PA184 - Heuristic Search Methods

Lecture 2 – Principles of Heuristic Search

- Traditional Techniques
- Motivation for Heuristic Search
- Basic Concepts in Heuristic Search
- Description of Seminar Activity 2

Learning outcomes:

- appreciate the limitations of some classical search algorithms
- understand the key differences between heuristic and classical optimisation methods
- describe the key concepts of heuristic search
- analyse some representations, fitness functions and greedy algorithms for some COPs

Traditional Techniques

Mathematical Programming

Refers to modelling optimisation problems using formulations based on mathematical expressions. Examples of classical models are:

- Linear programming (LP) problems
- Integer programming (IP) problems
- Mixed integer programming (MIP) problems

Then, the models are solved with classical techniques such as:

- Simplex method
- Branch and bound
- Branch and cut
- Dynamic programming

Even with improved versions of the above techniques, solving very large problems might not be practical.

The general LP formulation has the following elements:

- Parameters
- Decision variables
- Linear objective function
- Linear constraints

An LP problem induces a search space with feasible and infeasible regions. The goal is to find the optimal solution(s) which are located in the corner points of the feasible region.

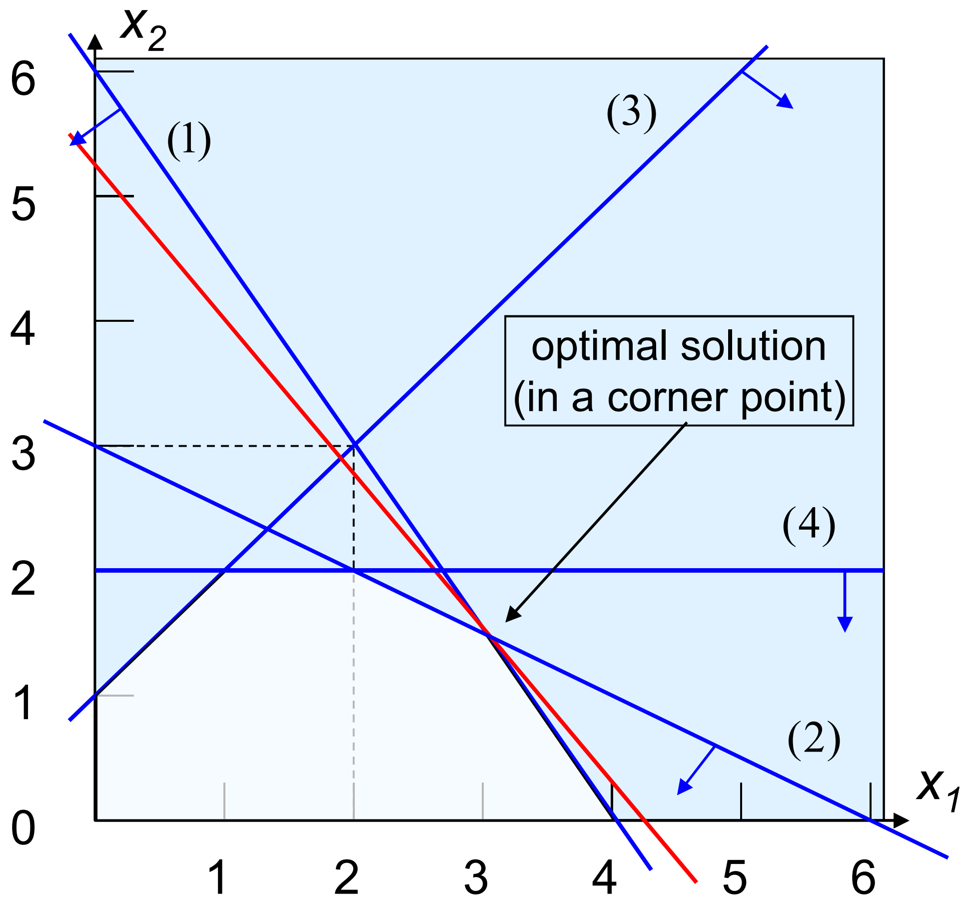
$$\text{Maximise} \quad Z = c_1 x_1 + c_2 x_2 + \ldots c_n x_n$$

subject to constraints

$$\begin{aligned}
&a_{11} x_1 + a_{12} x_2 + \ldots a_{1n} x_n \le b_1 \\
&a_{21} x_1 + a_{22} x_2 + \ldots a_{2n} x_n \le b_2 \\
&\vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\
&a_{m1} x_1 + a_{m2} x_2 + \ldots a_{mn} x_n \le b_m \\
&x_1 \ge 0, x_2 \ge 0 \ldots x_n \ge 0
\end{aligned}$$

The linear conditions and non-integral requirements of LP models allow to establish optimality conditions exploited by solvers. This does not happen with IP,BIP, MIP models.

$$
\begin{array}{lll}
\text{Maximise} & Z = 5x_1 + 4x_2 & \\
\text{subject to} & 6x_1 + 4x_2 \leq 24 & (1) \\
& x_1 + 2x_2 \leq 6 & (2) \\
& x_2 - x_1 \leq 1 & (3) \\
& x_2 \leq 2 & (4) \\
& x_1 \geq 0, x_2 \geq 0 &
\end{array}
$$



LP solution: $\boxed{\begin{array}{l} x_1 = 3.0 \\ x_2 = 1.5 \\ Z = 21.0 \end{array}}$

IP solution: $\boxed{\begin{array}{l} x_1 = 4 \\ x_2 = 0 \\ Z = 20 \end{array}}$

Branch and Bound (B&B) seeks to reduce the search space explored by eliminating non-attractive alternative solutions.

Two aspects are important to make B&B an efficient search method:

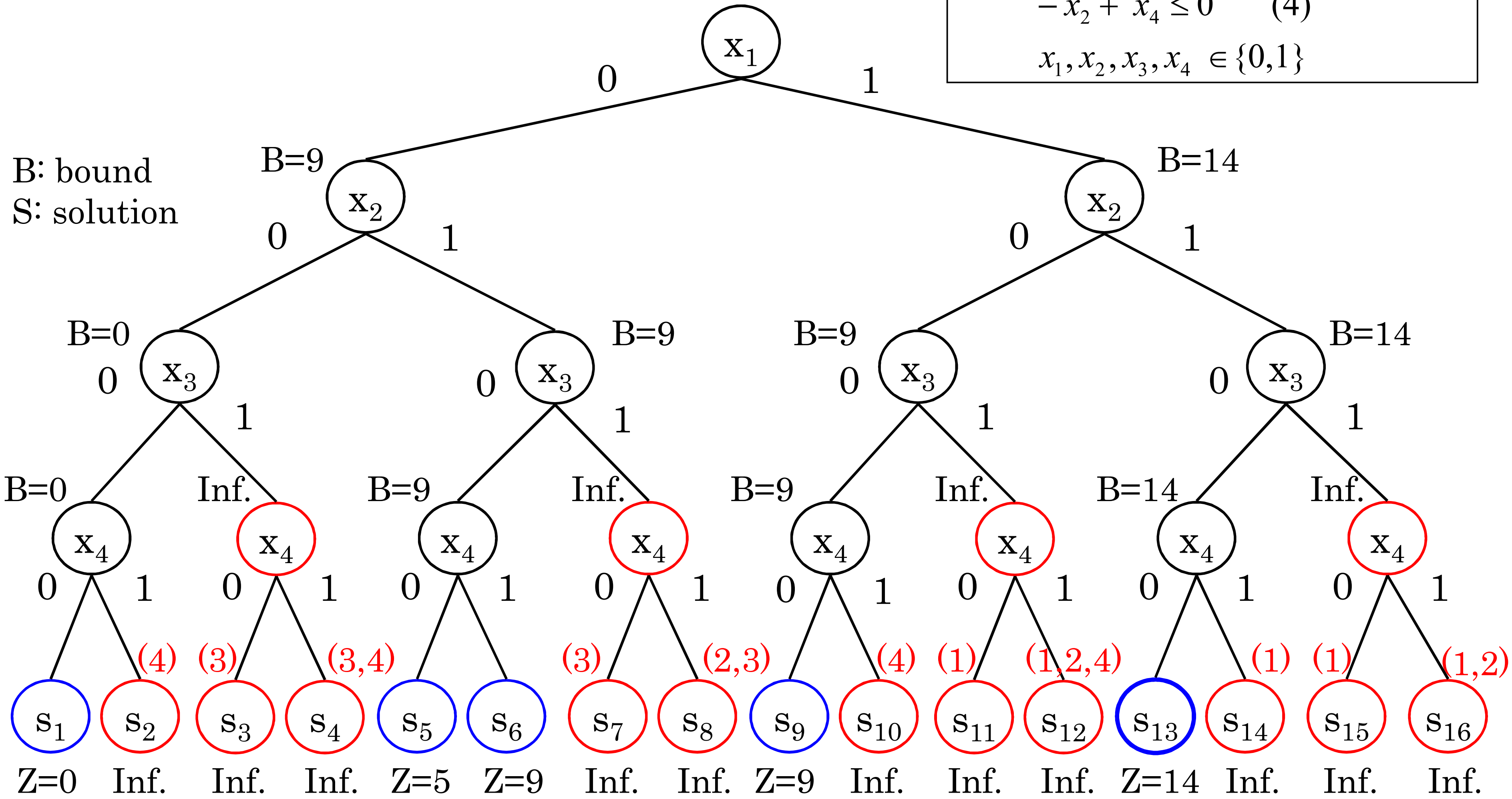
- Branching strategy
  - Depth-first (branch and backtrack)
  - Breadth-first
  - Based on increasing order of cost
- Quality of bounds (for pruning the tree)
  - Based on explored complete solutions
  - Based on relaxations (LP, Lagrangean, etc.)
  - Heuristics to generate bounds

Also, problem-domain knowledge can be incorporated to make B&B more efficient for some specific problems through improvements in the branching technique and computation of tighter bounds.

An illustration of B&B

$$\begin{aligned}
\text{Max} \quad & Z = 9x_1 + 5x_2 + 6x_3 + 4x_4 \\
\text{s.t.} \quad & 6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10 \quad (1) \\
& x_3 + x_4 \leq 1 \quad (2) \\
& -x_1 + x_3 \leq 0 \quad (3) \\
& -x_2 + x_4 \leq 0 \quad (4) \\
& x_1, x_2, x_3, x_4 \in \{0,1\}
\end{aligned}$$

B: bound
S: solution



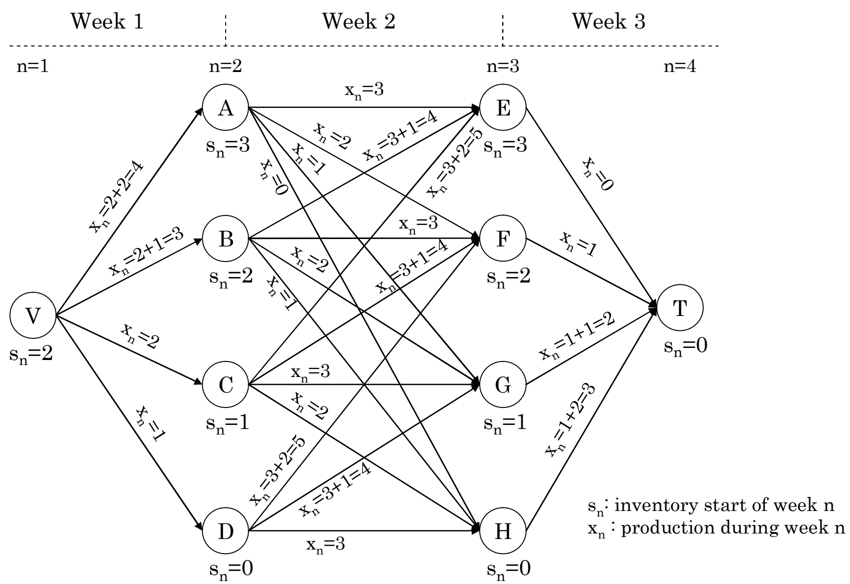
Dynamic Programming (DP) seeks to solve the problem in stages by breaking down the problem into simpler problems.

DP is a systematic procedure to determine the optimal combination of decisions that generates an optimal solution given as the overall optimal policy.

DP algorithms are designed specifically for the problem in hand.

The key ingredients of a dynamic programming approach are:

- Stages
- States
- Decisions
- Policies
- Cost (state-state)



Motivation for Heuristic Search

Real-world Optimisation Problems

Real-world optimisation problems (combinatorial or continuous) are difficult to solve for several reasons:

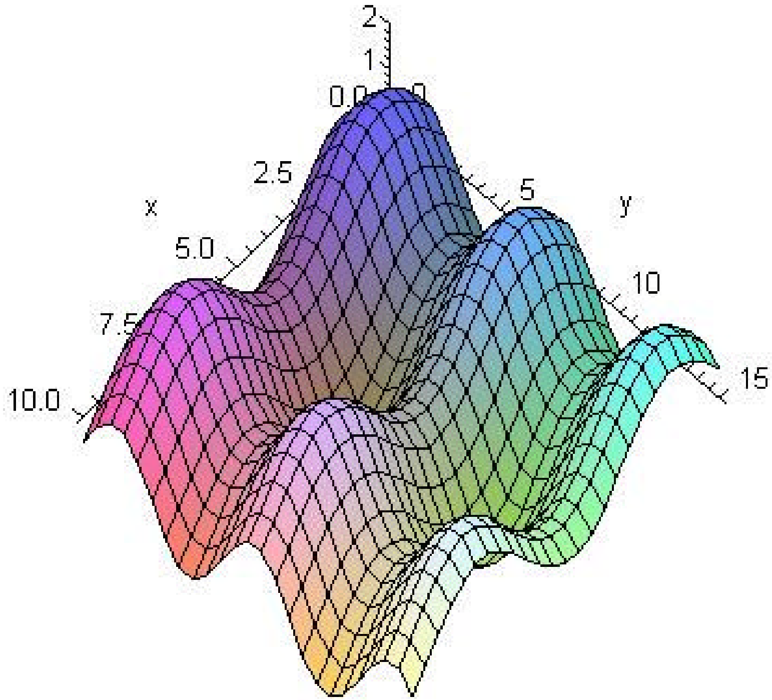
- Size of the search space is very large (exhaustive search impractical)
- Simplified models are needed to facilitate an answer
- Difficult to design an accurate evaluation function
- Existence of large number of constraints
- Difficult to collect accurate data
- Human limitations to construct solutions

Size of the Search Space |S|

In both continuous and combinatorial problems, |S| depends on the number of variables and their number of permissible values.

$$f(x,y) = x \cdot exp^{\left(-(x^2+y^2)+\frac{x^2+y^2}{20}\right)}$$

s.t. $0 < x < 10$ and $0 < y < 15$

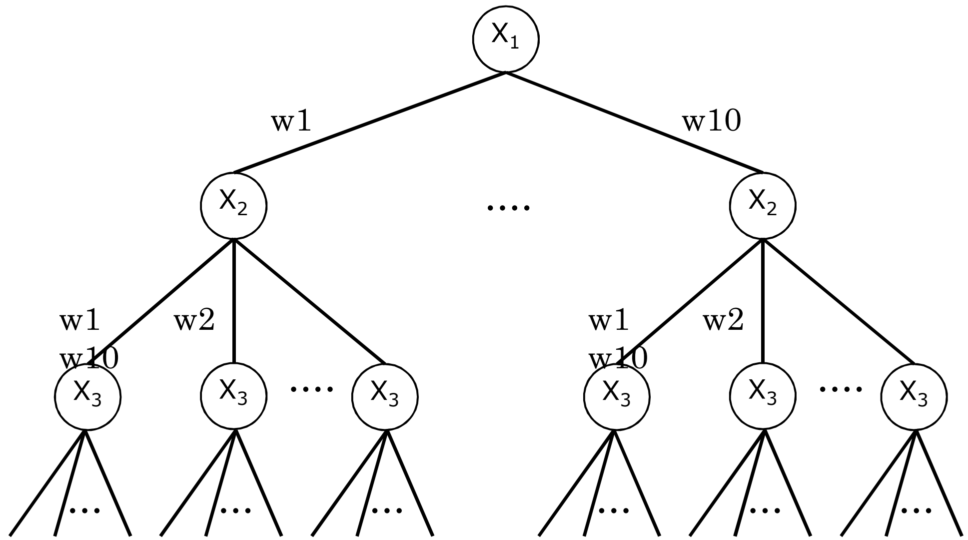


Minimise $Z = \sum_{j=1}^{10}\sum_{i=1}^{50} c_{ij}x_{ij}$

subject to $\sum_{j=1}^{10} x_{ij} = 1$ for $i = 1 \ldots 50$ (1)

$\sum_{i=1}^{50} t_{ij}x_{ij} \le T_j$ for $j = 1 \ldots 10$ (2)

$x_{ij} = 1$ if task i assigned to worker j, 0 otherwise



Problem vs. Model

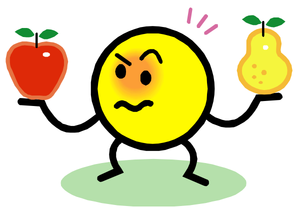
A solution is only a solution in terms of the model used to represent the problem.

Most times, assumptions have to be made to simplify the complexity of real-world problems.

Approximate Solution Method

⇩

Precise Model



Precise Solution Method

⇩

Approximate Model

Constraints

Hard constraints must be satisfied for a solution to be feasible.

Soft constraints are not mandatory but desirable conditions in good quality solutions.

Example 2.1. Consider the following variant of the GAP (generalised assignment problem):

Given n tasks, m workers, c_{ij} is cost of assigning task i to worker j. Worker j has limited time available T_j. Worker j takes t_{ij} time to complete task i. Each task is assigned to exactly one worker. A worker can undertake more that one task depending on T_j. No worker can have more than k tasks assigned. If possible, each worker in the subset W should have p percent of their available time free and each worker in W must not have only one task assigned. The set T_L of large tasks should be uniformly distributed among all workers. The set D of 'new' workers should not be assigned more than h tasks. Assign all the tasks to the workers so that the total cost is minimised without exceeding T_j for any worker.

How many hard constraints?

How many soft constraints?

Basic Concepts in Heuristic Search

Solution Representation

- Different representations (encodings) for the same problem.
- Defines the size of the search space.
- An appropriate encoding (creativity?) is crucially important.

Objective

- Minimise or maximise or find one solution?

Evaluation Function (fitness function)

- Maps the representation of a solution to a numeric value that indicates the quality of the solution.
- Exact or approximate evaluation?
- Differentiate between feasible and infeasible solutions.

Search Problem

Given search space S and a set of feasible solutions $F \subseteq S$, a search problem is to find a solution $x \in S$ such that:

$$\text{fitness\_function}(x) \le \text{fitness\_function}(y) \quad \forall y \in F$$

Two solutions x,y are said to be neighbours if they are 'close' to each other in the sense that the distance between their corresponding encodings or representations is within a certain limit.

Some search algorithms work with complete solutions, i.e. try to go from the current solution to a better one.

Example: neighbourhood search

Other search algorithms work with partial solutions, i.e. construct a complete solution step by step.

Example: greedy algorithms

Example 2.2. Consider a symmetric n-city TSP over a complete graph.

Representation

permutation of integers 1 to n e.g. 13546278

total number of permutations : $n!$

ignoring symmetric tours : $\frac{n!}{2}$ e.g. $13546278 = 87264531$

ignoring shifted identical tours : $\frac{(n-1)!}{2}$ e.g. $13546278 = 54627813$

Evaluation Function

S is set of solutions and a solution $s = c_1 c_2 c_3 \cdots c_n$ then

$$f(s) = d(c_1, c_2) + d(c_2, c_3) + \cdots d(c_{n-1}, c_n) + d(c_n, c_1)$$

Objective

find $s \in S$ such that $f(s) \leq f(s') \quad \forall s' \in S$

i.e. find a tour with the minimum length

Neighbour Solutions

2 - opt move : interchanges 2 non - adjacent edges

current : 13546278 neighbours : 13246578, 13746258, 13586274, etc.

2 - right_insert move : moves a city 2 positions to the right

current : 13546278 neighbours : 15436278, 13542768, etc.

Example 2.2 (cont.)

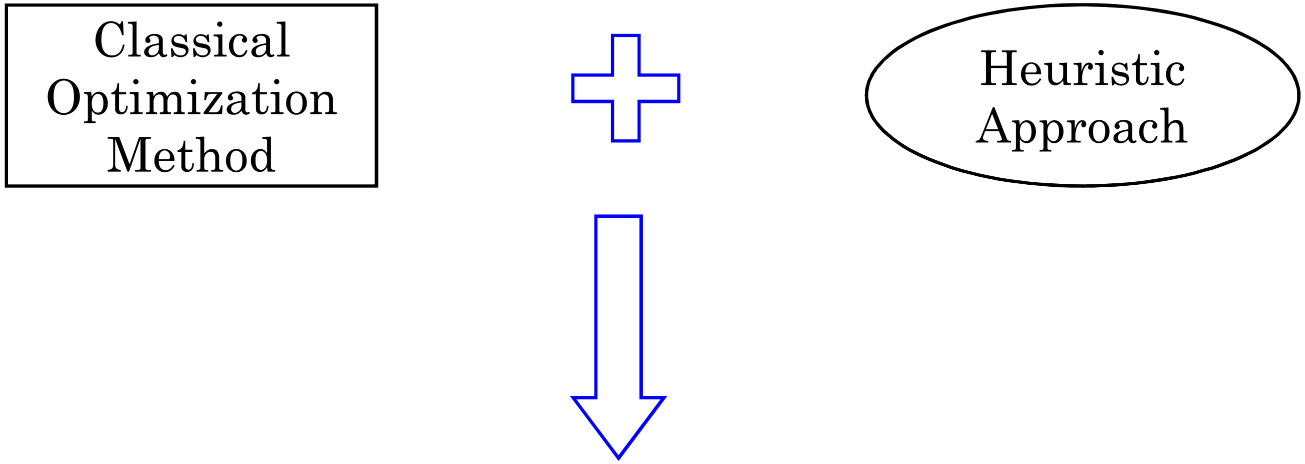
Greedy Algorithm 1

1. Select random starting city
2. Proceed to nearest unvisited city
3. Repeat step 2 until all cities are visited
4. Return to starting city

Greedy Algorithm 2

1. Find shortest edge (c_i, c_j) and add cities c_i, c_j to the tour
2. Select next chapest edge (c_r, c_t)
   (making sure no city is visited more than once)
3. Add cities c_r, c_t to the tour
4. Repeat steps 2 - 3 until a tour is formed

Integrating Classical Methods and Heuristics



- Use a classical method as part of a multi-stage approach in order to solve a part of the problem.
- Embed a classical method within a heuristic approach.
- Embed a heuristic approach within a classical method.
- Cross-fertilization by incorporating an ingredient of a classical method into a heuristic approach.

Additional Reading

Chapters 1-4 of (Michalewicz,2004)

D. Landa-Silva, F. Marikar, K. Le (2009). *Heuristic approach for automated shelf space allocation*. Proceedings of the 2009 ACM Symposium on Applied Computing (SAC 2009), Vol. 2, ACM press, 922-928.

E.K. Burke, J.D. Landa Silva (2004). *The design of memetic algorithms for scheduling and timetabling problems*. In: Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing, Vol. 166, Springer, 289-312.

Seminar Activity 2

The purpose of this seminar activity is to achieve an understanding of solution representation, greedy algorithms and neighbourhoods within the context of heuristic search.

For the GAP described in Lecture 1, do the following:

1. Write the corresponding mathematical programming model.

2. For the variant of Example 2.1, indicate how the soft constraints are incorporated into the model.

3. Describe 2 or 3 alternative solution representation schemes.

4. Describe 1 or 2 greedy algorithms that can be used to generate solutions to the problem.

5. Describe some neighbourhood moves that can be used to generate neighbouring candidate solutions from a given solution X.