Modelling Optimization Problems for Evolutionary Algorithms and Other Metaheuristics

Carlos M. Fonseca

CISUC – Centre for Informatics and Systems

Department of Informatics Engineering

Faculty of Science and Technology

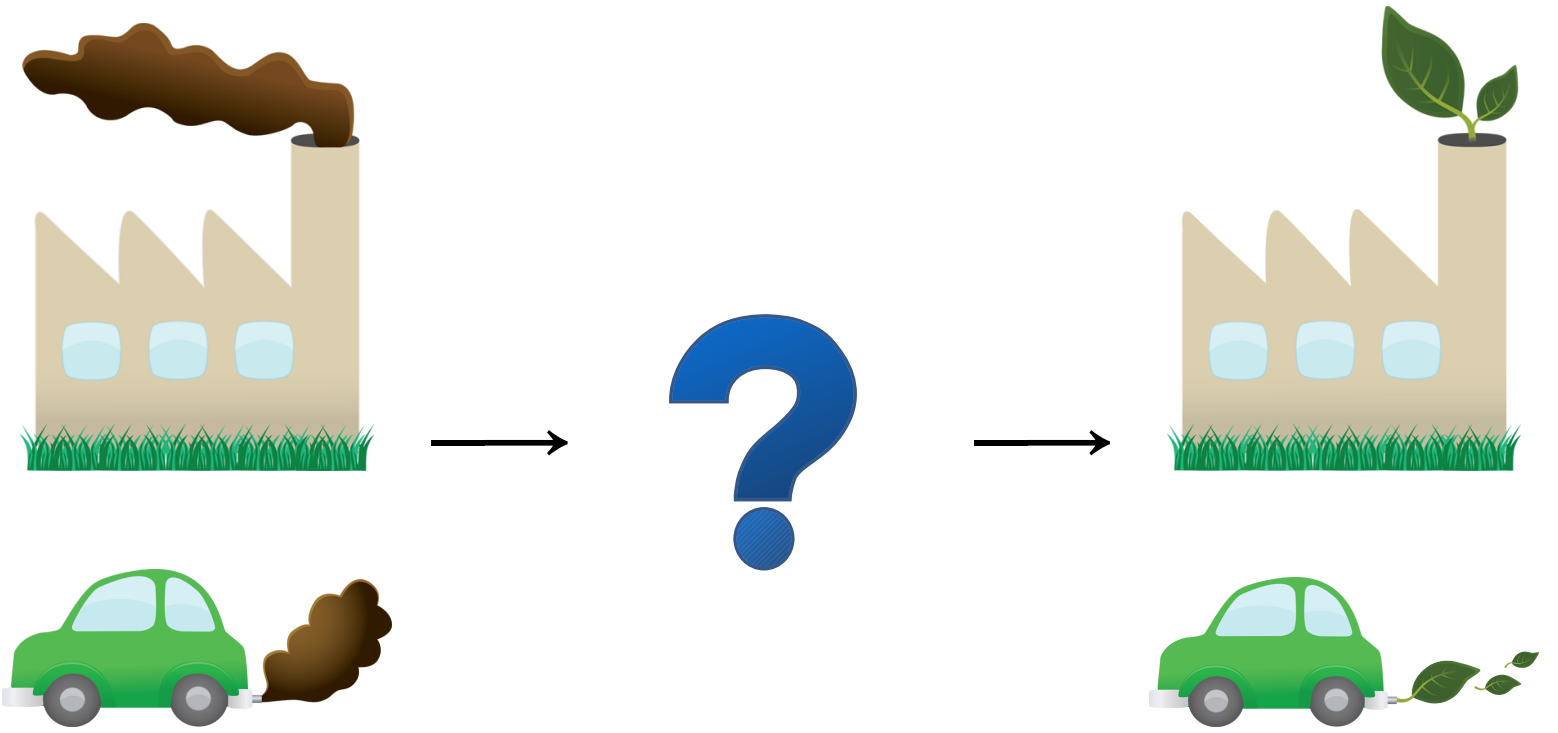
University of Coimbra

Portugal

Outline

- Problem solving
- Optimization
- Modelling
- Modelling how-to
- Worked examples
- Concluding remarks

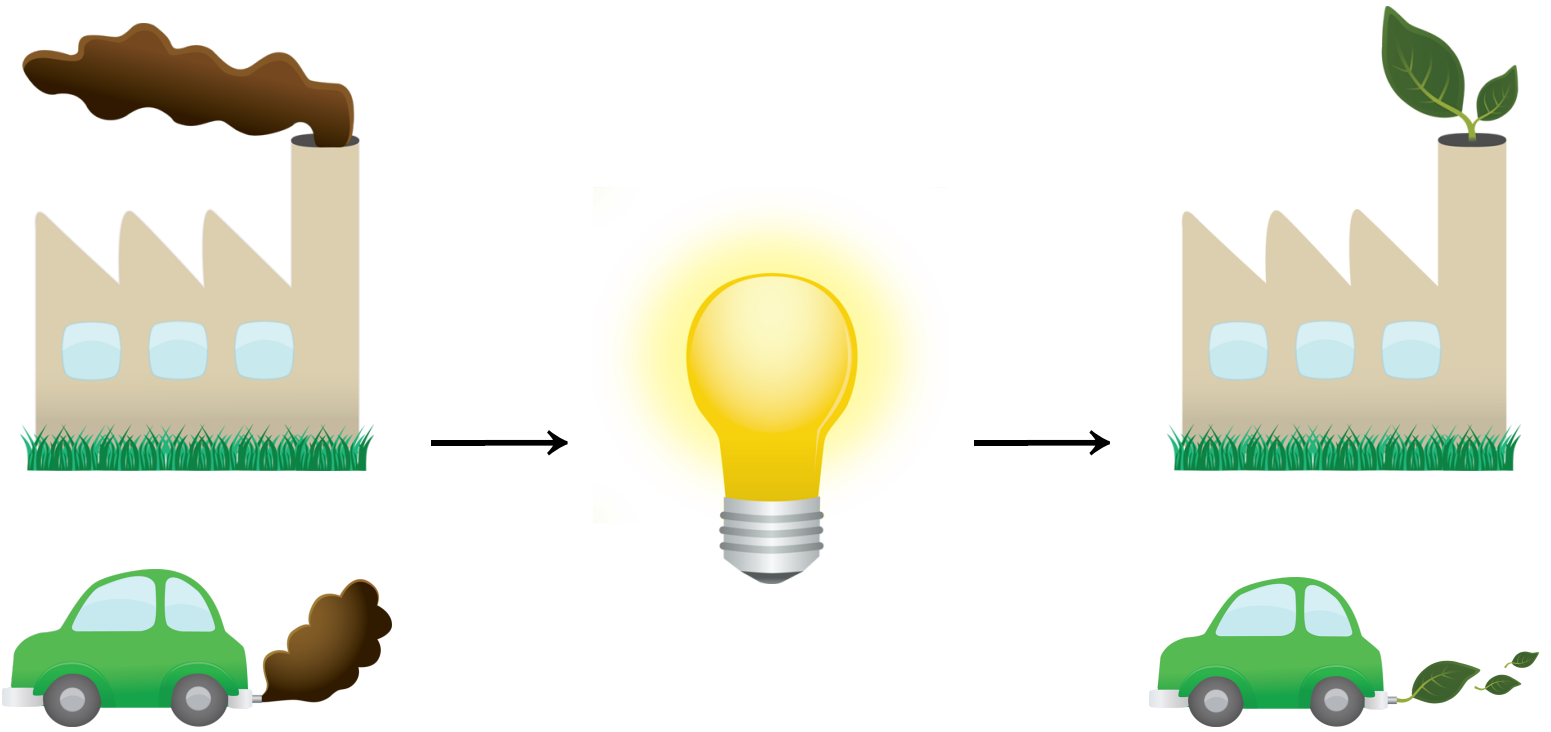
Problem solving



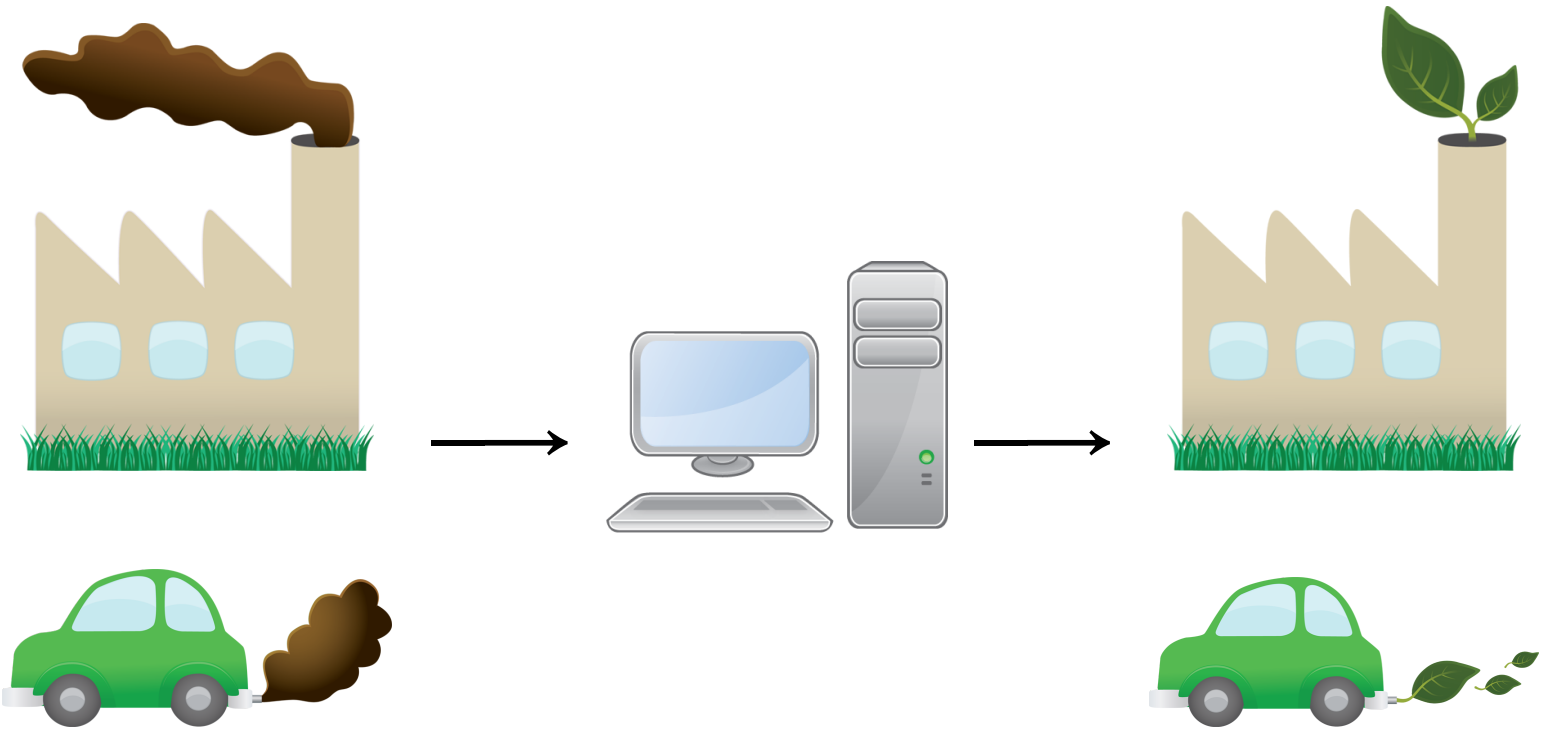
Problem

Solution

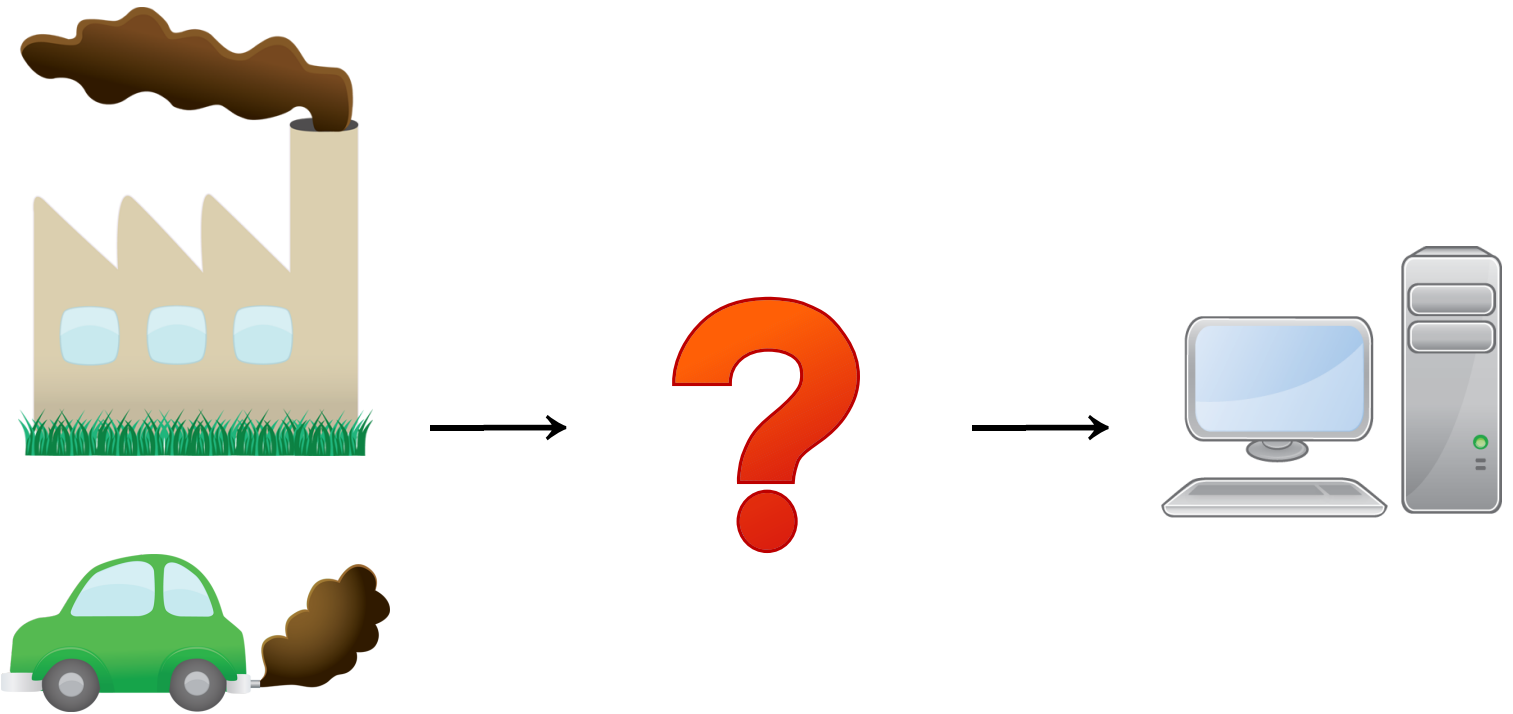
Problem solving



Problem solving



Problem solving



Problem

Computer

Problem solving

3D printing service

An additive manufacturing startup provides 3D printing services to private and business customers.

- Single metal 3D printer
- Parts must be printed without interruption
- Part-dependent printing time
- Printing deadlines, penalties proportional to how late parts are completed

Q: In what order should a set of parts be printed so as to minimize the total penalty?

Problem solving

Urban waste collection

Waste collection vehicles are used to regularly empty public waste containers located on the side of roads (away from intersections) across Neat City.

- Single vehicle serves one of the city quarters, large enough to empty all containers in a single tour
- Vehicle leaves the depot, visits all containers, disposes of the collected waste at the city's waste treatment plant, and returns to the depot
- One-way and two-way roads, emptying containers on either side of the road not always possible
- No U-turns allowed

Q: What is the shortest tour, subject to the above constraints?

Problem solving

Campus network

At a University campus, each building is to be connected directly to the computing centre by a single, dedicated high-speed cable.

- Trenches dug between buildings, used to lay cables
- Possibly more than one cable per trench
- Trench costs and cable costs both proportional to length

Q: What trenches to dig so as to minimize the total (trench and cable) set up cost of the new network?

Problem solving

Community detection

A fully-connected undirected graph, where vertices represent users and weighted edges represent the intensity of some attribute of their interaction that may be positive or negative, was obtained from social network data.

- Users connected by edges with positive weight show affinity to each other, whereas negative edge weights indicate lack of affinity
- Groups of users connected mostly by positively weighted edges suggest the existence of a community involving those individuals

Q: How should the individuals be grouped so as to maximize the total internal edge weight of all groups?

Problem solving

Candle race

There is a number of villages to be visited, and there is a candle in each village, except for the village where the race starts. At the beginning of the race, the candles are lit. The player's goal is to visit each village and blow out the candles. When a candle is blown out, the player scores points proportionally to the remaining length of the candle.

- Each candle has an initial length and a burning rate. As time passes, the length is reduced at the given rate until it eventually reaches 0
- The travel time between each pair of villages is known

Q: In which order should the villages be visited so as to maximize the score?

Problem solving

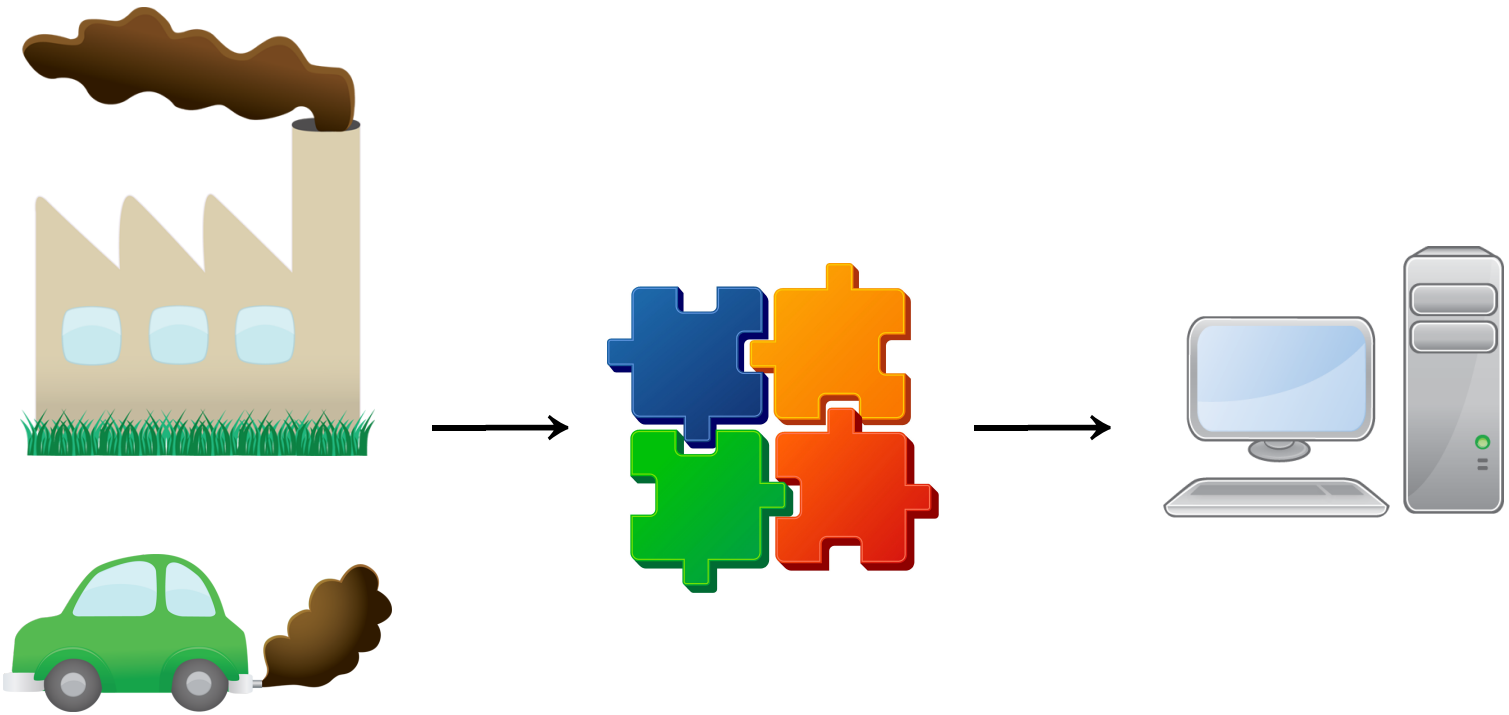
Laptop assembly

A laptop manufacturer pre-assembles different laptop models of several base designs in order to better meet customer demand. For a given base design, there is a number of different laptop models, and assembling one laptop unit of a given model requires certain numbers of parts of different types.

- Laptop units are assembled in sequence and take one time slot each
- The number of parts of each type required by each model and the numbers of units of each model to be assembled on a given day are known

Q: In which order should the various units of the different models be assembled in order to achieve a smooth flow of parts to the cell?

Problem solving

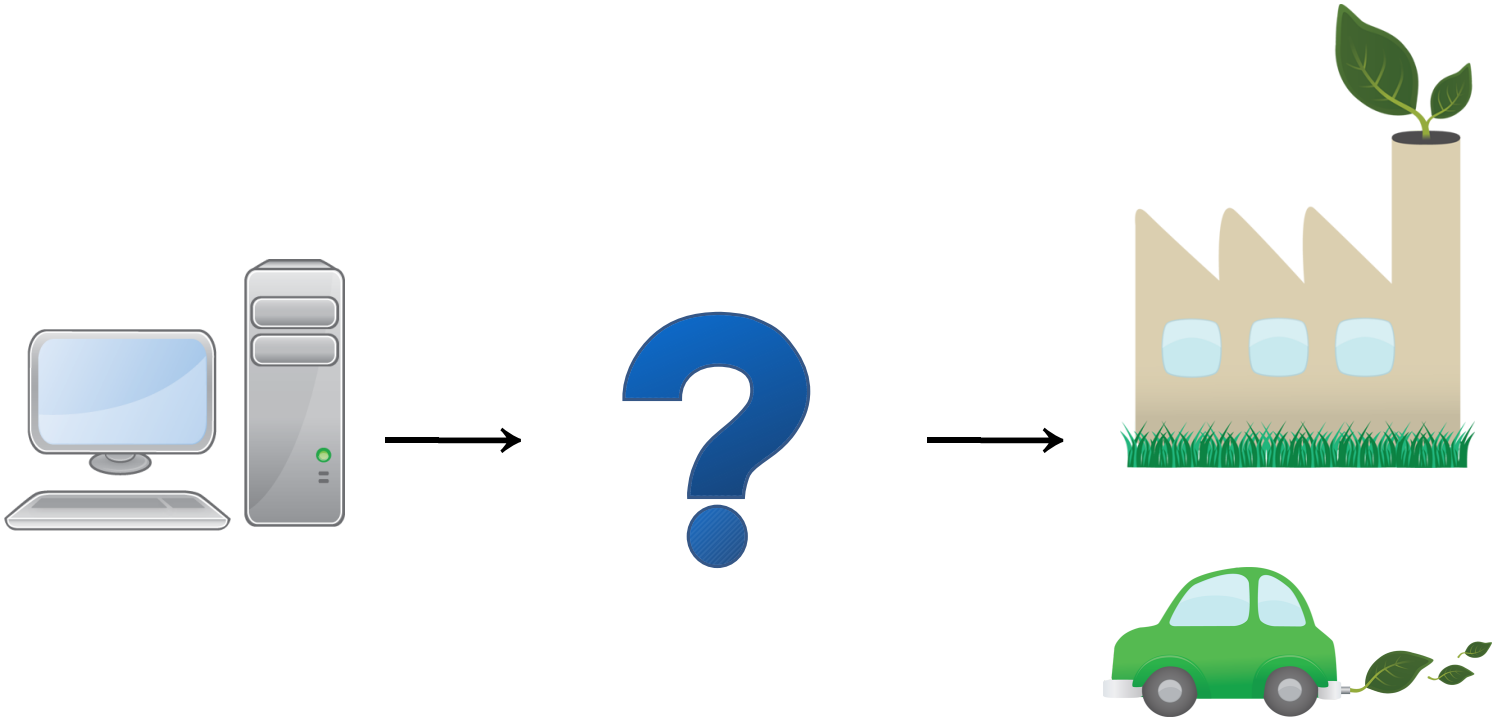


Problem

Model

Computer

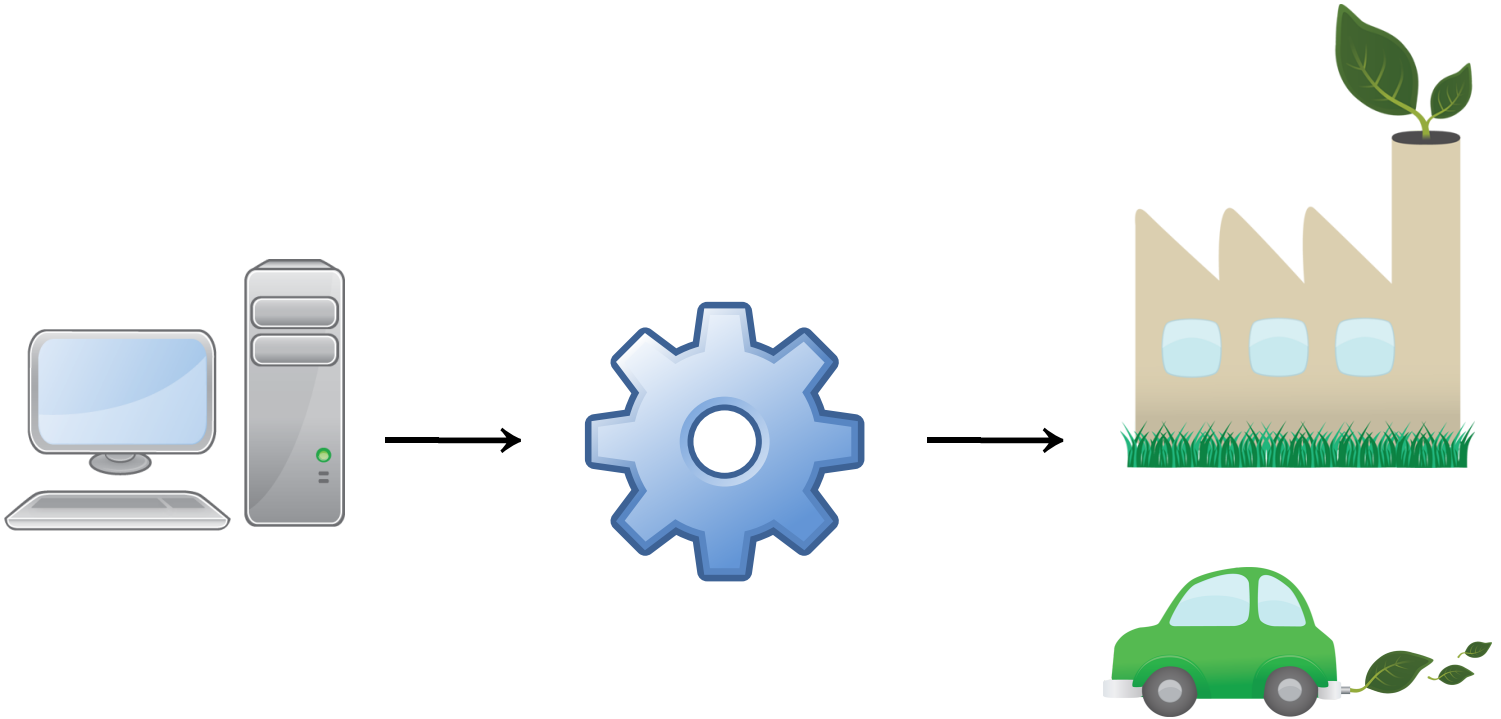
Problem solving



Computer

Solution

Problem solving



Optimization

Real-world problems are typically described in natural language first

- Qualitative, imprecise, ambiguous, incomplete, ...

Many such problems involve the idea of doing *well*, doing *better* than before, or even the *best* possible

- Optimization problems!

To solve an optimization problem on a computer, a formal characterization is required, including:

- A specification of the space of alternative solutions (solution space, or *decision* space)
- A means of evaluating the performance of those solutions
- The data that specifies the particular problem *instance* of interest

Optimization

Problem classification

Depending on the nature of the decision space, optimization problems are usually classified as:

Numerical problems Solutions are numbers, or vectors with numeric components, and solution performance is quantified in terms of an objective function (or functions)

Continuous Solutions range over a continuous set, such as a continuous subset of $\mathbb{R}^n$

Discrete Solutions range over a discrete set, such as a possibly infinite subset of $\mathbb{Z}^n$

Mixed Solutions are vectors with both continuous and discrete numeric components

Combinatorial problems Solutions are elements of a finite, discrete set, and can be interpreted as subsets of a so-called *ground set* of solution components

▷ Combinatorial optimization problems can usually be posed as discrete numerical problems

Optimization

Problem solving strategies

The choice of solution strategy depends on the nature and type of the optimization problem at hand, but also on how much information about the problem is available to the solver:

Glass box The mathematical formulation of the problem is fully available to, and can be directly manipulated by, the solver. The simplex method and most mixed integer-linear and constraint programming solvers are of this type

Black box The mathematical formulation of the problem is *not* available to the solver, which is restricted to evaluating the objective function at discrete points in the solution space. Higher-order black-box methods involve evaluating objective function derivatives at given solutions, as well.

Optimization

Black-box problem solving strategies (for combinatorial optimization)

Random search Guess! However, ...

- Even generating a feasible solution may be hard

Constructive search Build a solution from scratch by starting with an empty solution and successively adding components to it

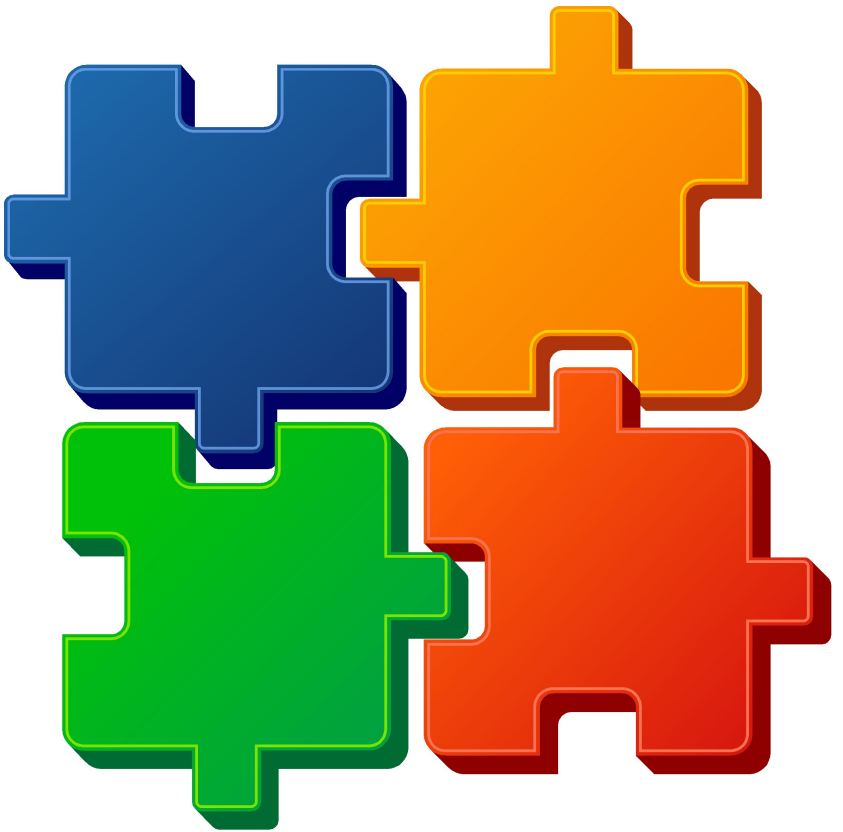
- A *partial* solution represents all *feasible* solutions that can be constructed from it

Local search Try to improve an existing (feasible) solution by modifying it

- Small changes to a solution should lead to small changes to its quality

Hybrid / other ...

Modelling



Modelling

Modelling is the *process* leading from

- a *natural language description* of a real-world problem

to

- a *mathematical formulation*

that can be *analyzed* and/or *solved* using mathematical or computational tools

▷ The model depends on the problem *and* on the type of solver

▷ Solvers applied to different models of the same (global) optimization problem are actually operating on different (local-search, constructive-search) problems!

▷ Not all models are equally easy to solve

Modelling

Problem-modelling approaches

Decomposition Decompose the original problem into subproblems that can be solved separately

Proxy function Consider a *related* objective function that can be evaluated or optimized more easily

- May miss the optimum of the original problem

Restriction *Narrow down* the decision space

- Solutions may become easier to find
- May miss the optimum of the original problem

Relaxation *Enlarge* the decision space allowing it to contain (some) infeasible solutions

- Solutions to the relaxed problem may become easier to find
- Solutions found may need to be “corrected” or “repaired”

Modelling

Exact and heuristic optimization algorithms

Constructive search Backtracking, dynamic programming, branch and bound, ant colony optimization, beam search, GRASP, ...

Local search Greedy/stochastic descent, evolutionary algorithms, memetic algorithms, particle-swarm optimization, differential evolution, simulated annealing, tabu search, iterated local search, GRASP, ...

- ▷ What defines the neighbourhood explored by, for example, an evolutionary algorithm? What about crossover???
- ▷ What (local) optimization problem do constructive-search algorithms actually solve?

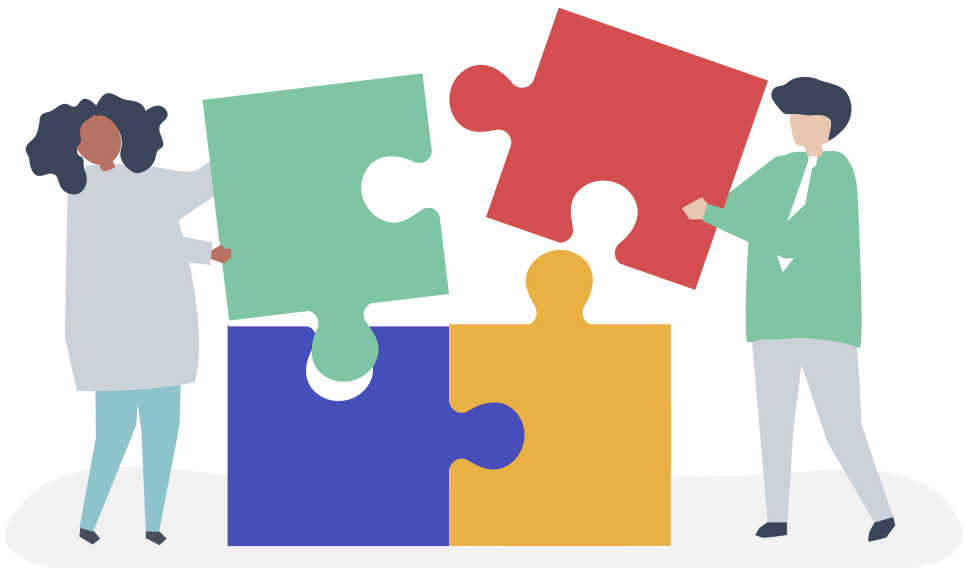
Modelling

Vision

- Problem modelling fully separated from black-box solver specification
- End users oblivious to the inner workings of the solvers
- Underlying modelling paradigm, training
- Built-in performance features
- The same model for many solvers
- Solvers transparently operate on all problems of the appropriate type

▷ How?

Modelling how-to



Designed by rawpixel.com / Freepik

Modelling how-to

Modelling a combinatorial optimization problem as a *constructive* or *local* search problem begins with asking and answering questions

Problem instance What (known) data is required to fully characterize an *instance* of the problem?

Solution What (unknown) data is required to fully characterize a (feasible) *solution*?

Objective function How can the performance of a given candidate solution be measured? Is the corresponding value to be minimized or maximized?

Neighbourhood structure (Local Search) What are *similar* solutions?

- "Parts" of the two solutions are somehow identical
- Similar performance (in most cases)
- Connect the whole space

Modelling how-to

Combinatorial structure (Constructive search) What is a *partial* or *incomplete solution*?

- Solutions as subsets of a larger *ground set* of solution *components*
- Partial solutions as a representation of *all feasible solutions* that contain them
- Not all subsets of components are valid (partial) solutions
  - ▷ *Construction rule*
- Performance of partial solutions inferred from the sets of solutions which they represent
  - ▷ *Lower bound* (minimization) or *upper bound* (maximization)

Modelling how-to

Computational model

Problem instance representation How can the problem instance data be stored in a data structure so that the objective function and corresponding bounds can be easily computed?

Solution representation How can (possibly partial) solutions be represented so that

- Objective function values (where applicable) and related bounds can be computed efficiently?
- **[Constructive search]** Feasible solutions can be easily constructed by successively adding components?
- **[Local search]** Solutions can be easily modified to obtain neighbouring solutions?

Solution evaluation How can the objective function and/or corresponding bounds be computed given the instance data and the solution representation?

Modelling how-to

Move representation How can *moves* be represented, i.e.,

[Constructive search] the *addition* or *removal* of components to/from a (partial) solution?

[Local search] changes that, when applied to a solution, lead to a neighbouring solution?

Solution modification What are *valid* moves, and how are they applied to a solution?

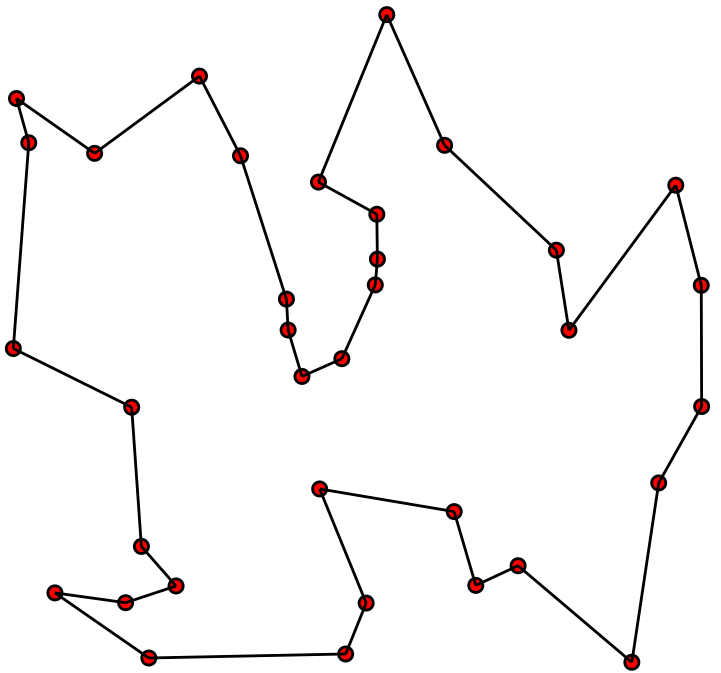
Incremental solution evaluation When an evaluated (partial) solution is modified by applying one or more moves to it, can the resulting solution be evaluated faster? How?

Move evaluation How would applying a move to a solution change its performance? Which is faster:

- Evaluating the move without actually applying it, or
- Evaluating the original solution, applying the move, and then evaluating the result?

Worked example – Constructive search

The travelling salesman problem



Source: Xypron

Worked example – Constructive search

The travelling salesman problem

Find the shortest route that visits each city in a list exactly once and returns to the initial city (a tour)

- Classical NP-hard optimization problem, commonly referred to as the TSP

Problem instance What (known) data is required to fully characterize an instance of the problem?

A: The distances from each city to every other city

Solution What (unknown) data is required to fully characterize a (feasible) solution?

A1: The list of cities in the order that they should be visited

A2: The *set* of arcs connecting the cities in the order that they should be visited

Worked example – Constructive search

The travelling salesman problem

Objective function How can the performance of a given candidate solution be measured? Is the corresponding value to be minimized or maximized?

A: The total length of the tour, to be minimized

Construction What is a partial solution?

A1: List of the first cities to visit in the order that they should be visited, or the corresponding subset of arcs

A2: *Any* subset of arcs that can be part of a tour

Worked example – Constructive search

The travelling salesman problem

Problem instance representation How can the problem instance data be stored in a data structure so that the objective function and corresponding bounds can be easily computed?

A: An $N \times N$ distance matrix, where N denotes the number of cities

Solution representation How can solutions be represented?

A1: A sequence of k distinct integers between 0 and $N-1$ (cities), where $1 \leq k \leq N$

A2: A list of up to N pairs of integers (arcs)

Which one should be better? Why?

Worked example – Constructive search

The travelling salesman problem

Solution evaluation How can the objective function and/or corresponding bounds be computed given the instance data and the solution representation?

A1: For each pair of cities to be visited immediately after one another, add the length of the corresponding arcs

- ▷ Weak lower bound on the length of any tour that can be obtained by adding components (arcs) to a partial solution

A2: Add *also* half the length of the shortest arc emanating from each city connected only to one other city *and* the average length of the two shortest arcs from each yet unconnected city

- ▷ Stronger lower bound

Worked example – Constructive search

The travelling salesman problem

Move representation How can moves, i.e., the *addition* or *removal* of components to/from a solution, be represented?

A: A pair of integers, (i, j), representing the arc to be added or removed

Solution modification What are valid moves, and how are they applied to a solution?

A1: To *add*, append j to the (partial) sequence of cities to be visited if the last city is i. To *remove*, drop the last city if the sequence ends in i, j.

A2: Add or remove the arc (i, j) to/from the solution arc list, provided that there is at most one ingoing arc and one outgoing arc for each city and no subtours are introduced

Worked example – Constructive search

The travelling salesman problem

Incremental solution evaluation When an evaluated solution is modified by applying one or more moves to it, can the resulting solution be evaluated faster? How?

A1: Yes, just add or subtract the length of the arc(s) that were added or removed, respectively

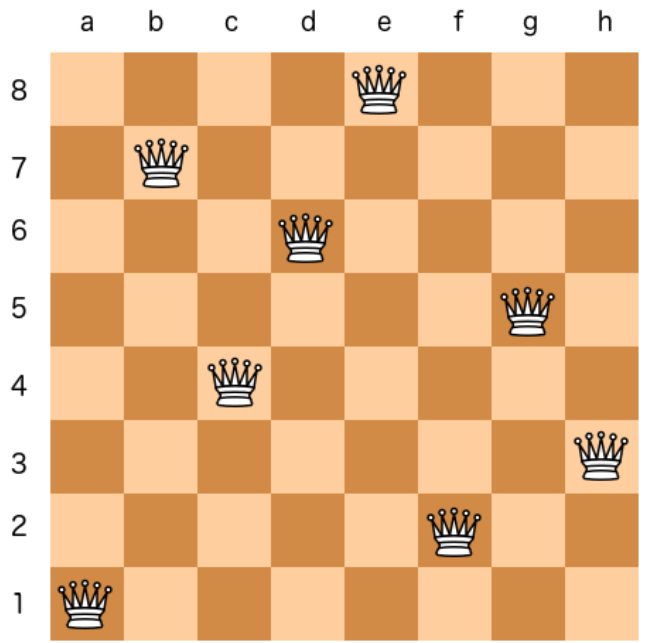
A2: As above, but must also subtract or add the length of other arcs not in the solution

Move evaluation How much would applying a given move to a solution change its performance? Which is faster, evaluating the move without actually applying it, or evaluating the original solution, applying the move, and evaluating the result?

A1: Evaluating the move without actually applying it should be faster. No need to modify or even evaluate the original solution

Worked example – Local search

The N-Queens problem



Source: Yue Guo

Worked example – Local search

The N-Queens problem

Place N chess queens on a $N \times N$ chess board so that no two queens threaten each other

- Simple but non trivial problem solvable in polynomial time

Problem instance What (known) data is required to fully characterize an *instance* of the problem?

A: The size of the board, N

Solution What (unknown) data is required to fully characterize a (feasible) *solution*?

A: The locations $(i, j) \in \{0, \ldots, N-1\}^2$ of all queens on the board

Worked example – Local search

The N-Queens problem

Objective function How can the performance of a given candidate solution be measured? Is the corresponding value to be minimized or maximized?

A1: Number of queens threatened by at least another queen, to be minimized

A2: Number of pairs of queens that threaten each other, to be minimized

Which one should be better? Why?

Neighbourhood structure What are *similar* solutions?

A: Solutions differing in the position of a single queen (or a small number of queens)

Moving a single queen adds or subtracts *at most* $N-1$ to/from the objective value

Worked example – Local search

The N-Queens problem

Problem instance representation How can the problem instance data be stored in a data structure so that the objective function can be easily computed?

A: It is just an integer, N

Solution representation How can solutions be represented?

A1: A sequence of pairs of integers between 0 and $N-1$

A2: A permutation of the integers $0,\ldots,N-1$, where each position i represents the row and the value j at position i represents the column where queen i is placed

Which one should be better? Why?

Worked example – Local search

The N-Queens problem

Solution evaluation How can the objective function be computed given the instance data and the solution representation?

A1: Set a counter to 0. Iterate over all pairs of queens, and let (i, j) and (k, ℓ) denote their positions on the board. For each pair, increment the counter if

$$i = k \text{ or } j = \ell \text{ or } |i - k| = |j - \ell|$$

A2: Iterate over all pairs of queens, as before. For each pair, increment the counter if

$$|i - k| = |j - \ell|$$

Worked example – Local search

The N-Queens problem

Move representation How can *moves*, i.e., changes that, when applied to a solution, lead to a neighbouring solution, be represented?

A2: A pair of integers, (i, k), representing the rows where two queens are located

Solution modification How is applying a *move* to a solution performed?

A2: Swap the permutation values, j and ℓ, at permutation positions i and k, respectively. The new location of the queens is (i, ℓ) and (k, j)

Worked example – Local search

The N-Queens problem

Incremental solution evaluation When an evaluated solution is modified by applying one or more moves to it, can the resulting solution be evaluated faster? How?

A2: Yes. Iterate over the $2 \times (N-2)$ pairs of queens containing one of the queens moved, first in the original position, and then in the final position, and update the objective value accordingly

Move evaluation How much would applying a given move to a solution change its performance? Which is faster, evaluating the move without actually applying it, or evaluating the original solution, applying the move, and evaluating the result?

A2: Evaluating the move without actually applying it is faster. No need to evaluate the original solution

Concluding remarks

- Modelling before solving!
- Different types of solvers require different models
- Modelling paradigms
- Modelling how-to, with examples
- Six modelling projects to be carried out in groups during the School