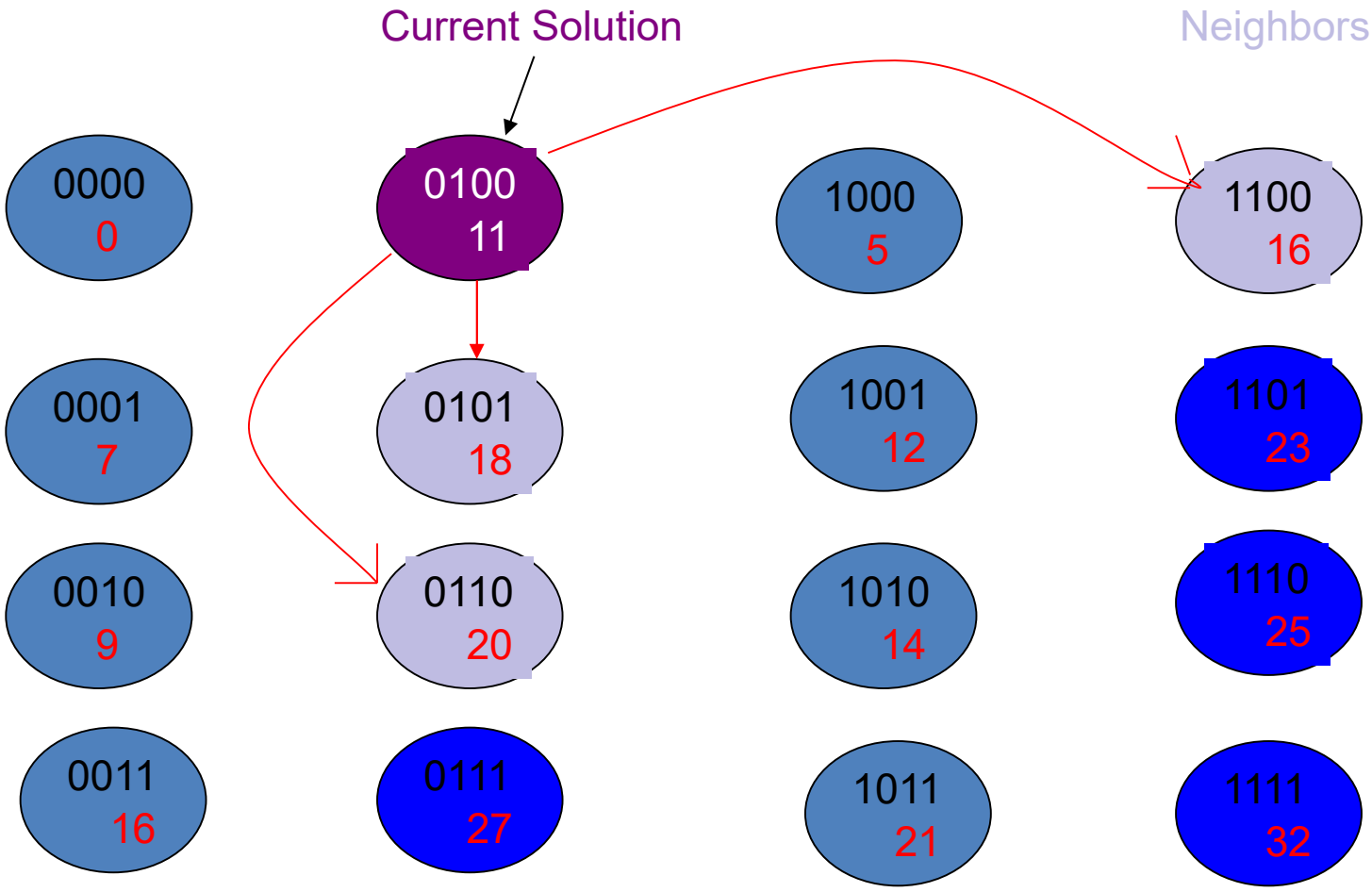


Feasible/Infeasible Space/Solutions

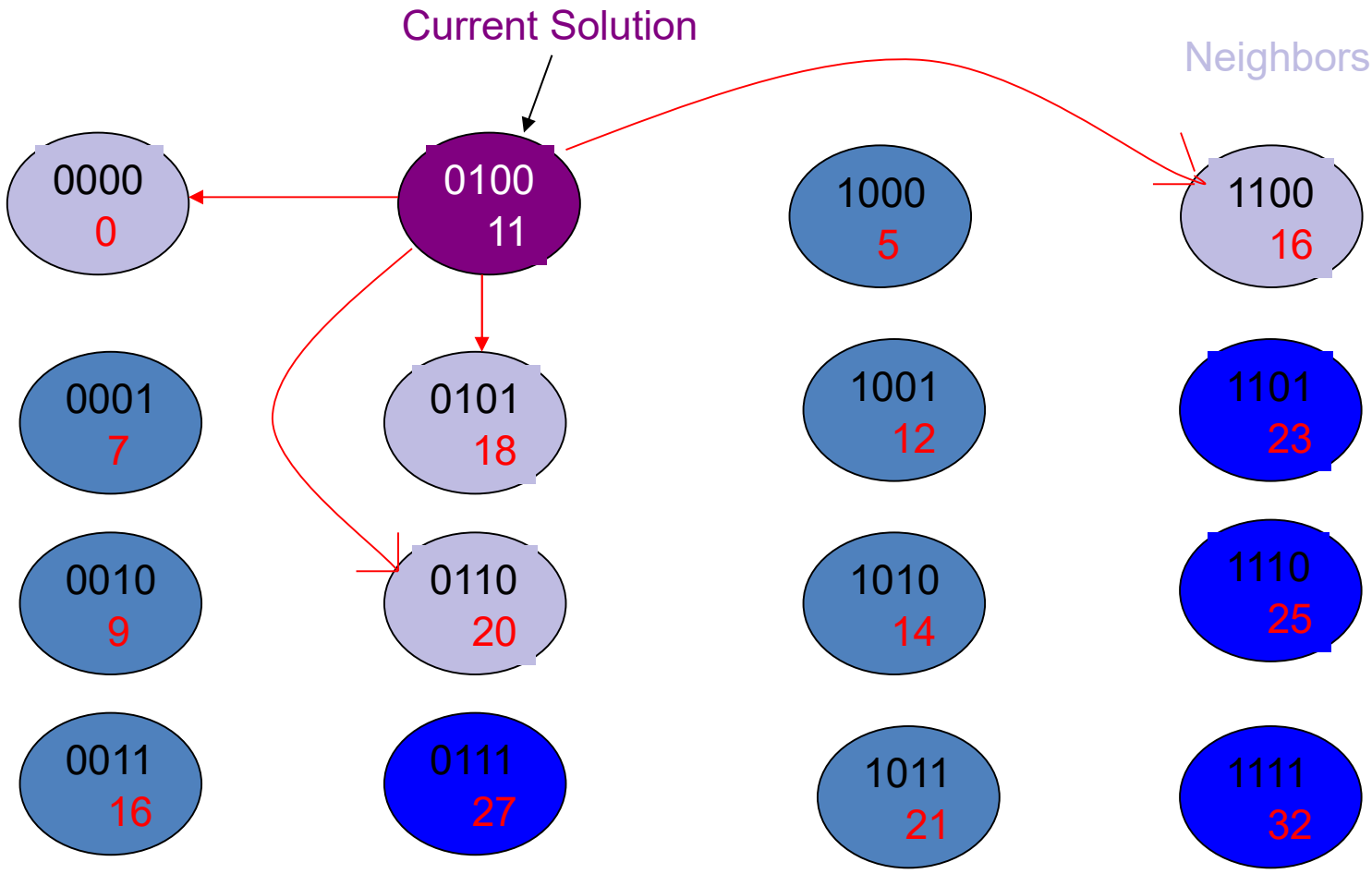
Feasible solutions: $\sum_{i=1}^{n} a_i x_i \leq b$



Example of a “Add” Neighbourhood



Example of a “Flip” Neighbourhood



Advantages of Local Search

- For many problems, it is quite easy to design a local search (i.e., LS can be applied to almost any problem)
- The idea of improving a solution by making small changes is very easy to understand
- The use of neighbourhoods sometimes makes the optimal solution seem ”close”, e.g.:
  - A knapsack has *n* items
  - The search space has 2^n members
  - From achieving *any* solution, no more than *n* flips in the solution are required to reach an optimal solution!
  - E.g. for a problem with 100 binary variables the search space size will be 2^{100} = 1.27E+30 but the solution is in reach on 100 correctly decided flips

Disadvantages of Local Search

- The search stops when no improvement can be found
- Restarting the search might help, but is often not very effective in itself
- Some neighbourhoods can become very large (time consuming to examine all the neighbours)

Main Challenge in Local Search

How can we avoid the search stopping in a local optimum?

Heuristics and Meta-Heuristics

Heuristics (from greek *heuriskein,* meaning "to find")

- (…)A heuristic is a technique designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution (…). [Wikipedia]
- Heuristics are intended to gain computational performance or conceptual simplicity, potentially at the cost of accuracy or precision

Meta-Heuristics

- Concept introduced by Glover (1986)
- Generic heuristic solution approaches designed to control and guide specific problem-oriented heuristics
- Often inspired from analogies with natural processes
- Rapid development over the last 30 years

Metaheuristics (1)

- Concept introduced by Glover (1986)
- Generic heuristic solution approaches designed to control and guide specific problem-oriented heuristics
- Often inspired from analogies with natural processes
- Rapid development over the last 15 years
- **Different definitions:**
  - "Iterative generating process, controlling an underlying heuristic, by combining (in an intelligent way) various strategies to explore and exploit search spaces (and learning strategies) to find near-optimal solutions in an efficient way"
  - "A master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality"
  - "Procedure that has the ability to escape local optimality"

Metaheuristics (2)

- **Glover and Hochenberger (2003) writes:**

  "Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of solution space. "

  "Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighbourhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes."

A History of Success…

- Metaheuristics have been applied quite successfully to a variety of difficult combinatorial problems encountered in numerous application settings
- Because of that, they have become extremely popular and are often seen as a panacea

… and of Failures

- There have also been many less-than-successful applications of metaheuristics
- The moral being that one should look at alternatives first (exact algorithms, problem specific approximation algorithms or heuristics)
- If all else is unsatisfactory, metaheuristics can often perform very well

Some well-known Metaheuristics

- Simulated Annealing (SA)
- Tabu Search (TS)
- Genetic Algorithms (GA)

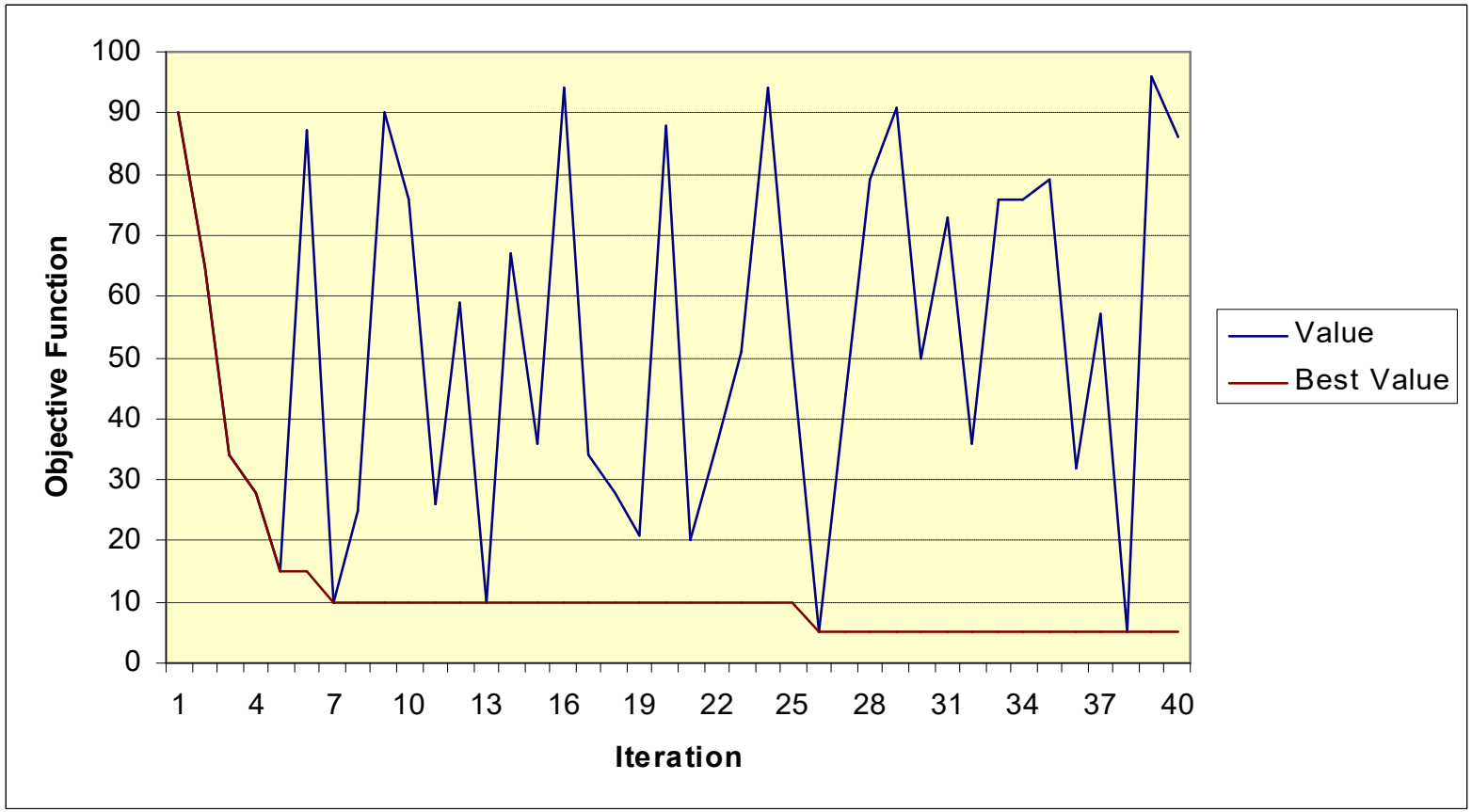
Some other Metaheuristics

- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)
- Scatter Search (SS)
- Adaptive Memory Procedures (AMP)
- Variable Neighbourhood Search (VNS)
- Iterative Local Search (ILS)
- Guided Local Search (GLS)
- Threshold Acceptance methods (TA)
- Greedy Randomized Adaptive Search Procedure (GRASP)
- Memetic Algorithms (MA)
- And several others…
  - Harmony Method, The Great Deluge Method, Shuffled Leaping-Frog Algorithm, Squeaky Wheel Optimization, …

Metaheuristics Classification

- **x/y/z Classification**
  - x = A (adaptive memory) or M (memoryless)
  - y = N (systematic neighbourhood search) or S (random sampling)
  - z = 1 (one current solution) or P (population of solutions)
- **Some Classifications**
  - Scatter Search (M/S/1)
  - Tabu search (A/N/1)
  - Genetic Algorithms (M/S/P)
  - Scatter Search (M/N/P)

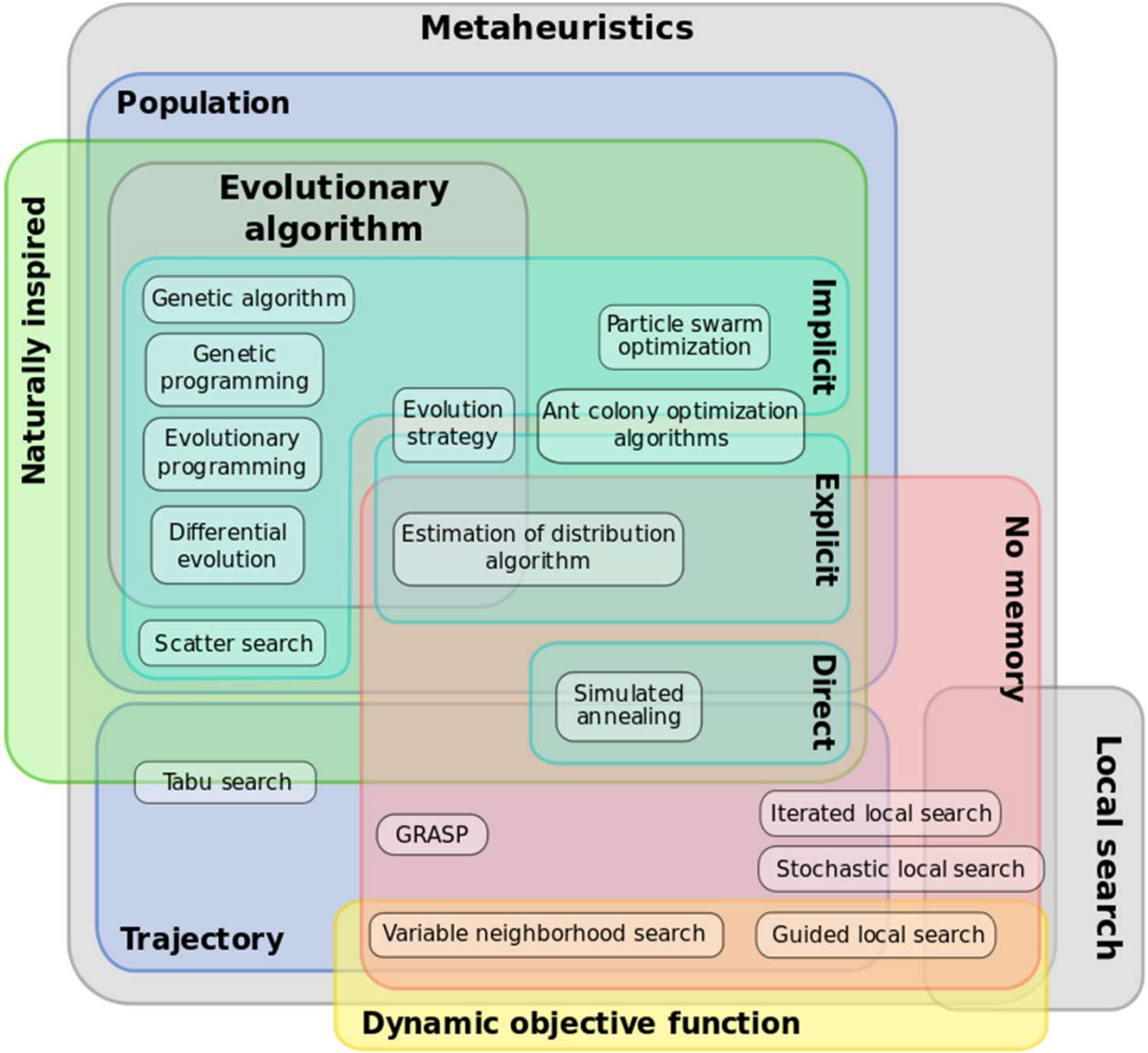
Typical Search Trajectory



Metaheuristics and Local Search

- In Local Search, we iteratively improve a solution by making small changes until we cannot make further improvements
- Metaheuristics can be used to guide a Local Search, and to help it to escape a local optimum
- Several metaheuristics are based on Local Search, but the mechanisms to escape local optima vary widely

Optimization Algorithms and Meta-Heuristics



Summary and Conclusions

- **Optimization Problems**
- **Combinatorial Optimization Problems (COPs)**
- **Formulating COPs**
  - Examples: Knapsack Problem and Travelling Salesman Problem
- **Local and Global Optima**
- **Local Search**
- **Best Accept and First Accept**
- **Evaluation, Neighbourhoods, Stopping Criteria**
- **Metaheuristics**
  - Definition and Classification
  - Escaping local optima

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence
Lecture 3a: Optimization and Local Search

(based on Løkketangen, 2019)

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence