# Heuristic Optimization Methods

## Local Search / Metaheuristics

# Summary of the Previous Lecture

- Some practical information
- About formal problems
  - Formulations: COP, IP, MIP, …
  - Problems: TSP, Set Covering, …
- How to solve problems
  - Exact-, Approximation-, Heuristic algorithms
- Why use heuristics?
  - Complexity (**P** vs **NP**), combinatorial explosion

# Agenda

- Local Search
- The Knapsack Problem (example)
- The Pros and Cons of Local Search
- Metaheuristics
- Metaheuristics and Local Search

# Motivation for Heuristic Solution Methods for COP (1)

- Complexity theory, NP-complete problems
- Complexity theory looks at decision problems
- Close connection between decision problems and optimization problems
- Optimization at least as hard as decison
- NP-complete decision problem -> NP-hard optimization problem
- For NP-hard COP there is probably no exact method where computing time is limited by a polynomal (in the instance size)
- Different choices
  - Exact methods (enumerative)
  - Approximation method (polynomial time)
  - Heuristic method (no a priori guarantees)
- NB! Not all COP's are NP-hard!

# Motivation for Heuristic Solution Methods for COP (2)

- In the real world:
  - Often requirements on response time
  - Optimization only one aspect
  - Problem size and response time requirements often excludes exact solution methods
- Heuristic methods are often robust choices
  - The real world often don’t need the optimal solution
  - Men are not optimizers, but ”satisficers”
    - Herb Simon
- Exact methods can be a better choice

# Exact Methods for COP

- COP has a finite number of solutions
- Exact methods guarantee to find the optimal solution
- Response time?
- Exact methods are
  - Good for limited problem sizes
  - Perhaps good for the instances at hand?
  - Often basis for approximation methods
  - Often good for simplified problems

# Heuristic

- ”A technique that improves the efficiency of a search process, usually by sacrificing completeness”
- Guarantees for solution quality vs. time can seldom be given
- General heuristics (e.g. Branch & Bound for IP)
- Special heuristics exploits problem knowledge
- The term “heuristic” was introduced in How to solve it [Polya 1957]
  - A guide for solving matematical problems

# COP Example: The Assignment Problem

- n persons (*i*) and n tasks (*j*)
- It costs $c_{ij}$ to let person *i* do task *j*
- We introduce decision variables:

$$x_{ij} = \begin{cases} 1 & \text{If person } i \text{ does task } j \\ 0 & \text{Otherwise} \end{cases}$$

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

- Find the minimal cost assignment:

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

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

# COP Example: TSPTW

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

$a_i$ Arrival time at city i

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

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

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

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

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

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

# COP Example: The Knapsack Problem

- *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

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

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

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

# 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

Earlier solution:

1 2 7 3 4 5 6 1 (184)

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 |

# Example: Knapsack Problem

| | 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 |

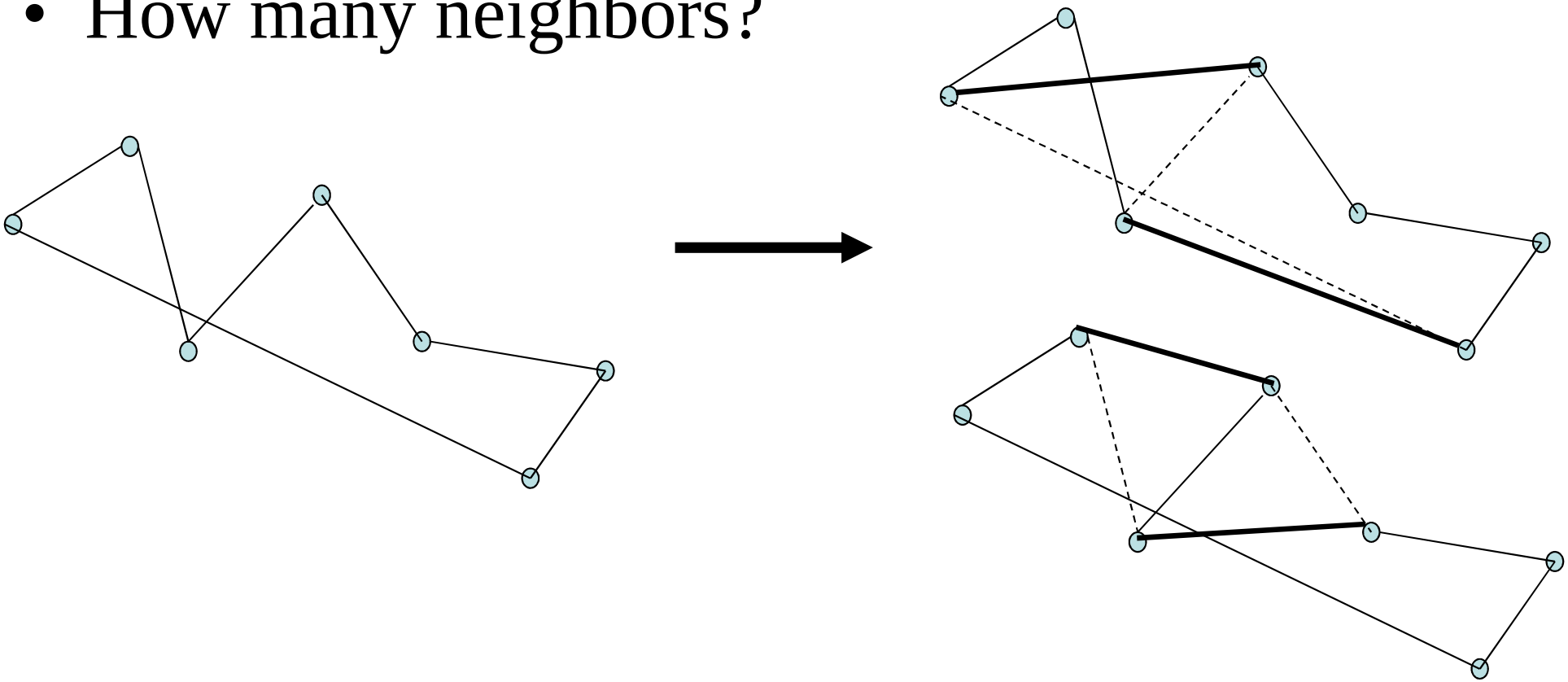
- Knapsack with capacity 101
- 10 ”items” (e.g. projects, ...) 1,...,10
- Trivial solution: empty backpack, value 0
- Greedy solution, assign the items after value:
  - (0000010000), value 85
  - Better suggestions?

# 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 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 / Neighborhood Search (1)

- Start with an *initial solution*
- Iteratively search in the neighborhood for better solutions
- Sequence 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”
- 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

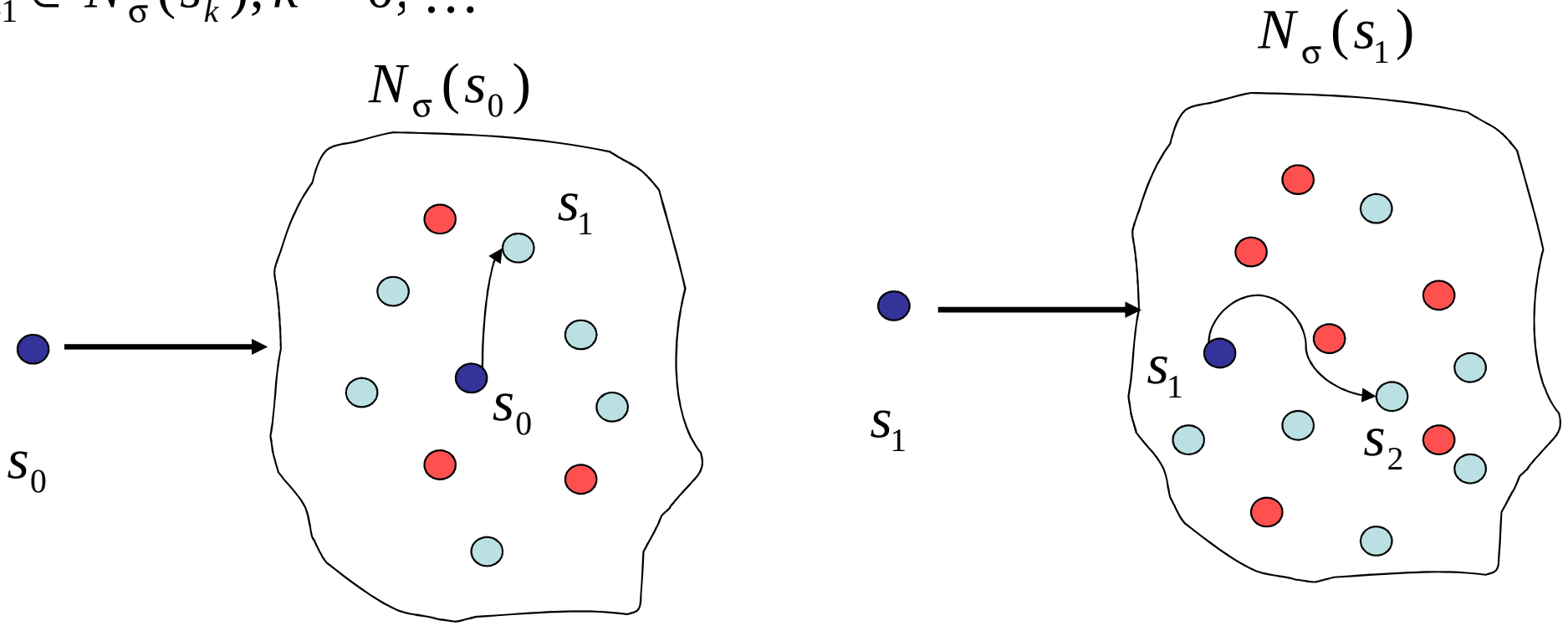
```
1: input: starting solution, s_0
2: input: neighborhood operator, N
3: input: evaluation function, f
4: current ⇐ s_0
5: done ⇐ false
6: while done = false do
7:    best_neighbor ⇐ current
8:    for each s ∈ N(current) do
9:       if f(s) < f(best_neighbor) then
10:         best_neighbor ⇐ s
11: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

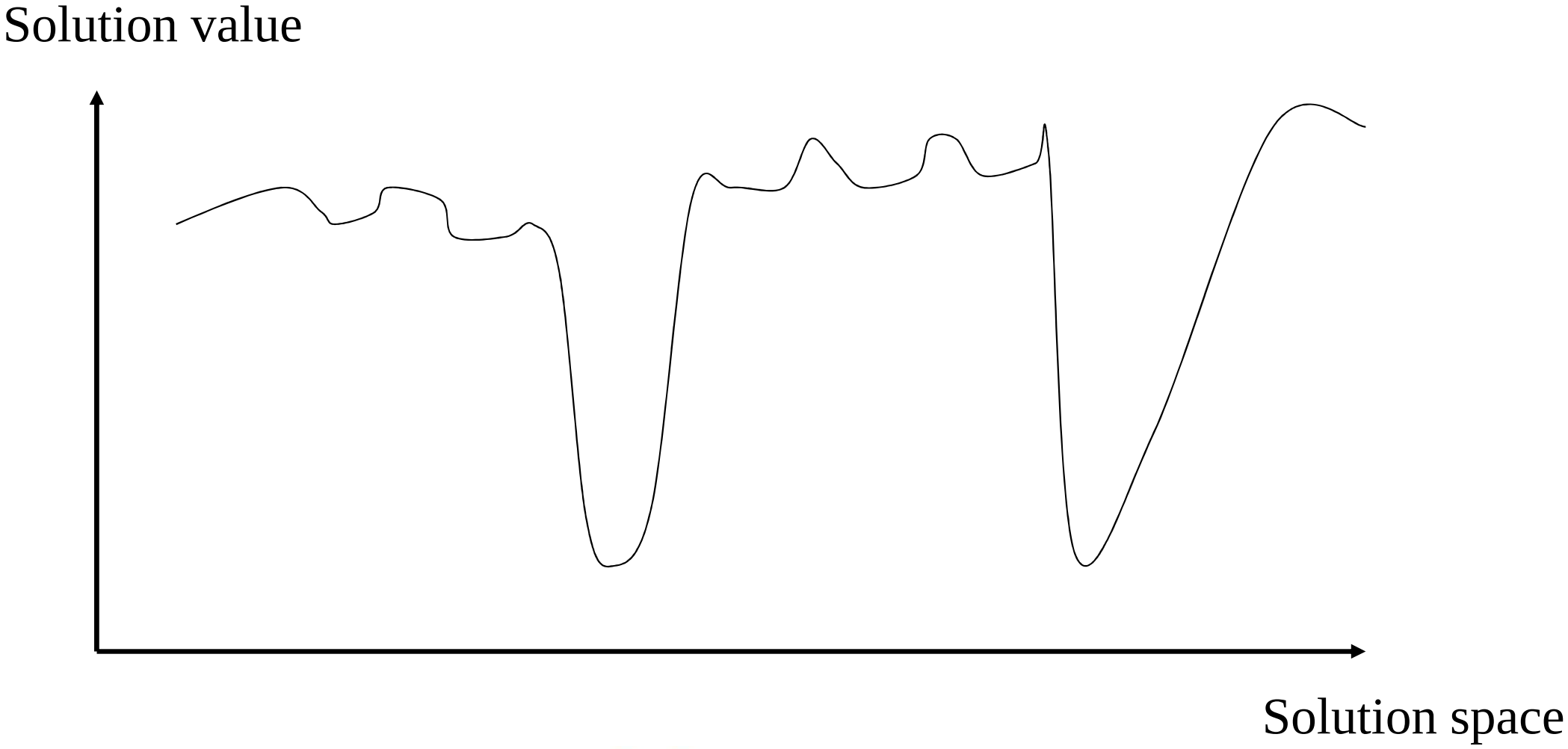
# Local Search: Traversal of the Neighborhood Graph

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




**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.)

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

Such that: $2x_1 + 4x_2 + 3x_3 + 2x_4 \leq 7$

# 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*

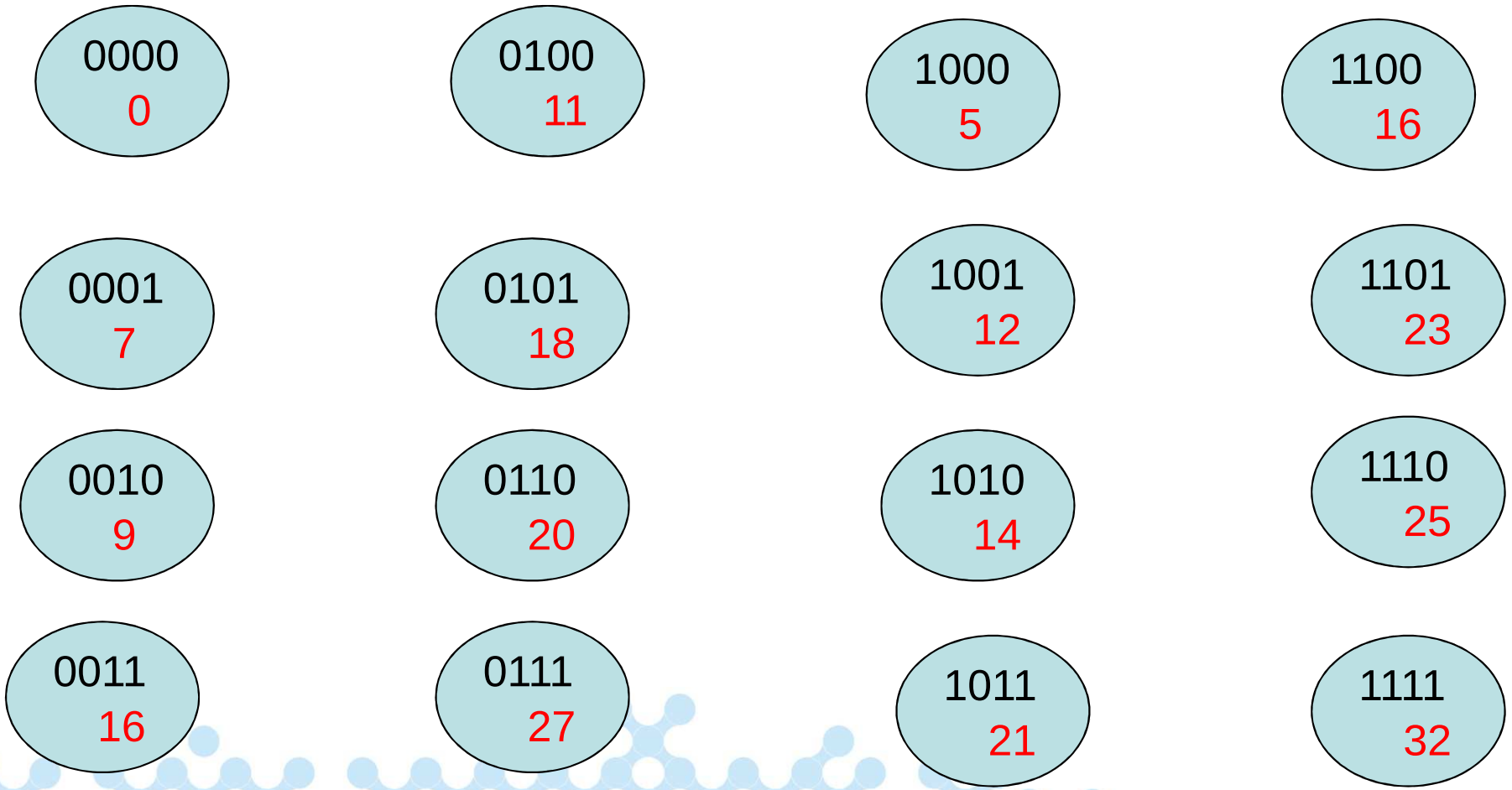
$$\max \sum_{i=1}^{n} c_i x_i$$

# Search Space

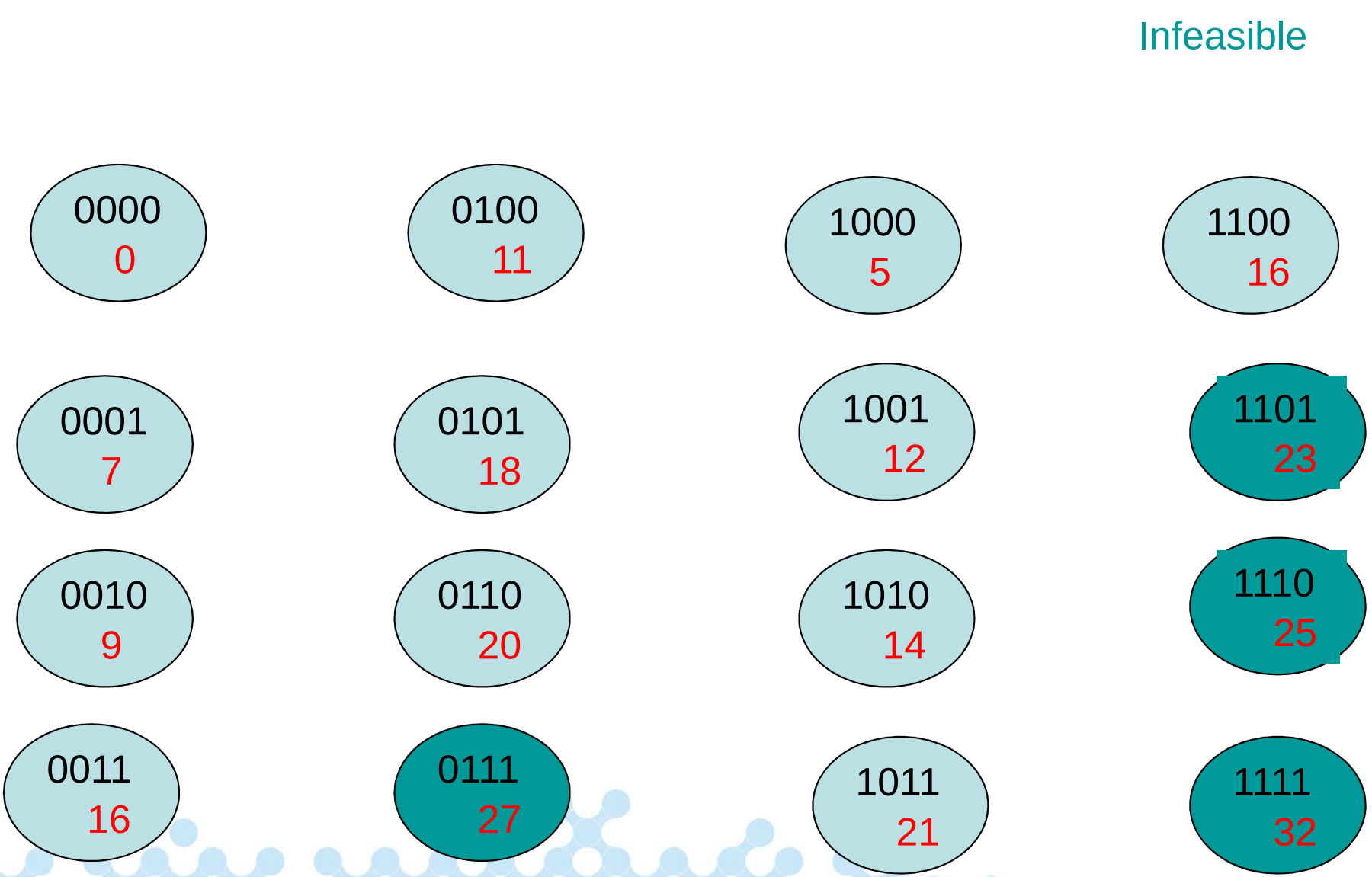
xxxx ⇒ Solution

Obj. Fun. Value

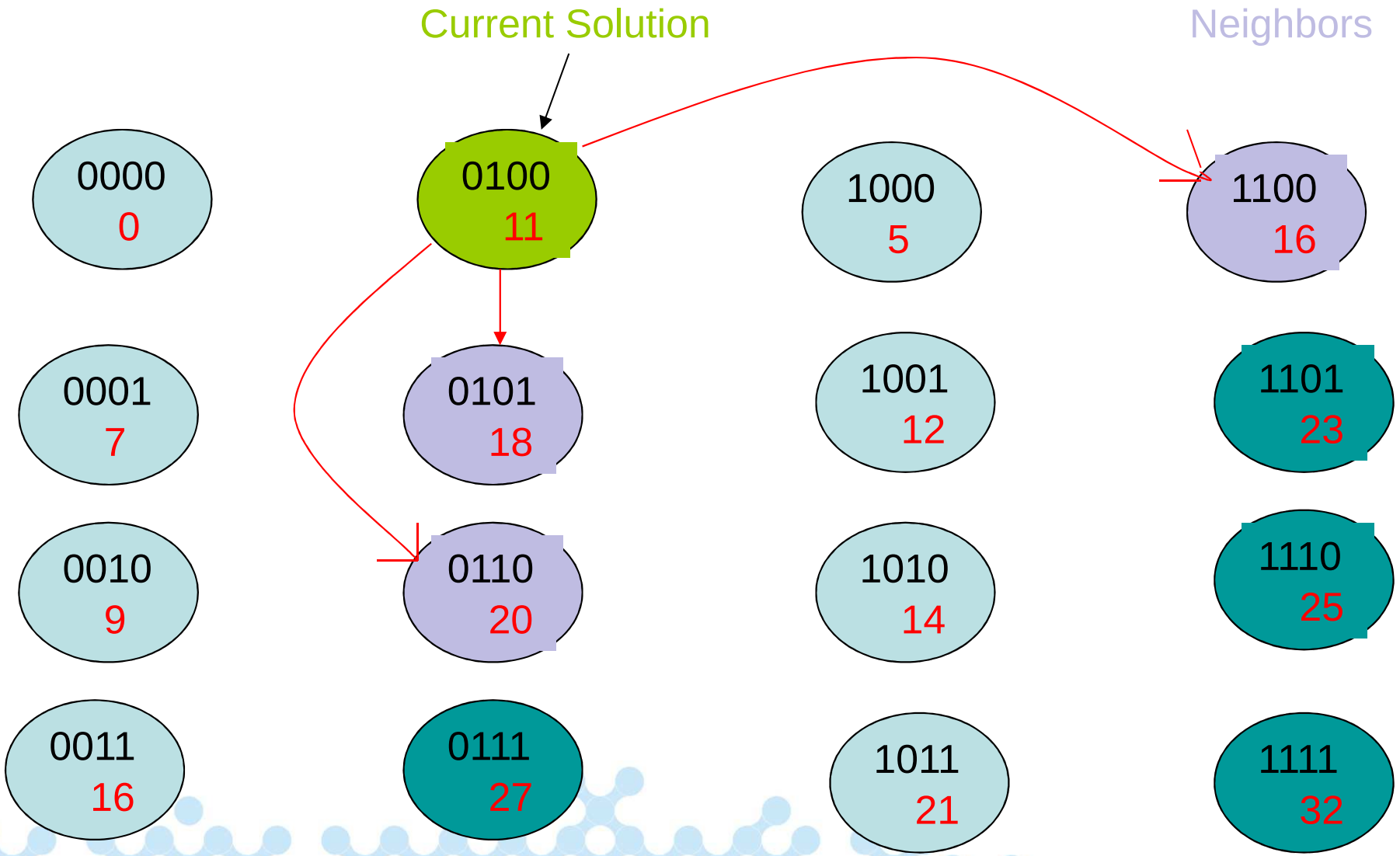
- The search space is the set of solutions



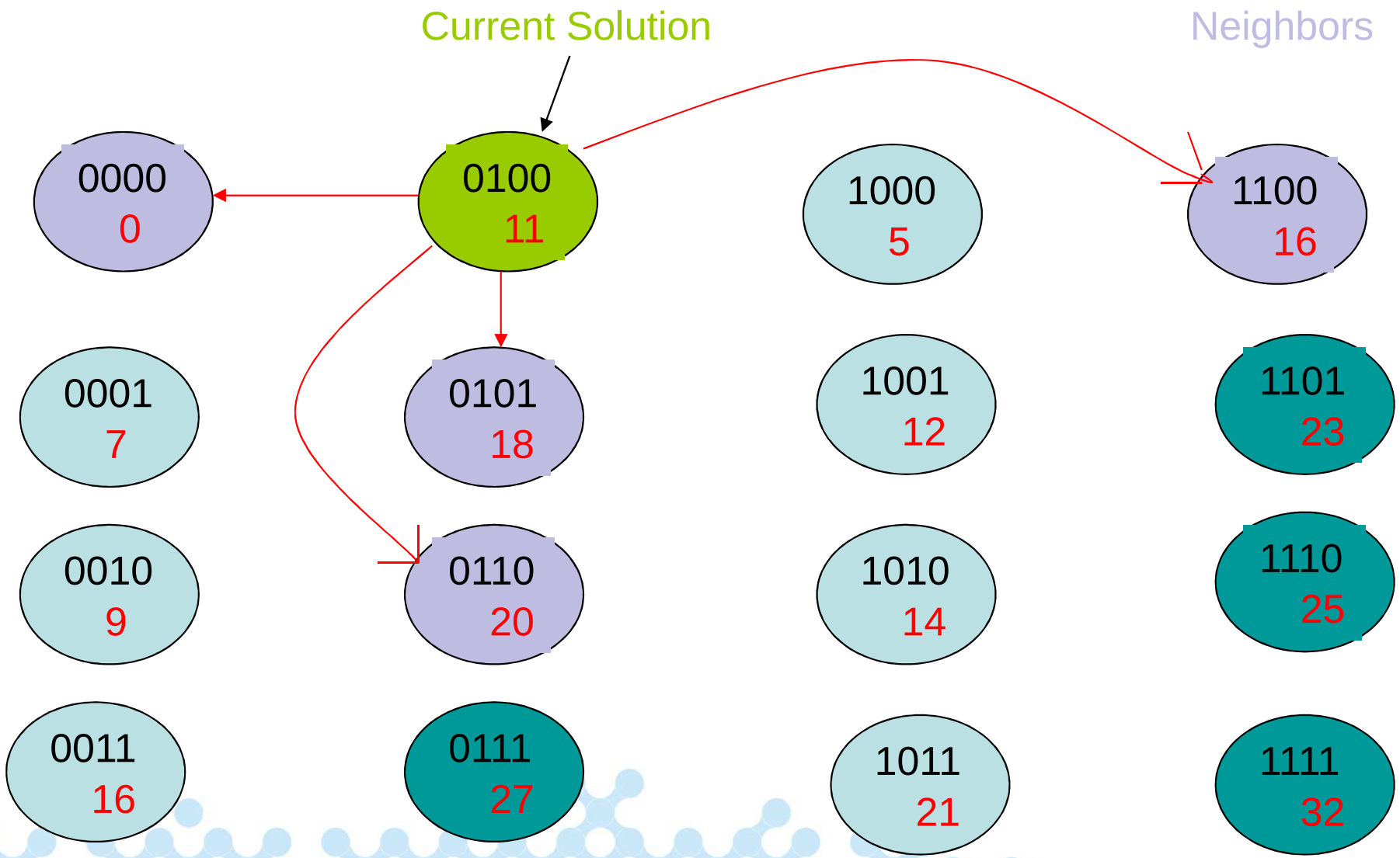
# Feasible/Infeasible Space



# Add - Neighborhood



# Flip Neighborhood



# 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 search 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)

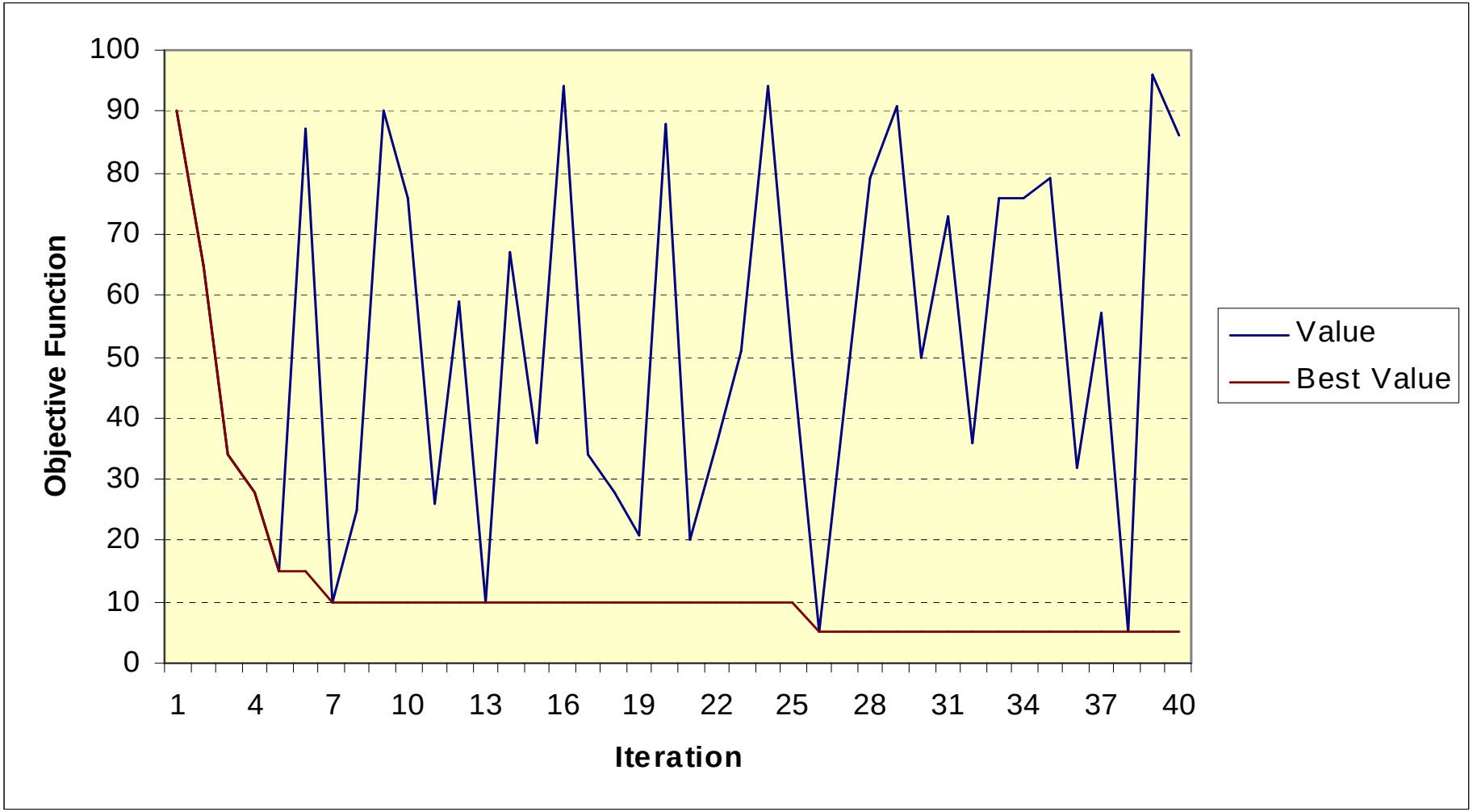
# Some other Metaheuristics

- Scatter Search (SS)
- 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)
- Particle Swarm Optimization (PSO)
- And several others…
  - 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
  - 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
  - We will look at Simulated Annealing and Tabu Search, as well as mention some others

# Summary

- Local Search
  - Example: Knapsack Problem
- Metaheuristics
  - Classification
- Metaheuristics based on Local Search
  - Escaping local optima