# CSCU9YE - Artificial Intelligence

## Lecture 3: Problem Solving by Search (Optimisation)

Prof. Gabriela Ochoa, University of Stirling

# Content

1. Problem solving and search
2. Optimisation problems
    - Definition and applications
    - Two concrete examples
        - The *Knapsack Problem*
        - The *Traveling Salesman Problem* TSP (next week)
3. Heuristics
4. Optimisation and search

# *Search* in Computing Science

## At least 4 meanings of the word ***search*** in CS

### 1. Search for stored data

- Finding information stored in disc or memory.
- Examples: Sequential search, Binary search

### 2. Search for web documents

- Finding information on the world wide web
- Results are presented as a list of results

### 3. Search for paths or routes

- Finding a set of actions that will bring us from an initial stat to a goal stat
- Relevant to  AI
- Algorithms: depth first search, breadth first search, branch and bound, A*, Monte Carlo tree search.

### 4. Search for solutions

- Find a solution in a large space of candidate solutions
- Relevant to AI, Optimisation, OR
- Algorithms: evolutionary algorithms, Tabu search, simulated annealing, ant colony optimisation, etc.

# Examples of search problems

- A robot vehicle would search for a route to a given destination.
- An automated air traffic controller would search for a safe landing sequence for a set of incoming planes
- In games of strategy, such as chess or checkers: search for a sequence of moves to beat your opponent
- More generally: trying to find a particular object from a large number of such objects.
- Search problems are common in AI: Planning and Learning.
- Optimisation problems can be seen as  type of search problems (search for solutions, instead of search for a sequence of actions)

# Optimisation problems

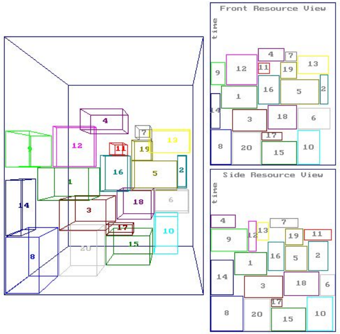
- Wide variety of applications across industry, commerce, science and government
- Optimisation occurs in the minimisation of time, cost and risk, or the maximisation of profit, quality, and efficiency
- Examples
    - Finding shortest round trips in graphs (TSP)
    - Planning, scheduling, cutting & packing, logistics, transportation, communications, timetabling, resource allocation, genome sequencing
    - Software engineering: test case minimisation and prioritisation, requirements analysis, code design and repair, etc.

# Optimisation problems are everywhere!

Logistics, transportation, supply change

Manufacturing, production lines

Timetabling




Cutting & packing

Computer networks and Telecommunications

Software - SBSE

# Optimisation problems

General constrained optimisation problem:

Min/Max $f(x)$

Subject to:

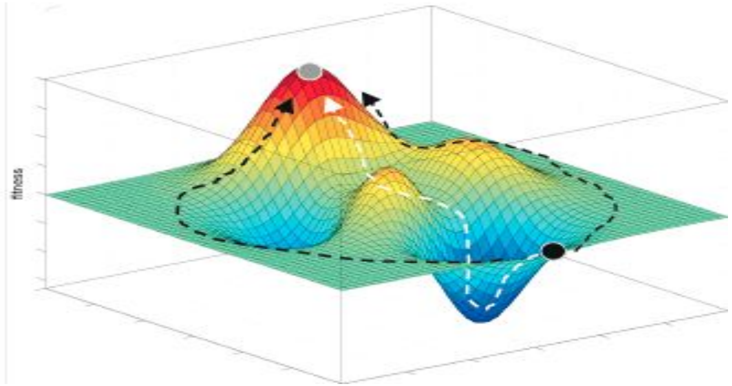
- Equality constraints
- Inequality constraints

**Search Space**: set of candidate solutions. All possible combinations of the decision variables.

**Optimisation through search**

Iteratively generate and evaluate candidate solutions.

- Systematic search
- (Stochastic) local search



# Optimisation problems: two categories

## Continuous

- *Continuous* variables
- Looking for a set (vector) of real numbers [45.78, 8.91, 3.36]
- Objective function has a mathematical expression
- Special cases studied in mathematics and OR: *Convex*, *Linear*

## Combinatorial

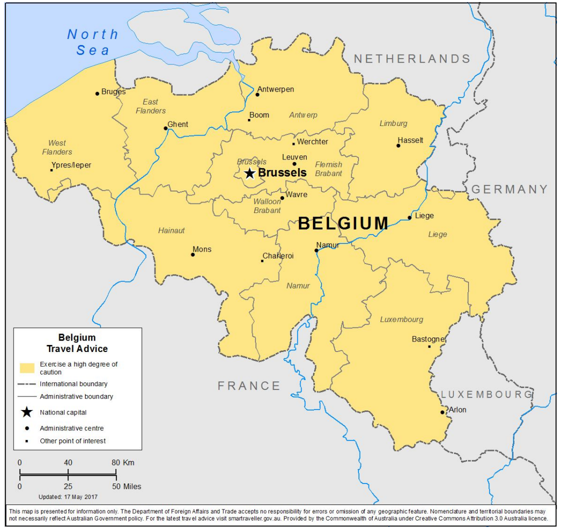
- *Discrete* variables
- Looking for an object from a finite set
  - Binary digits [1011101010]
  - Integer [1, 53, 4, 67, 39]
  - Permutation [3,5,1,2,4]
  - Graph

# Combinatorial optimisation: The Knapsack problem

An example of an Optimisation Problem

**Thanks to:** Dr Steven Adriaensen
Free University of Brussels
Brussels, Belgium

- In Brussels a traveler named Tom is faced with a  problem.
- During his trip he has bought numerous souvenirs, varying in size and value.
- However, he has bought more souvenirs than fit into his luggage.
- Which souvenirs should he pack?



# Brussels attractions

The Atomium is a landmark building in Brussels, originally constructed for the 1958 Brussels World's Fair. It is now a museum.  A structure depicting atoms. There is a restaurant in the top sphere with a panoramic view over Brussels

Manneken Pis (little pee man in Flemish). The peeing boy is a small bronze fountain statue from the 17th century that is tall just 61cm (24 inches). Symbolise the good humour

# Knapsack problem

€ 45 (size: 4)

€ 15 (size: 2)


€ 500 (size: 1)

€ 20 (size: 2)


€ 40 (size: 4)

capacity: 6

€ 10 (size: 3)


Candidate solutions are different subsets of items Tom could pack and the objective is to find the one maximizing the total value of the packed items.

# Knapsack problem

€ 45 (size: 4)

€ 15 (size: 2)


€ 500 (size: 1)

€ 20 (size: 2)

€ 10 (size: 3)



€ 40 (size: 4)

capacity: 6

Candidate solutions are different subsets of items Tom could pack and the objective is to find the one maximizing the total value of the packed items.

# A possible algorithm for a small problem

Let us consider all the combinations that we can fit in rucksack


€ 20 (size: 2)

€ 40 (size: 4)


€ 500 (size: 1)

€ 20 (size: 2)

€ 10 (size: 3)


€ 500 (size: 1)

€ 20 (size: 2)

€ 15 (size: 2)


€ 20 (size: 2)

€ 45 (size: 4)

# A possible algorithm for a small problem


We could enumerate all 9 maximal subsets of items, determine the total value of their contents and return the most valuable.

€ 500 (size: 1)

€ 45 (size: 4)

# Heuristic optimisation

## Heuristic

- Describes how to derive an output for any given input (~ “ordinary” algorithm)
- Rule of thumb to solve a problem
- Provides no guarantees w.r.t. the quality (optimality) of the output.

## Why heuristics?

- Heuristics work well in practice!
- Heuristics trade *theoretical* guarantees on efficacy for: *practical* efficiency
- For some problems efficient exact algorithms are not known and unlikely to exist (e.g. NP-hard problems).

# Terminology and dates

- *Heuristic*: Greek word *heuriskein*, the art of discovering new strategies to solve problems
- Heuristics for solving optimization problems, G. Poyla (1945)
    - A method for helping in solving of a problem, commonly informal
    - "rules of thumb", educated guesses, or simply common sense
- Prefix *meta*: Greek for "upper level methodology"
- *Metaheuristics*:  term was introduced by Fred Glover (1986).
- Other terms: *modern heuristics*,  *heuristic optimisation, stochastic local search*

# Otpimisation and local search

- In many optimisation problems, the path to the goal is irrelevant; the goal state itself is the solution
- So we do not use tree-based search
  Search Space = set of all configurations or candidate solutions
- Find configuration or solution satisfying constraints, maximising or minimising a quality function
- In such cases, we can use local search algorithms also called metaheuristics
- Keep a single "current" state, try to improve it

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack<br>
2. repeat until no more items can be added:<br>
3.     determine $i_{\text{next}}$ the most valuable item that still fits.<br>
4.     add $i_{\text{next}}$ to the knapsack

**Greedy Construction [metaheuristic]:**

1. start with an empty solution<br>
2. repeat until solution is complete:<br>
3.     determine the solution component $c_{\text{next}}$ that can<br>
4.     be added to the partial solution at minimal cost.<br>
4.     add $c_{\text{next}}$ to the partial solution

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine $i_{\text{next}}$ the most valuable item that still fits.
4. add $i_{\text{next}}$ to the knapsack

€ 45 (size: 4)


€ 15 (size: 2)

€ 500 (size: 1)

€ 40 (size: 4)

remaining capacity: 6
value contents: €0

€ 10 (size: 3)

€ 20 (size: 2)

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine $i_{\text{next}}$ the most valuable item that still fits.
4. add $i_{\text{next}}$ to the knapsack

€ 45 (size: 4)


€ 15 (size: 2)

€ 40 (size: 4)

remaining capacity: 6
value contents: €0

€ 500 (size: 1)


€ 10 (size: 3)

€ 20 (size: 2)

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine $i_{\text{next}}$ the most valuable item that still fits.
4. add $i_{\text{next}}$ to the knapsack

€ 45 (size: 4)

€ 15 (size: 2)

€ 40 (size: 4)

remaining capacity: 5
value contents: €500

€ 10 (size: 3)

€ 20 (size: 2)

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine $i_{\text{next}}$ the most valuable item that still fits.
4. add $i_{\text{next}}$ to the knapsack

€ 45 (size: 4)


€ 15 (size: 2)

€ 40 (size: 4)

remaining capacity: 5
value contents: €500

€ 10 (size: 3)

€ 20 (size: 2)

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine $i_{\text{next}}$ the most valuable item that still fits.
4. add $i_{\text{next}}$ to the knapsack


€ 15 (size: 2)


€ 10 (size: 3)

€ 40 (size: 4)

remaining capacity: 1
value contents: €545

€ 20 (size: 2)

# Greedy construction heuristic

**Greedy Construction heuristic for Knapsack [heuristic]:**

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine $i_{\text{next}}$ the most valuable item that still fits.
4. add $i_{\text{next}}$ to the knapsack


€ 15 (size: 2)



€ 10 (size: 3)

€ 40 (size: 4)

remaining capacity: 1
value contents: €545

€ 20 (size: 2)

# Optimisation problems

General constrained optimisation problem:

Min/Max $f(x)$

Subject to:

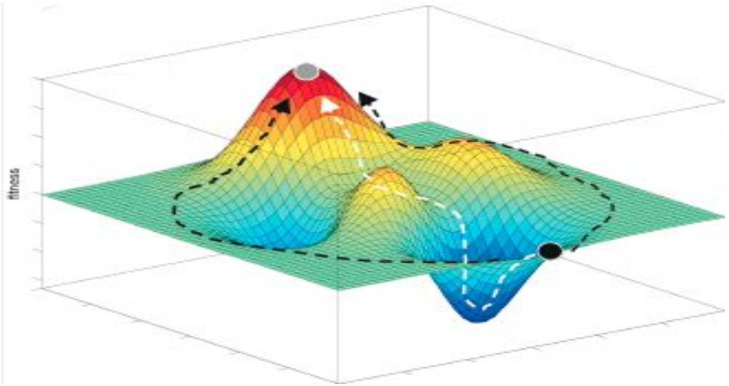
- Equality constraints
- Inequality constraints

**Search Space**: set of candidate solutions. All possible combinations of the decision variables.

**Optimisation through search**

Iteratively generate and evaluate candidate solutions.

- Systematic search
- (Stochastic) local search



# The knapsack problem

Given a knapsack of capacity *W*, and a number *n* of items, each with a *weight* and *value*. The objective is to maximise the total value of the items in the knapsack

Maximise $\sum_{i=1}^{n} v_i x_i$ Subject to $\sum_{i=1}^{n} w_i x_i \leqslant W, \quad x_i \in \{0,1\}$

- Search space size = $2^n$
- $n = 100$, $2^{100} \approx 10^{30}$

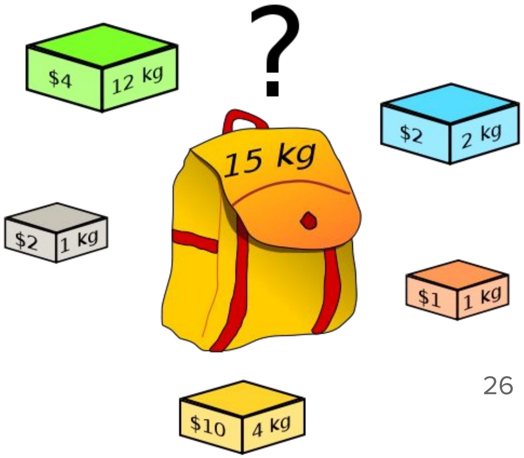
*maximise*

$$4x_1+2x_2+x_3+10x_4+2x_5 \qquad x_i = \begin{cases} 1 & \text{If we select item } i \\ 0 & \text{Otherwise} \end{cases}$$

*subject to*

$$12x_1+2x_2+x_3+4x_4+x_5 \leq 15$$

$$x_1,x_2,x_3,x_4,x_5 \in \{0, 1\}$$ Binary representation [11010]



# Example of small dataset and encoding as binary string

- Try out all possible ways of packing/leaving out the items
- For each way, it is easy to calculate the total weight carried and the total value carried
- Consider the following knapsack problem instance:

```
3
1  5  4
2 12 10
3  8  5
11
```

- Where: The first line gives the number of items. The last line gives the capacity of the knapsack. The remaining lines give the index, value and weight of each item.

# Knapsack, full enumeration

| Items | Value | Weight | Feasible? |
|---|---|---|---|
| 000 | 0 | 0 | Yes |
| 001 | 8 | 5 | Yes |
| 010 | 12 | 10 | Yes |
| 011 | 20 | 15 | No |
| 100 | 5 | 4 | Yes |
| 101 | 13 | 9 | Yes |
| 110 | 17 | 14 | No |
| 111 | 25 | 19 | No |

**Optimal!!**

# Real-world example of the Knapsack problem

- Consider a cargo company, that has an airplane and need to carry packages.
- Customers state the weight of the cargo item they would like delivered, and the amount they are prepared to pay.
- The airline is constrained by the total amount of weight the plane is allowed to carry.
- The company must choose a subset of the packages (bids) to carry in order to make the maximum possible profit, given the weight limit that they must respect.

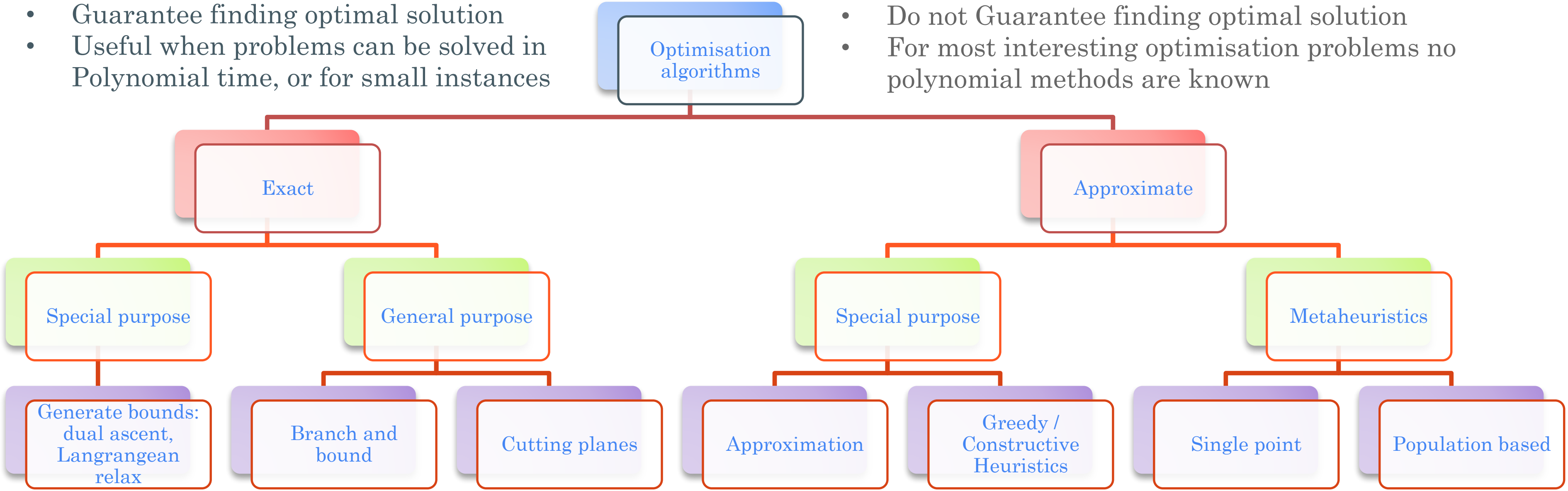
# Next Lab: Solving the Knapsack problem

- Read a file with the data describing an instance.. Two datasets will be provided with 20 and 200 items
- Solve the problems using two very different algorithms
  - A random search algorithm
  - A Greedy constructive heuristic
  - Optional: Full enumeration

# Optimisation/search algorithms

- Guarantee finding optimal solution
- Useful when problems can be solved in Polynomial time, or for small instances

- Do not Guarantee finding optimal solution
- For most interesting optimisation problems no polynomial methods are known




Approximation algorithms:

- An attempt to formalise heuristics (emerged from the field of theoretical computer science)
- Polynomial time heuristics that provide some sort of guarantee on the quality of the solution

# Summary

- Mayny real-world problems can be formulated as Search Problems
- We can distinguish between
  a. Searching for a sequence of actions or paths
  b. Searching for a solution in a large space of possible candidate solutions
- Optimisation problems are those where a quantity needs to be maximised or minimised
- They can be formulated as search problems
- Heuristics: are rules of thumb to solve problems