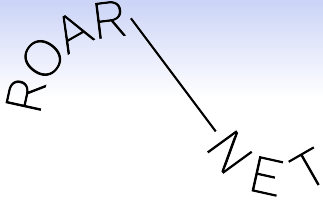


Optimisation and Modelling

First ROAR-NET Training School

Carlos M. Fonseca
University of Coimbra
Portugal

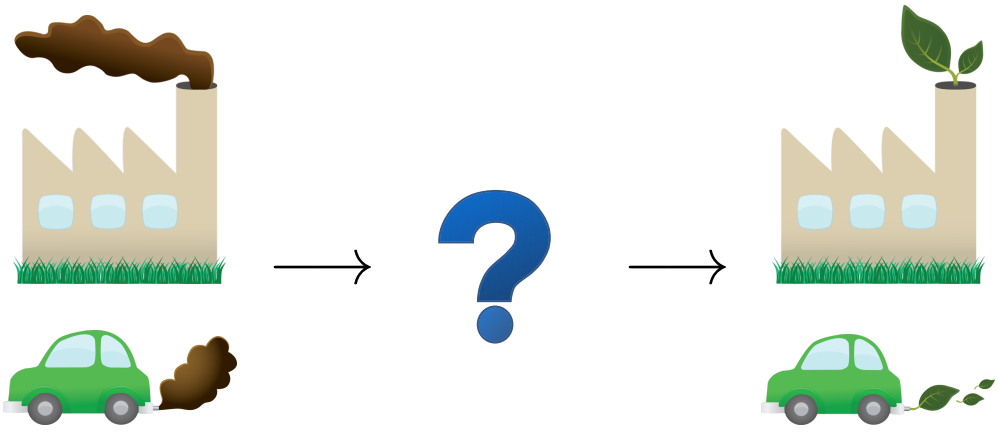
cost
EUROPEAN COOPERATION IN SCIENCE & TECHNOLOGY

Funded by the European Union

Outline

- Problem solving
- Optimisation
- Metaheuristics
- Modelling
- Constructive search
- Local search
- Worked examples
- Concluding remarks

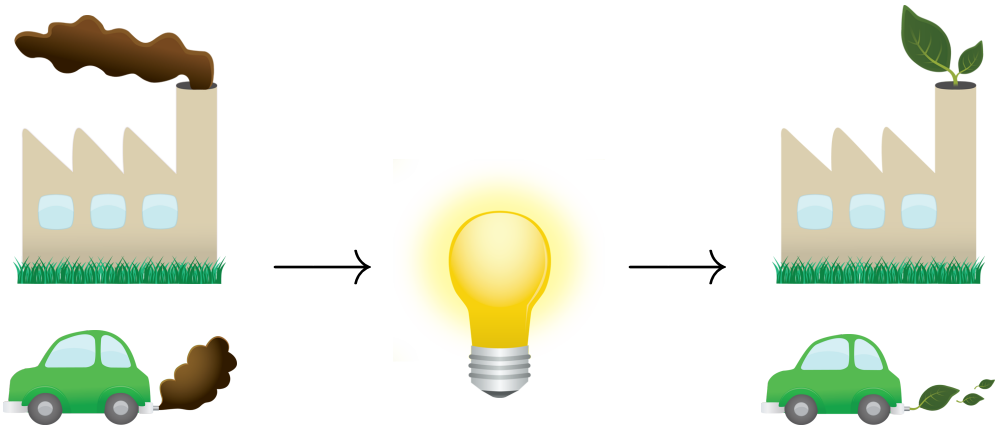
Problem solving



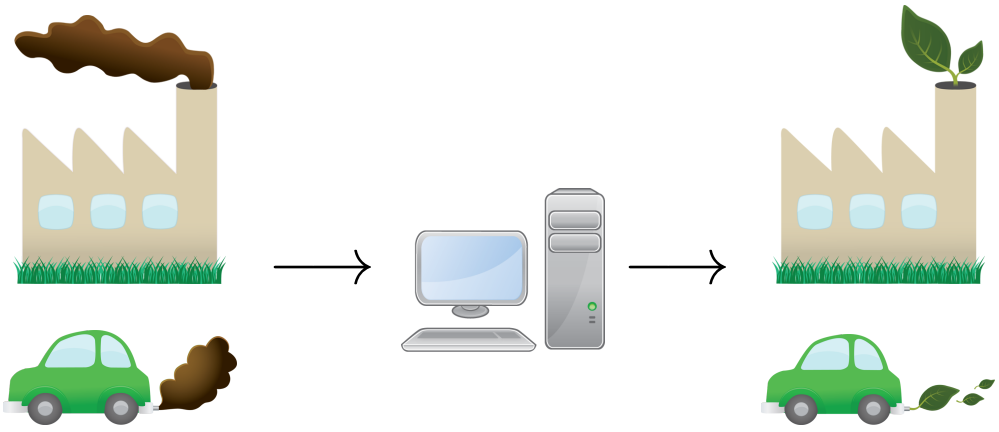
Problem

Solution

Problem solving



Problem solving



Optimisation

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

- Optimisation problems!

To solve an optimisation problem on a computer, a formal characterisation 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

$\Longrightarrow$ Problem statement!

Optimisation

Depending on the nature of the decision space, optimisation 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 *ground set* of solution components

- ▷ Combinatorial optimisation problems can usually be posed as discrete numerical problems

Problem solving strategies

The choice of solution strategy depends on the nature and type of the optimisation 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.

Optimisation

Black-box problem-solving strategies (for combinatorial optimisation)

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 …

Optimisation

ROAR NET

Exact and heuristic optimisation algorithms

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

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

- ▷ But are evolutionary algorithms not *global* optimisation methods? What about crossover???
- ▷ What neighbourhood does an evolutionary algorithm explore?
- ▷ What (local) optimisation problem do constructive-search algorithms actually solve?

Metaheuristics

"... metaheuristics are algorithms that combine heuristics (that are usually very problem-specific) in a more general framework." (Bianchi *et al.*, 2009)

Evolutionary algorithms

- Inspired by the process of natural selection and genetic evolution
- Emphasis on solution representation
- Problem-specific search operators
  - Recombination (crossover)
  - Mutation

Metaheuristics

Simulated annealing

- Inspired in the physical process of cooling a material down slowly to improve its properties
- First-improvement local search with probabilistic degradation acceptance

Tabu search

- First-improvement local search with least-degradation acceptance in case of no improvement
- Problem-specific short-term memory mechanism to avoid looping
- Problem-specific long-term memory (construction) mechanism to seed the local search

Metaheuristics

ROAR NET

Iterated local search

- Strategy designed to exploit the big-valley hypothesis
- Local search in the space of local optima
- Problem-specific perturbation mechanism

Beam search

- Breadth-first constructive search with breadth constraint (beam width, β)
- Best β nodes retained at each level
- $\beta = 1$ amounts to greedy construction

Metaheuristics

Ant colony optimisation

- Inspired in the foraging behaviour of ants
- Solution construction based on a pheromone-based communication mechanism and heuristic information
- Construction typically followed by local search

Greedy Randomised Adaptive Search Procedure (GRASP)

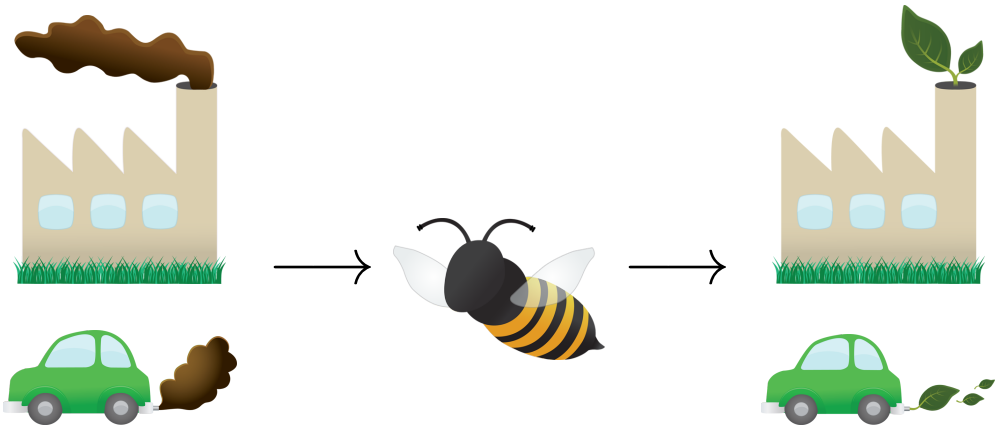
- “Soft” greedy randomised construction
- Typically followed by local search

Metaheuristics

Common practice

- When addressing a new problem
  - Choose favourite family of metaheuristics
  - Design (new?) problem-specific search operators
  - Produce a “new algorithm” for that problem
- Most metaheuristic software frameworks support this view
- Limitations
  - End users must also be familiar with the algorithms
  - Problem-specific developments tied to a particular algorithm

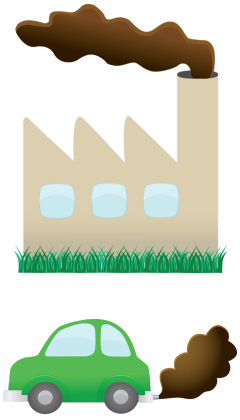
Problem solving with metaheuristics (so far)



Problem solving with metaheuristics (vision)

- General-purpose *black-box solvers* rather then dedicated algorithms
- Problem modelling fully separated from 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

Problem solving with metaheuristics (vision)

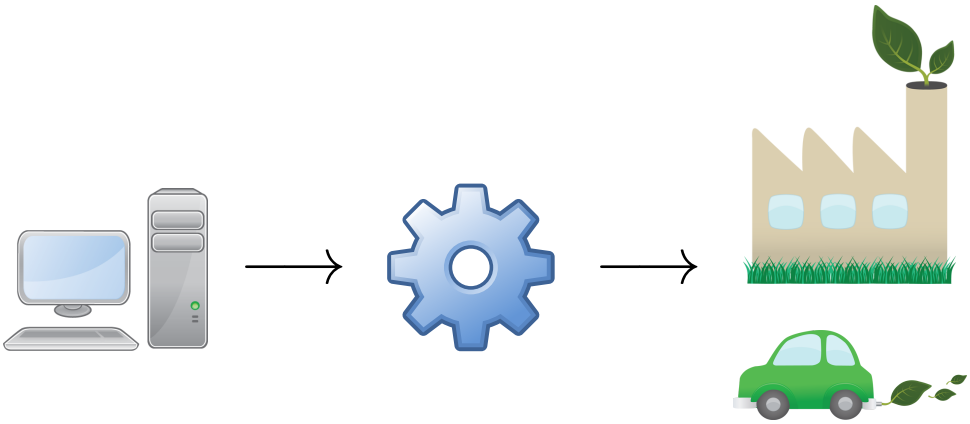


Problem

Model

Computer

Problem solving with metaheuristics (vision)



Problem solving with metaheuristics (vision)

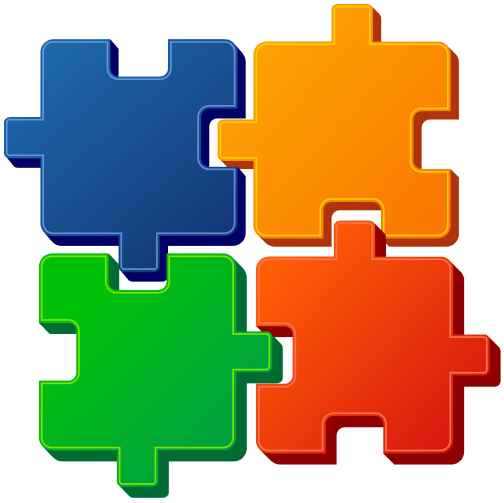
Why?

- The value of optimisation is realised at the user end
- End users care about finding solutions
- Modelling empowers users to solve *their own* problems
- Metaheuristic solvers as software products
- Solvers are only as good as the solutions they produce and how quickly they produce them
- General-purpose solver development as a new avenue for research

Problem solving with metaheuristics (vision)

How?

Modelling



Modelling

Modelling is the *process* leading from

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

  to

- a *mathematical formulation*

that can be *analysed* 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) optimisation problem are actually operating on different (local-search, constructive-search) problems!
- ▷ Not all models of the same (global) optimisation problem 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 optimised 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”

Constructive search

Modelling a combinatorial optimisation problem as a *constructive* search problem begins with asking and answering questions

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

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

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

Constructive search

Combinatorial structure 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* (minimisation) or *upper bound* (maximisation)

Constructive search

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?
- Feasible solutions can be easily constructed by successively adding components?

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

Constructive search

Computational model

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

Local search

Modelling a combinatorial optimisation problem as a *local* search problem also begins with asking and answering questions

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

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

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

Local search

Neighbourhood structure What are *similar* solutions?

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

Example 1 Symmetric Travelling Salesman Problem

- Tour length is the sum of the lengths of the tour edges
- Solutions are similar if they differ in a small number of edges
- 2-opt and 3-opt moves

Local search

ROAR NET

Example 2

Asymmetric Travelling Salesman Problem

- Tour length is the sum of the lengths of the tour arcs
- Solutions are similar if they differ in a small number of arcs
- Due to asymmetry, 2-opt moves and some 3-opt moves are not suitable because they reverse parts of the tour
- Insertion move as a special case of 3-opt

Local search

Computational model

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

Solution representation How can solutions be represented so that

- Objective function values can be computed efficiently?
- Solutions can be easily modified to obtain neighbouring solutions?

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

Local search

Computational model

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

Local search

Neighbourhood exploration

Local search in a nutshell

1. Visit neighbours of a current solution
2. Decide whether to reject them or to accept one as the next solution
3. Repeat

Local move generation

- Enumeration
- Random sampling with replacement
- Random sampling without replacement

Local search

ROAR NET

Local move enumeration

- Full neighbourhood exploration
- Enumeration order dictated by convenience
- Filter out invalid moves if needed
- Neighbourhood size may not be known in advance

Random sampling with replacement

- Sampling uniformly at random typically preferred
- Rejection sampling useful when neighbourhood size not known in advance

Local search

ROAR NET

Random sampling without replacement

- Partial to full neighbourhood exploration
- Random enumeration order to avoid search bias
- Filter out invalid moves if needed (rejection)
- Neighbourhood size may not be known in advance
- Generating (pseudo-)random permutations
  - Fisher-Yates shuffle
  - Linear congruential generator (low quality but constant space)
  - A plethora of other approaches

Concluding remarks

- Modelling before solving!
- Different types of solvers require different models
- Metaheuristics usually incorporate both a model and a search algorithm
- Modelling abstractions
- Language-independent modelling API specification
- API implementations in different languages
- ROAR-NET is a community-building effort

Acknowledgments

This presentation is based upon work from COST Action Randomised Optimisation Algorithms Research Network (ROAR-NET), CA22137, supported by COST (European Cooperation in Science and Technology).

This work is funded by national funds through FCT – Foundation for Science and Technology, I.P., within the scope of the research unit UID/00326 – Centre for Informatics and Systems of the University of Coimbra