Simulation

Lecture H1

Heuristic Methods: Local Search

Author : Saeed Bastani
Saeed.bastani@gmail.com
Teacher: Mohammadhassan Safavi
Mohammadhassan.safavi@eit.lth.se

May 2018

Thanks to Prof. Arne Løkketangen at Molde University College, Norway, for making and sharing his presentation slides
(http://kursinfo.himolde.no/forskningsgrupper/)

Outline

- ✓ Introduction
- ✓ Why do we need heuristics?
- ✓ Local Search
- ✓ Meta-Heuristics

How can we solve problems?

- It can sometimes be advantageous to distinguish between three groups of methods for finding solutions to our abstract problems

- (Exact) Algorithms
- Approximation Algorithms
- Heuristic Algorithms

(Exact) Algorithms

- An algorithm is sometimes described as a set of instructions that will result in the solution to a problem when followed correctly
- Unless otherwise stated, an algorithm is assumed to give the optimal solution to an optimization problem
  - That is, not just a **good** solution, but the **best** solution

Approximation Algorithms

- Approximation algorithms (as opposed to exact algorithms) do not guarantee to find the optimal solution
- However, there is a bound on the quality
  - E.g., for a maximization problem, the algorithm can guarantee to find a solution whose value is at least half that of the optimal value
- We will not see many approximation algorithms here, but mention them as a contrast to heuristic algorithms

Heuristic Algorithms

- Heuristic algorithms do not guarantee to find the optimal solution, however:
  - Heuristic algorithms do not even necessarily have a bound on how bad they can perform
    - 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 successful
  - Most of the following lectures of the course will focus on this type of heuristic solution method

What is a heuristic?

- From greek *heuriskein* (meaning ”to find”)
- Wikipedia says:
  - (…)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 (…).
  - Heuristics are intended to gain computational performance or conceptual simplicity, potentially at the cost of accuracy or precision.

Why don't we always use exact methods?

- If a heuristic does not guarantee a good solution, why not use an (exact) algorithm that does?
- The running time of the algorithm
  - For reasons explained soon, the running time of an algorithm may render it useless on the problem you want to solve
- The link between the real-world problem and the formal problem is weak
  - Sometimes you cannot properly formulate a COP/IP that captures all aspects of the real-world problem
  - If the problem you solve is not the right problem, it might be just as useful to have one (or more) heuristic solutions, rather than the optimal solution of the formal problem

P vs **NP** (1)

- Computational complexity is sometimes used to motivate the use of heuristics
- Formal **decision** problems are divided into many classes, but two such classes are
  - **P** (**P**olynomial)
    - Includes problems for which there exist algorithms that have a running time that is a polynomial function of the size of the instance
  - **NP** (**N**ondeterministic **P**olynomial)
    - Includes problems for which one can **verify** in polynomial time that a **given solution** is correct

P vs **NP** (2)

- We believe that **P** is different from **NP**, but nobody has proven this
  - $1 000 000 awaits those that can prove it
  - (note that all of **P** is contained in **NP**)
- Some optimization problems can be solved in polynomial time
  - If you have a graph with n nodes, the shortest path between any two nodes can be found after f(n) operations, where f(n) grows as quickly as n^2
  - If you want to distribute n jobs among n workers, and each combination of job and worker has a cost, the optimal assignment of jobs can be found in g(n) operations, where g(n) grows as quickly as n^3

P vs **NP** (3)

- However, for some optimization problems we know of no polynomial time algorithm
  - Unless **P**=**NP** there are none!
- Sometimes, finding the optimal solution reduces to examining **all** the possible solutions (i.e., the entire solution space)
  - Some algorithms do **implicit enumeration** of the solution space (but this sometimes reduces to examining all solutions)
- So, how many solutions must we examine in order to find the optimal solution?

What is a Combinatorial Optimization Problem (COP)? (1)

- 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

What is a Combinatorial Optimization Problem (COP)? (2)

- Can be expressed, very generally, as

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

- Where
  - $\boldsymbol{x}$ is a vector of decision variables
  - f($\boldsymbol{x}$) is the objective function
  - $\boldsymbol{S}$ is the solution space
  - $\boldsymbol{\mathcal{F}}$ is the set of feasible solutions

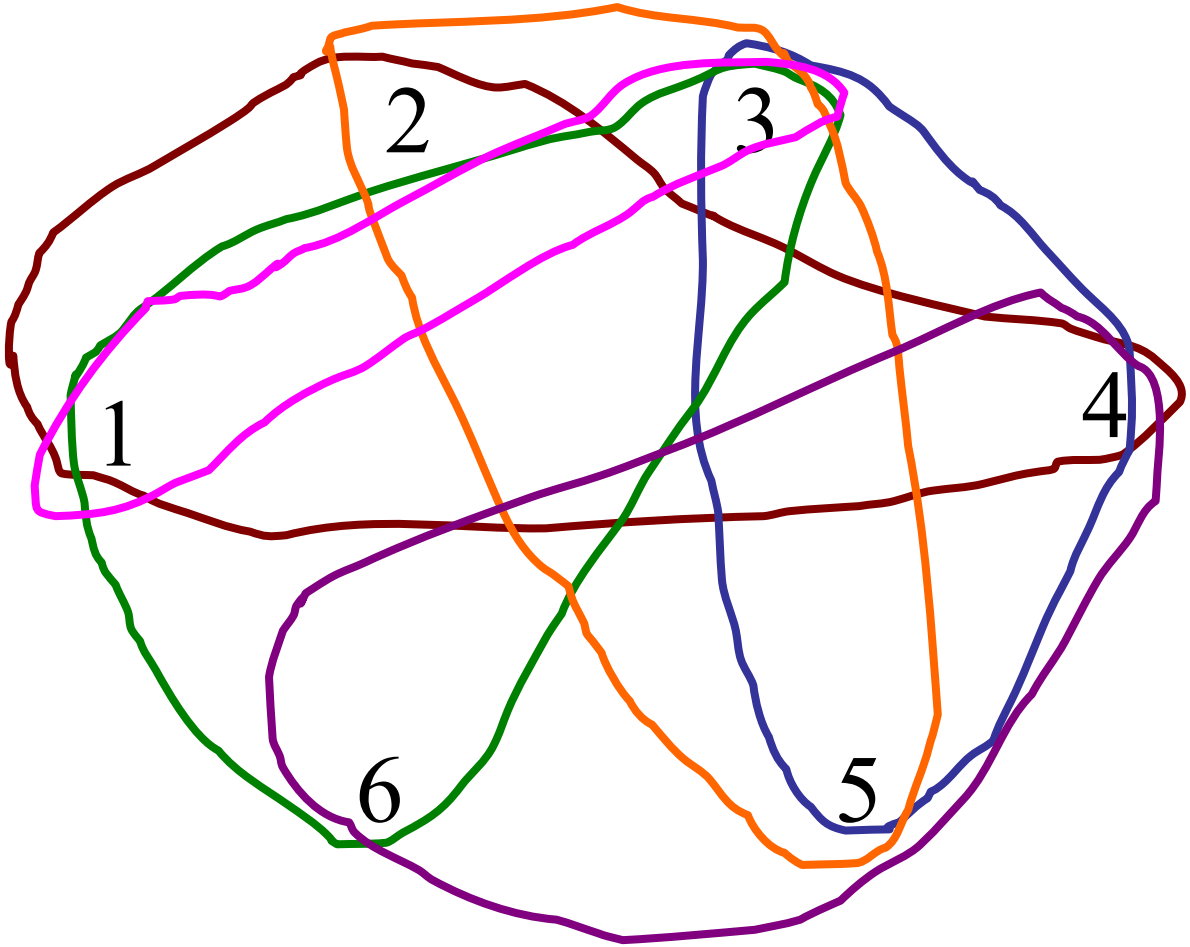
Example COP: Set Cover (1)

- **We are given:**
  - A finite set S = {1, …, n}
  - A collection of subsets of S: S_1, S_2, …, S_m
- **We are asked:**
  - Find a subset T of {1, …, m} such that $U_{j \in T} S_j = S$
  - Minimize |T|
- **Decision variant of the problem:**
  - we are additionally given a target size k, and
  - asked whether a T of size at most k will suffice
- **One instance of the set cover problem:**

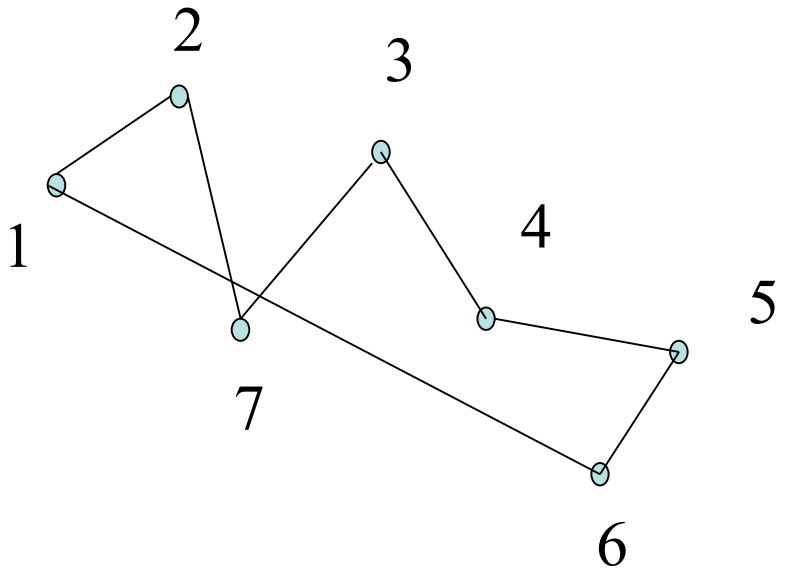
  S = {1, …, 6}, S_1 = {1,2,4}, S_2 = {3,4,5}, S_3 = {1,3,6}, S_4 = {2,3,5}, S_5 = {4,5,6}, S_6 = {1,3}

Example COP: Set Cover (2)

- S = {1, …, 6}, S_1 = {1,2,4}, S_2 = {3,4,5}, S_3 = {1,3,6}, S_4 = {2,3,5}, S_5 = {4,5,6}, S_6 = {1,3}



Example COP:
TSP – Travelling Salesman Problem



	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

Feasible Solution: 1 2 7 3 4 5 6 1 with value: 184

The Combinatorial Explosion (1)

- The number of possible solutions for different problems:
  - The Set Covering Problem:
    - The size of the S is *n*
    - The number of solutions: 2^n
  - The Traveling Salesman Problem:
    - A salesman must travel between n cities, visiting each once. The salesman can visit the cities in any order
    - The number of solutions: $\frac{1}{2}*(n-1)!$

The Combinatorial Explosion (2)

- The combinatorial explosion refers to the fact that some functions (such as those that result as the number of solutions for some hard optimization problems) increase **very** quickly!

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!

- How would you solve a Traveling Salesman Problem with 100 customers?

Polynomial vs Exponential time

Assume: computer speed 10^6 IPS and input size n = 10^6

Time complexity	Running time
n	1 sec.
n log n	20 sec.
n^2	12 days
2^n	40 quadrillion (10^{15}) years

A Small Note on the TSP

- Although the number of solutions of the TSP grows very quickly, surprisingly large instances has been solved to optimality
  - A TSP has been solved for 24978 cities in Sweden (the length was about 72500 kilometers)
  - A TSP for 33810 points on a circuit board was solved in 2005
    - It took 15.7 CPU-years!
- However, we also have heuristic methods that can quickly find solutions within 2-3% of optimality for problems with millions of cities!

How to find solutions?

- Exact methods
  - Explicit enumeration
  - Implicit enumeration
    - Divide problem into simpler problems
    - Solve the simpler problems exactly
- Trivial solutions
- Inspection of the problem instance
- Constructive method
  - Gradual costruction with a greedy heuristic
- Solve a simpler problem
  - Remove/modify constraints
  - Modify the objective function

Example: TSP

Trivial solution:

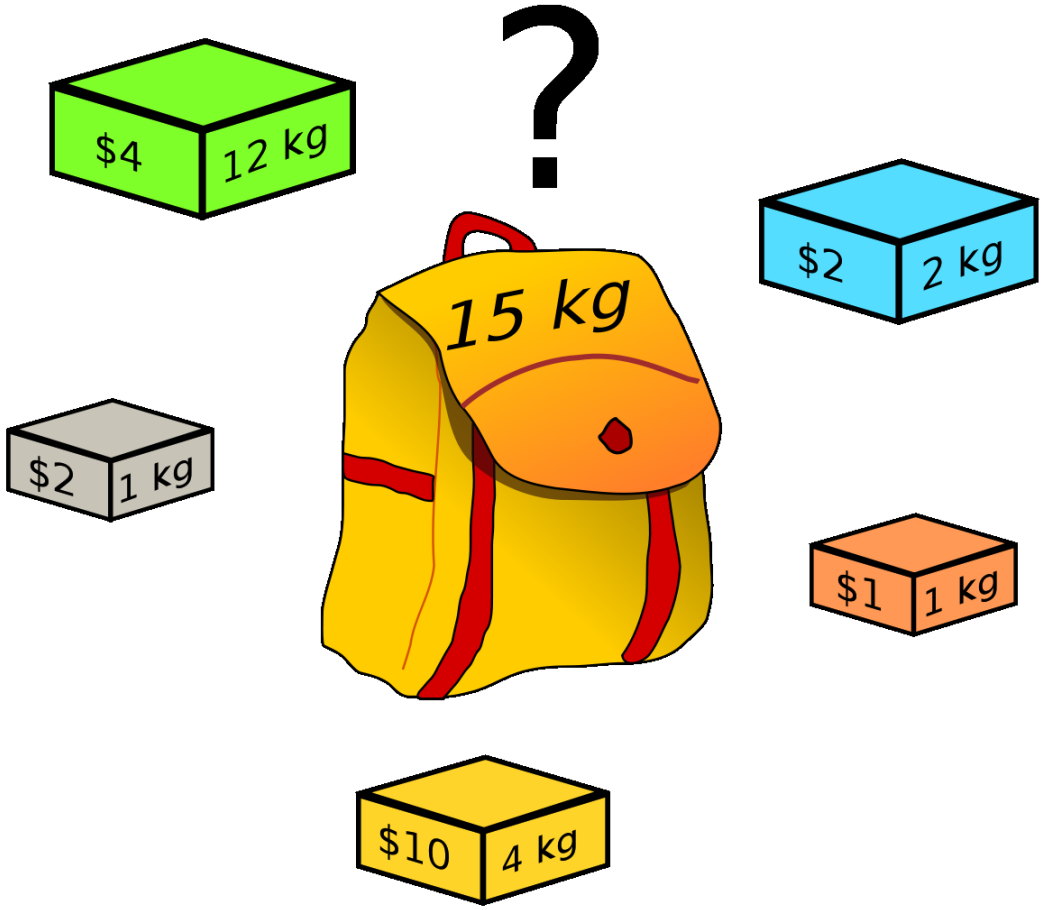
1 2 3 4 5 6 7 1 (288)

Greedy construction:

1 3 5 7 6 4 2 1 (160)

	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

COP Example: The Knapsack Problem (1)



COP Example: The Knapsack Problem (2)

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

$$x_i = \begin{cases} 1 & \text{If the item i is in the knapsack} \\ 0 & \text{otherwise} \end{cases}$$

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

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

Example: Knapsack Problem (3)

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

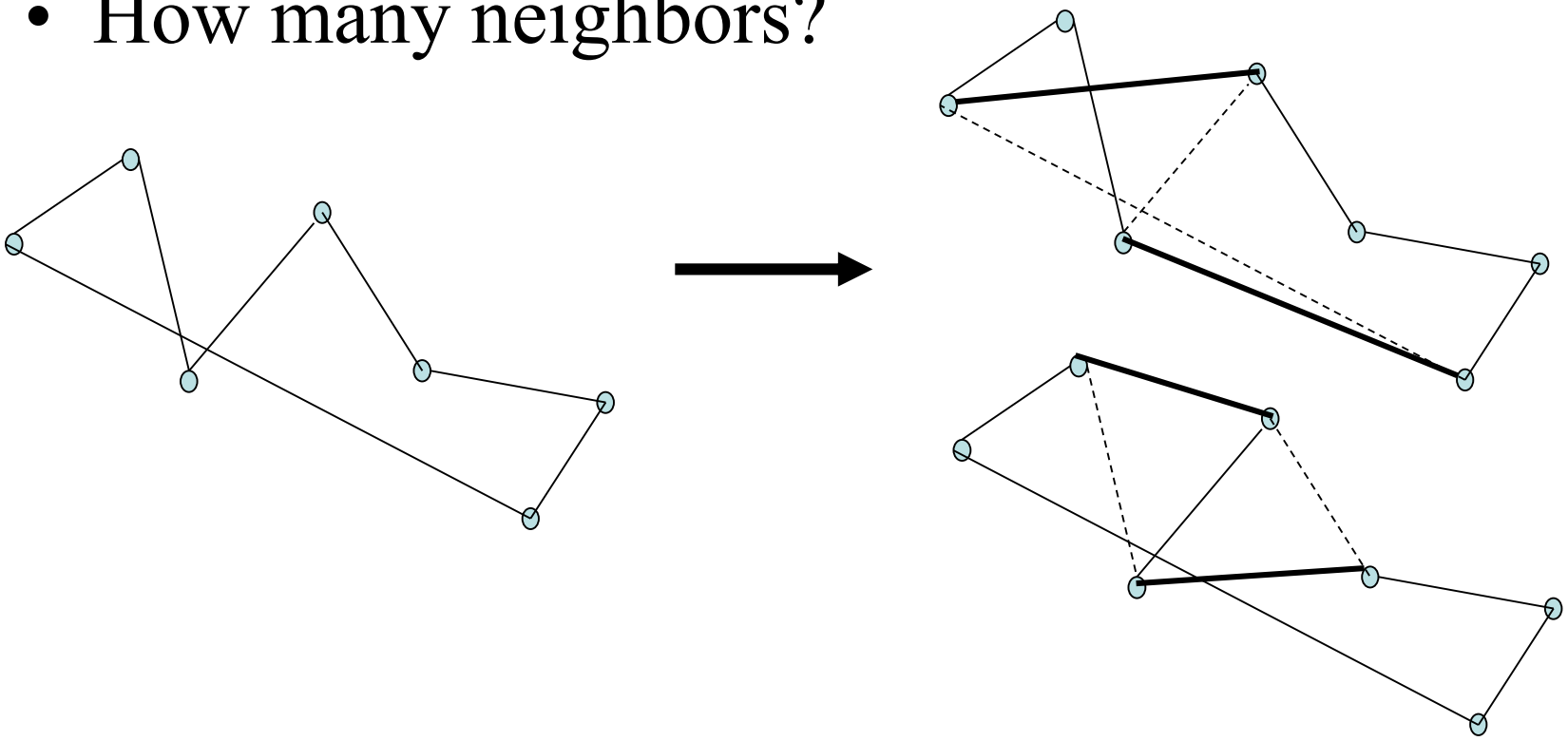
- Knapsack with capacity 101
- 10 ”items” (e.g. projects, ...) 1,...,10
- Trivial solution: empty knapsack, value 0
- Greedy solution, assign the items after value:
  - (0000010000), value 85
  - Better suggestions?

Given a Solution: How to Find a Better One

- Modification of a given solution gives a ”neighbor solution”
- A certain set of **operations** on a solution gives a set of neighbor solutions, a **neighborhood**
- Evaluations of neighbors
  - Objective function value
  - Feasibility ?

Example: TSP

- Operator: 2-opt
- How many neighbors?



Example: Knapsack Instance

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

- Given solution 0010100000 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 Neighbors:
  - 0110100000 value 105
  - 1010100000 value 152, not feasible
  - 0010000000 value 47

Definition: Neighborhood

- Let (S,f) be a COP-instance
- A neighborhood 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 neighborhood of solutions, $t \in N(s)$, that in some sense is ”near” to s
- $N(s) \subseteq S$ is then a ”neighbor” of s

Neighborhood Operator

- Neighborhoods are most often defined by a given operation on a solution
- Often simple operations
  - Remove an element
  - Add an element element
  - Interchange two or more elements of a solution
- Several neighborhoods – qualify with an operator

$$N_{\sigma}(s), \sigma \in \Sigma$$

Terminology: Optima (1)

- Assume we want to solve $\max_{x \in \mathcal{F} \subseteq \mathcal{S}} f(x)$
- Let $\boldsymbol{x}$ be our current (incumbent) 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 $\boldsymbol{\mathcal{F}}$.

Terminology: Optima (2)

- Further assume that $\mathcal{N}$ is a neighborhood operator, so that $\mathcal{N}(\boldsymbol{x})$ is the set of neighbors of $\boldsymbol{x}$

- If $f(\boldsymbol{x}) \geq f(\boldsymbol{y})$ for all $\boldsymbol{y}$ in $\mathcal{N}(\boldsymbol{x})$, then we say that $\boldsymbol{x}$ is a **local optimum** (of f, with respect to the neighborhood operator $\mathcal{N}$)

- Note that all **global optima** are also **local optima** (with respect to **any** neigborhood)

Local Search (LS)/ Neighborhood Search (1)

- Start with an *initial solution*
- Iteratively search in the neighborhood for better solutions
- Sequense of solutions $s_{k+1} \in N_\sigma(s_k), k = \cdots$
- Strategy for which solution in the neighborhood that will be accepted as the next solution
- Stopping Criteria
- What happens when the neighborhood does not contain a better solution?

Local Search / Neighborhood Search (2)

- We remember what a local optimum is:
  - If a solution x is ”better” than all the solutions in its neighborhood, $N(x)$, we say that x is a local optimum
  - We note that local optimality is defined relative to a particular neighborhood
- Let us denote by S_N the set of local optima
  - S_N is relative to N
- If S_N only contains global optima, we say that N is exact
  - Can we find examples of this?

Local Search / Neighborhood Search (3)

- Heuristic method
- Iterative method
- Small changes to a given solution
- Alternative search strategies:
  - Accept first improving solution **(”First Accept”)**
  - Search the full neighborhood and go to the best improving solution
    - ”Steepest Descent”
    - ”Hill Climbing”
    - ”Iterative Improvement”

    (These methods are called **”Best Accept”**)
- Strategies with randomization
  - Random neighborhood search (”Random Walk”)
  - ”Random Descent”
- Other strategies?

Local Search / Neighborhood Search (4)

In a local search need the following:

- a Combinatorial Optimization Problem (COP)
- a starting solution (e.g. random)
- a defined search neighborhood (neighboring solutions)
- a move (e.g. changing a variable from $0 \rightarrow 1$ or $1 \rightarrow 0$), going from one solution to a neighboring solution
- a move evaluation function – a rating of the possibilities
  - Often *myopic*
- a neighborhood evaluation strategy
- a move selection strategy
- a stopping criterion – e.g. a local optimum

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

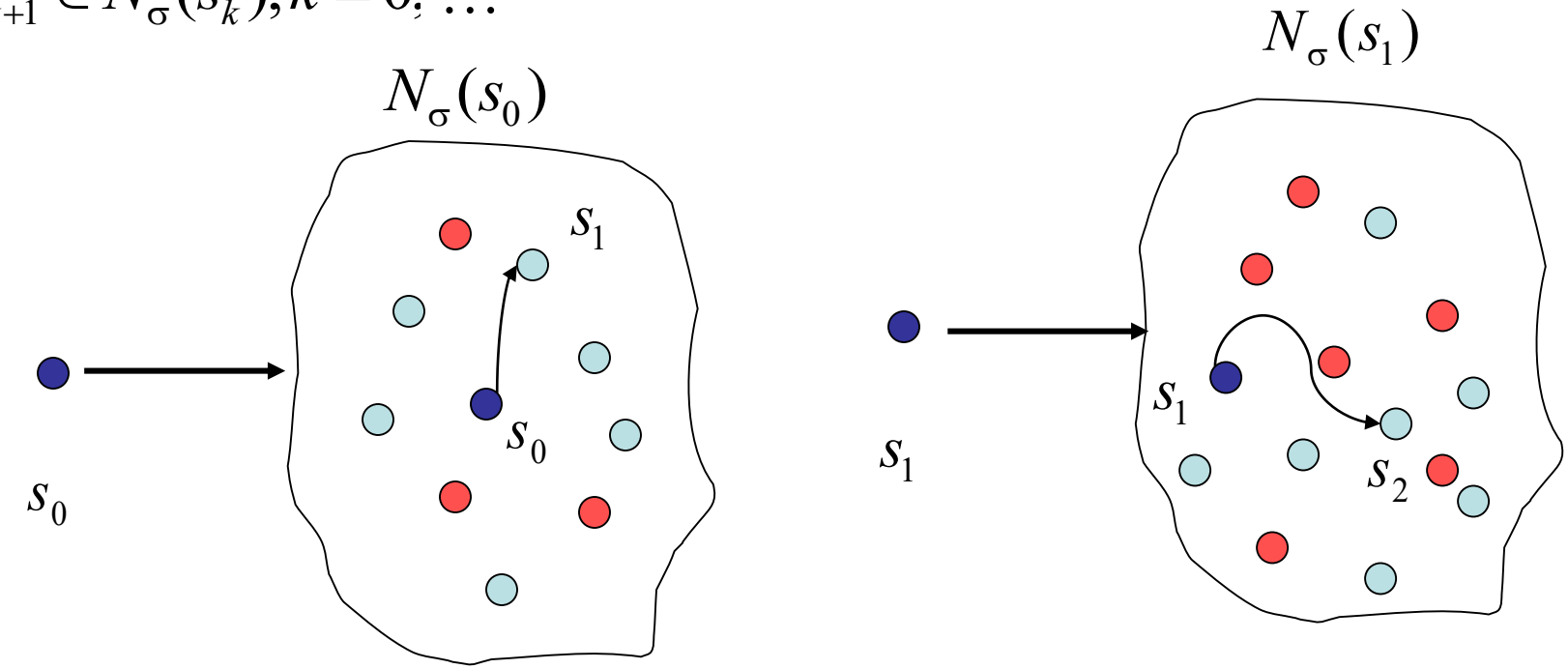
```
1: input: starting solution, s_0
2: input: neighborhood operator, N
3: input: evaluation function, f
4: current ⇐ s_0
5: done ⇐ false
6: while done = false do
7:    best_neighbor ⇐ current
8:    for each s ∈ N(current) do
9:       if f(s) < f(best_neighbor) then
10:         best_neighbor ⇐ s
11: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

Observations

- ”Best Accept” and ”First Accept” stops in a local optimum
- If the neighborhood *N* is exact, then the local search is an exact optimization algorithm
- Local Search can be regarded as a traversal in a directed graph (the *neighborhood graph),* where the nodes are the members of *S*, and *N* defines the topolopy (the nodes are marked with the solution value), and f defines the ”topography”

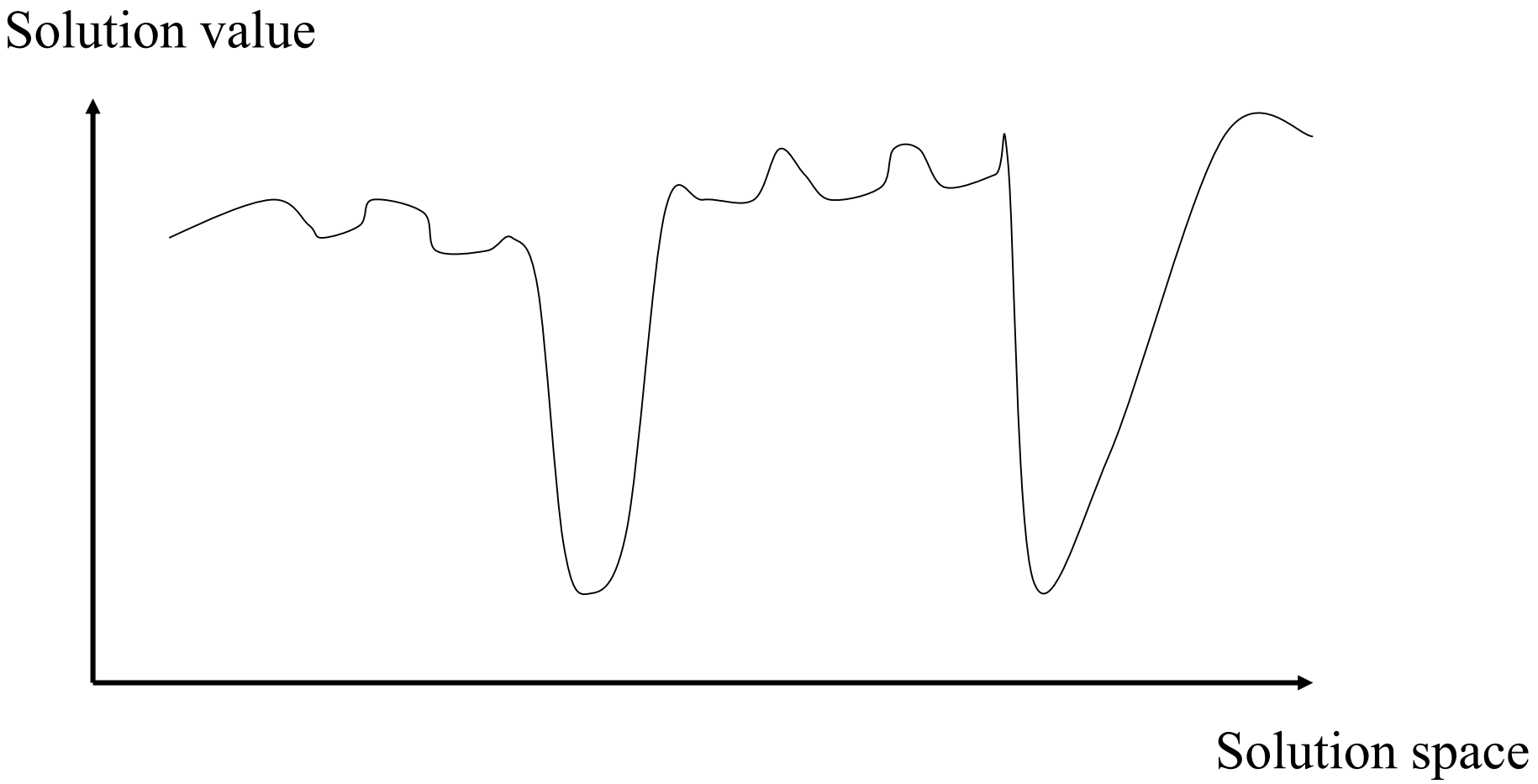
Local Search: Traversal of the Neighborhood Graph

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



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 of Local Search

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

Example: The Knapsack Problem

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

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

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

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

Example (cont.)

$$\text{Max } z = 5x_1 + 11x_2 + 9\,x_3 + 7x_4$$

$$\text{Such that: } \quad 2x_1 + \quad 4x_2 + 3x_3 + 2x_4 \leq 7$$

Example (cont.)

- The 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*

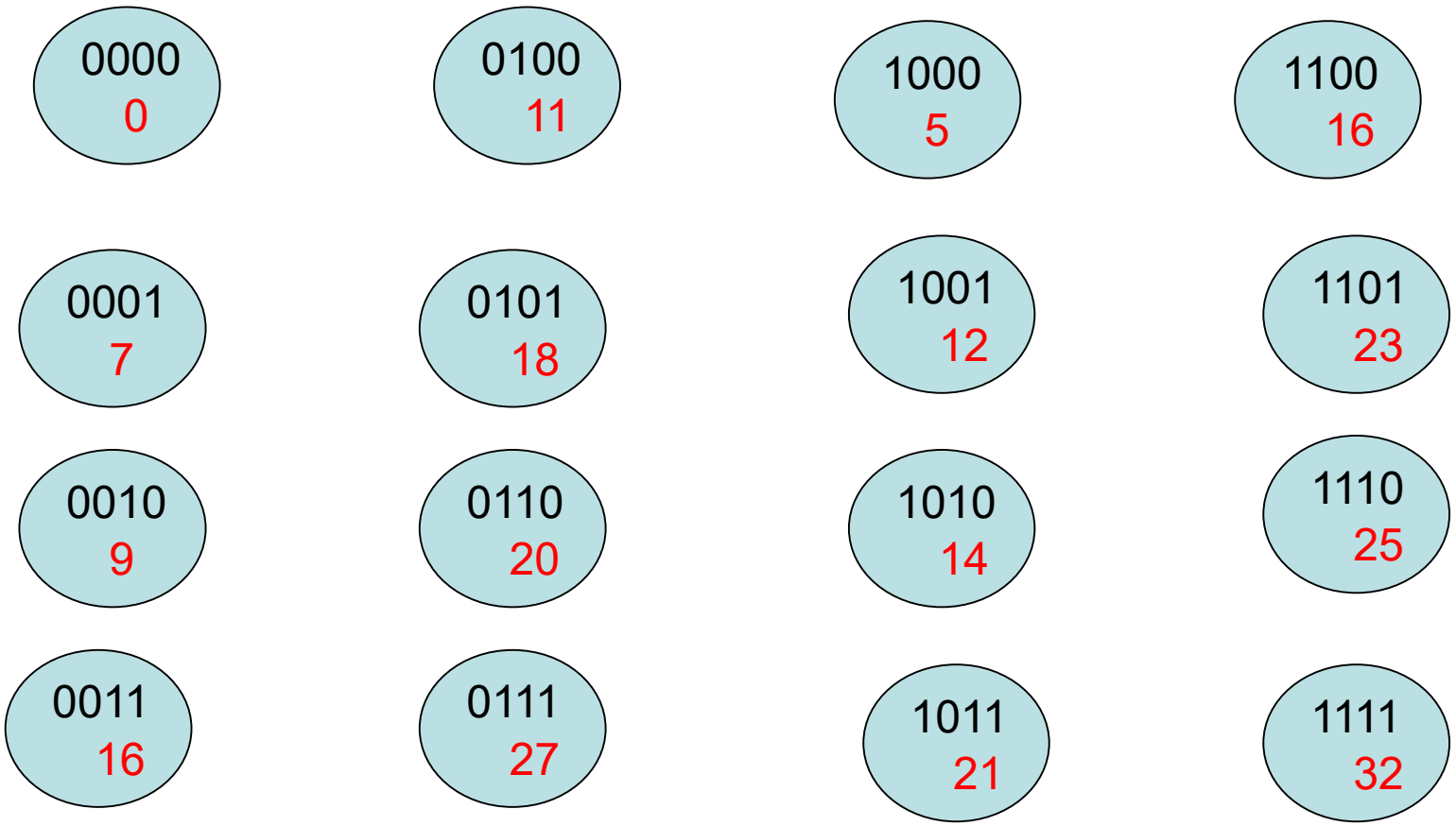
$$\max \sum_{i=1}^{n} c_i x_i$$

Search Space

xxxx ⇒ Solution

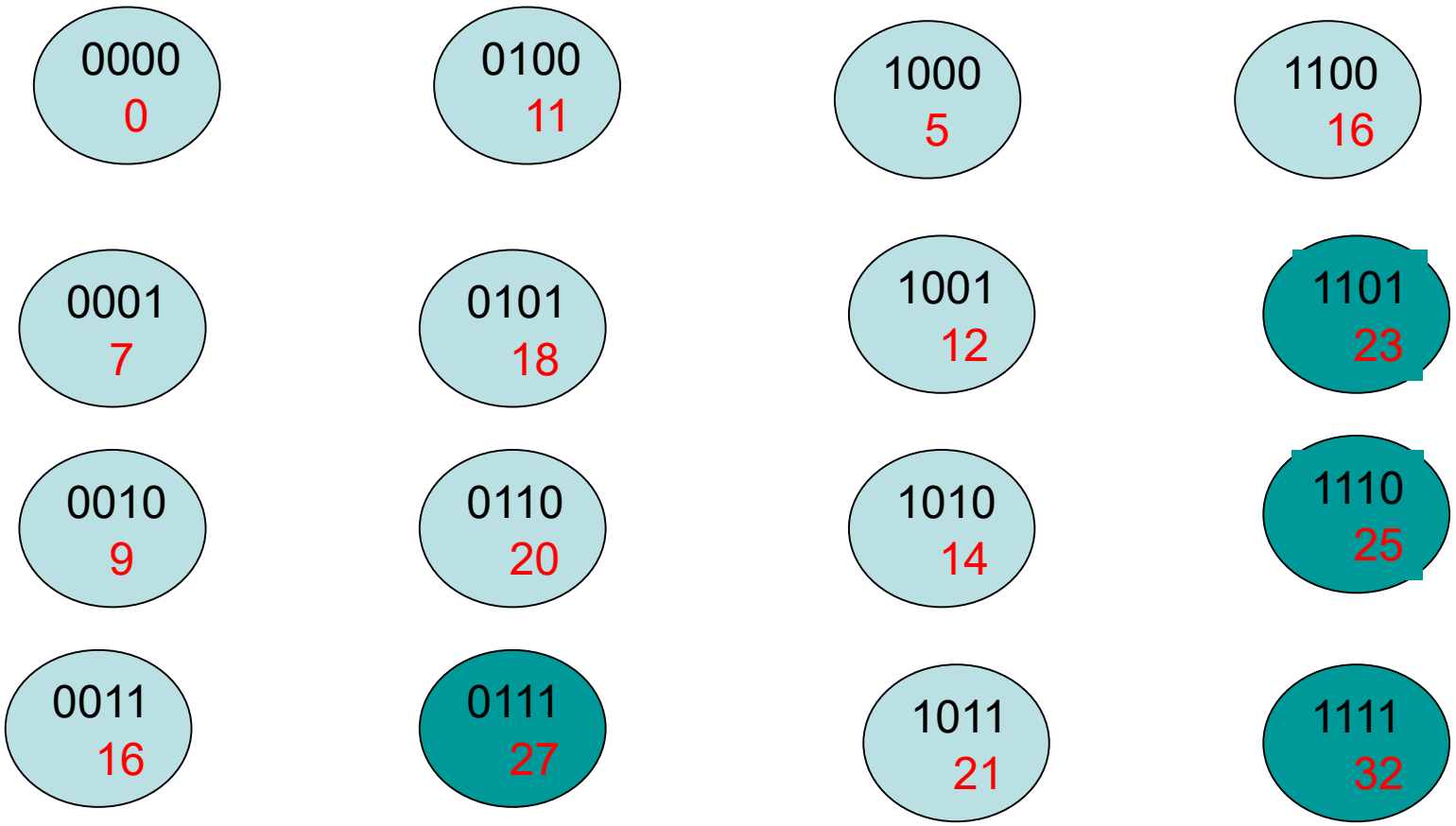
Obj. Fun. Value

- The search space is the set of solutions

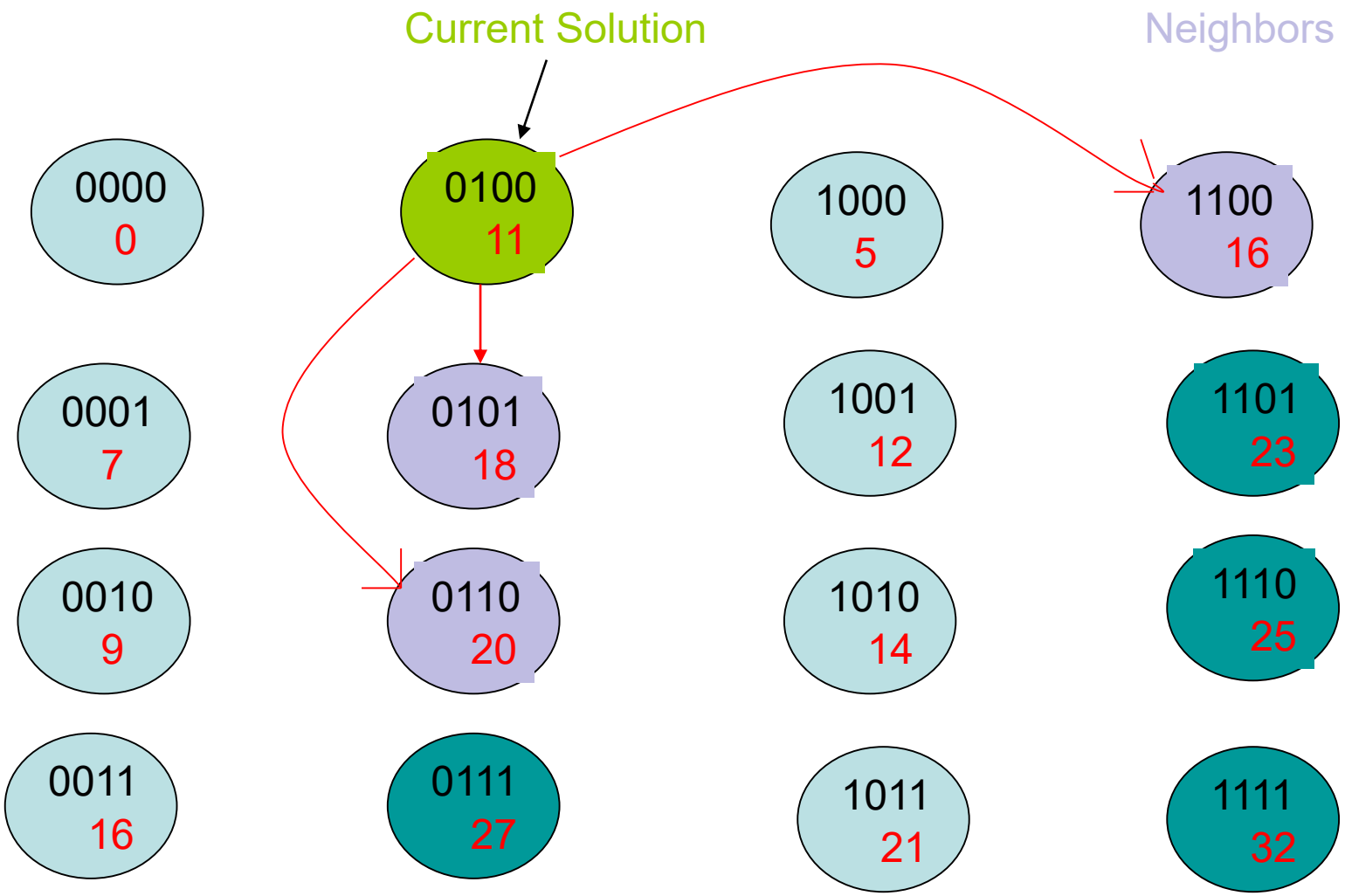


Feasible/Infeasible Space

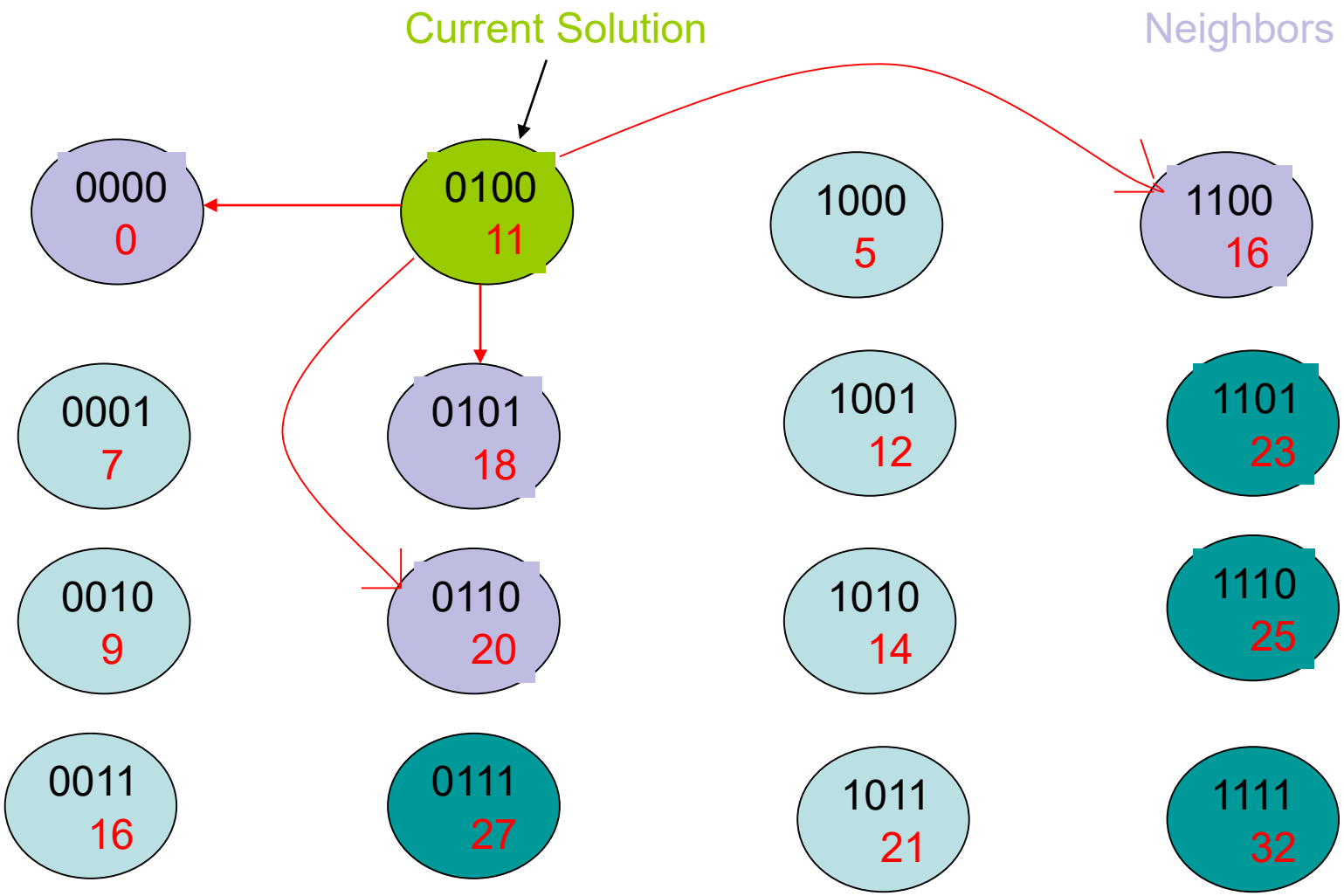
Example : Sum of weights equal or less than b=7 are feasible



Add - Neighborhood

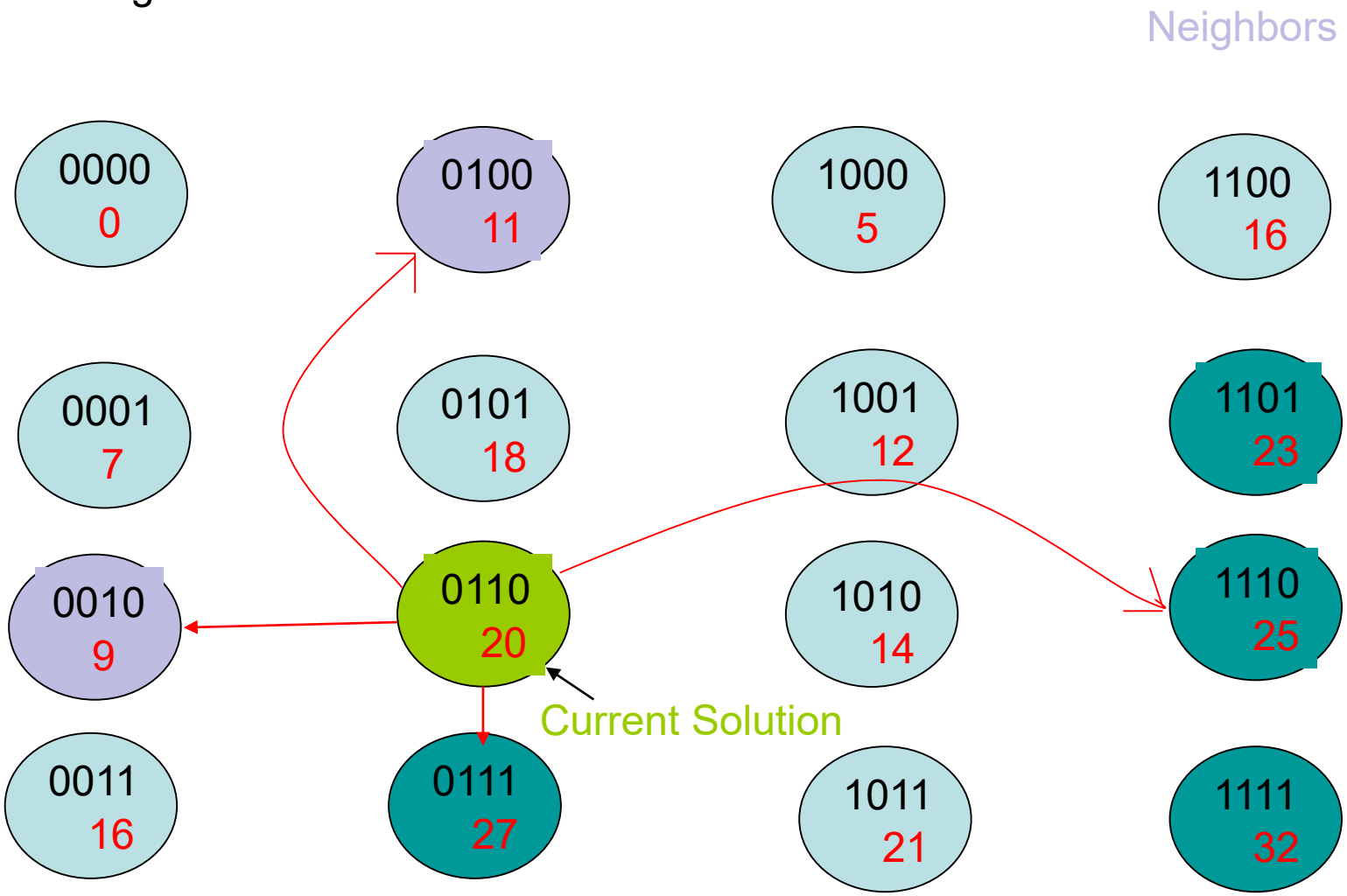


Flip Neighborhood



Applying best accept

The algorithm ends with f=20



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 easy to understand
- The use of neigborhoods sometimes makes the optimal solution seem ”close”, e.g.:
  - A knapsack has n items
  - The search space has 2^n members
  - From *any* solution, no more than n flips are required to reach an optimal solution!

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 neighborhoods can become very large (time consuming to examine all the neighbors)

Main Challenge in Local Search

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

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

Metaheuristics (2)

- Different definitions:
  - A metaheuristic is an 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 metaheuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality.
  - A metaheuristic is a procedure that has the ability to escape local optimality

Metaheuristics (2)

- Glover and Kochenberger (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 neighborhood 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)
- Scatter Search (SS)

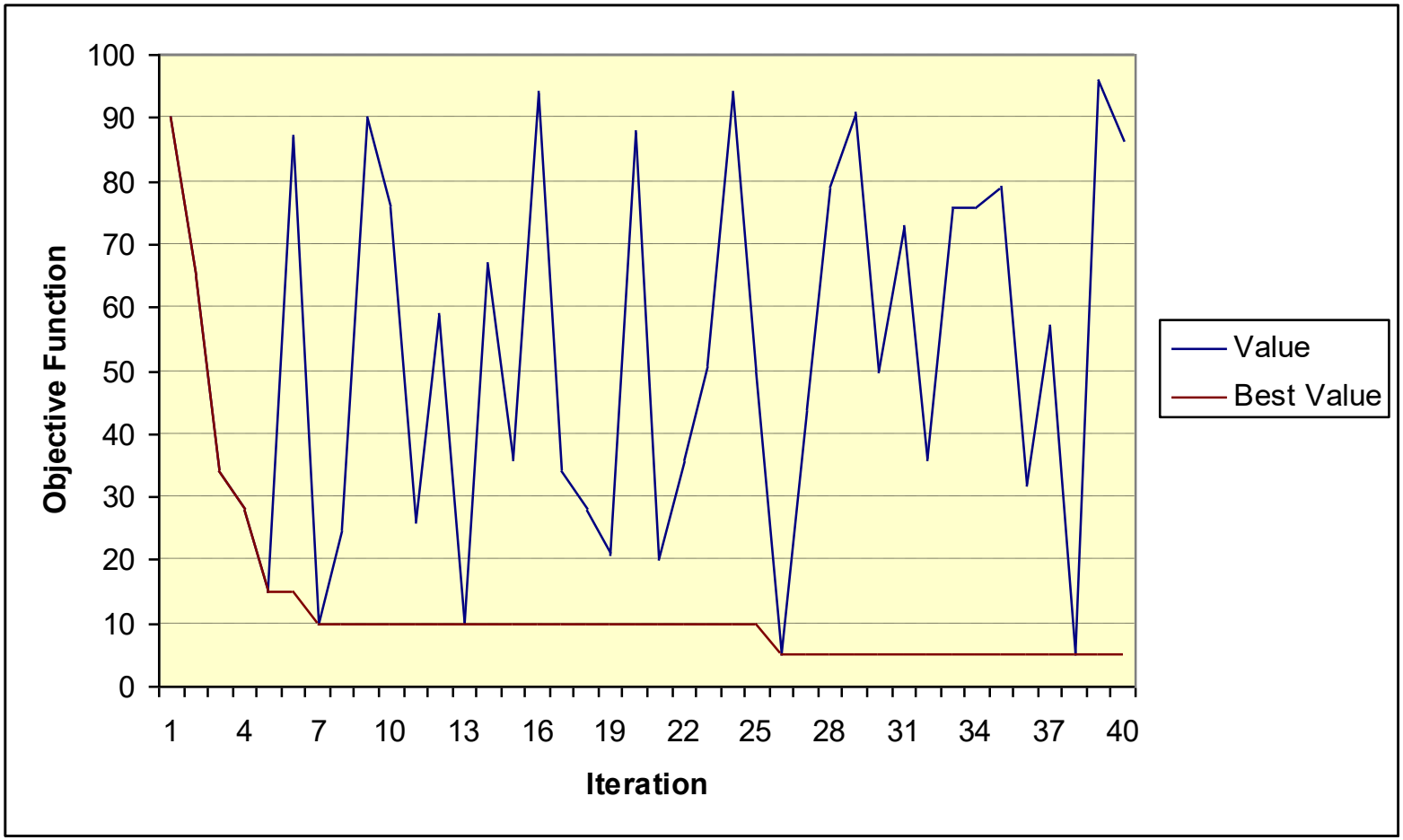
Some other Metaheuristics

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

Metaheuristic Classification

- x/y/z Classification
  - x = A (adaptive memory) or M (memoryless)
  - y = N (systematic neighborhood search) or S (random sampling)
  - z = 1 (one current solution) or P (population of solutions)
- Some Classifications
  - Simulated Annealing (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
  - We will look at Simulated Annealing and Tabu Search, as well as mention some others