Introduction to Constructive Search

Carlos M. Fonseca

University of Coimbra

Outline

- Modelling
- Decision space
- Solution structure
- Construction rules
- Solution quality
- Bounds

Modelling

Modelling a combinatorial optimization problem as a *constructive* search problem begins with asking and answering a few 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?

Modelling

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* (minimization) or *upper bound* (maximization)

Decision space

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

- A possible representation of the feasible solutions is given by the solution output format in the problem description
- In general, such a representation is not unique
- It is often redundant: the same solution may be output in different ways

⚠ The solution output format may or may not be the most appropriate representation to support the solving process!

The *decision space* is the *set of all candidate solutions* that may be *visited* during the solving process

- Both *feasible* and *infeasible* solutions may be included
- Infeasible solutions have *no* objective value

Decision space

Example

Supose that an agent is placed on a maze defined on a rectangular grid and can travel the maze by making a sequence of single steps N, S, E, W. Banging on walls is not allowed. Each cell provides either a reward or a penalty and can be visited only once. Starting from a given cell, $(5,5)$, the goal is to collect as much reward as possible and then stop.

- A given solution might be represented as a sequence of moves, such as

$$(N, E, E, S, S, W, W, W, S, W, W)$$

- Alternatively, it might be represented as a sequence of visited cells

$$[(5,5), (6,5), (6,6), (6,7), (5,7), (4,7), (4,6), (4,5), (4,4), (3,4), (3,3), (3,2)]$$

Decision space

Other alternatives

- A set of visited cells?

$$\{(5,5),(6,5),(6,6),(6,7),(5,7),(4,7),(4,6),(4,5),(4,4),(3,4),(3,3),(3,2)\}$$

- A set of position-cell pairs?

$$\{(0,(5,5)),(1,(6,5)),(2,(6,6)),(3,(6,7)),(4,(5,7)),(5,(4,7)),(6,(4,6)),(7,(4,5)),(8,(4,4)),$$
$$(9,(3,4)),(10,(3,3)),(11,(3,2))\}$$

- A set of cell pairs, or a subset of arcs of the directed graph representing the moves allowed?

$$\{((5,5),(6,5)),((6,5),(6,6)),((6,6),(6,7)),((6,7),(5,7)),((5,7),(4,7)),((4,7),(4,6)),$$
$$((4,6),(4,5)),((4,5),(4,4)),((4,4),(3,4)),((3,4),(3,3)),((3,3),(3,2))\}$$

Decision space

Discussion

- The step-sequence representation is more compact, but the same N, S, E, or W step has a different effect on the score depending on its position on the sequence
- In contrast, in the cell-sequence representation, the contribution of a given cell to the score does not depend on where in the sequence it appears
- The cell-set may or may not uniquely define the path
- The position-cell pair representation uniquely defines the path, but the absolute position of a cell in the path is irrelevant to the score
- In the cell-pair representation, only the second cell in each pair contributes to the score. The cell-sequence representation encodes the same information and is more compact

Solution structure

In general, the solutions of combinatorial optimization problems can be understood as *subsets* of a larger set of *components*, the *ground set*

- Unified modelling of combinatorial optimization problems
- Feasible solutions are subsets of components, but not all component subsets represent feasible solutions
- Constraints on the joint presence of certain components in a solution
- Relation between solution quality and the components, or combinations of components, in a solution

⇒ Solving the problem by learning which (combinations of) components lead to good solutions!

Solution structure

Constructive-search methods build solutions by successively adding components to the empty set

- The number of components in a feasible solution is usually constrained from above
- Depending on the problem, it may also be constrained from below

⇒ The decision space may (have to) include infeasible solutions with fewer components than required for a solution to be feasible

- Solutions are called:
  - *Partial* or *incomplete*, if more components can still be added to them, and *complete* otherwise
  - *Feasible*, if they satisfy the constraints of the problem, and *infeasible* otherwise

In general, both partial and complete solutions may be feasible or infeasible. The decision space is usually such that complete solutions are feasible, but that is not always the case

Solution structure

Partial solutions may be interpreted in two complementary ways

1. As extending the decision space in order to allow the construction of feasible solutions (when partial solutions are themselves infeasible)
2. As a representation of the sets of feasible solutions that can be constructed by adding components to them (whether they are themselves feasible or infeasible)

The second interpretation can be extended further by allowing certain components to be excluded from those that can be added to a solution in order to complete it

- Solutions are *augmented* with a set of *forbidden* components
- When components are forbidden, infeasible complete solutions may arise
- Support for effective decision-space subdivision, as required by Branch and Bound

Construction rules

Consider a ground set $\mathcal{G}$, a (possibly augmented) decision space $\Omega \subseteq 2^{\mathcal{G}}$, a solution $s \in \Omega$, and a component $c \in \mathcal{G}$

- If $c \notin s$ and $s \cup \{c\} \in \Omega$, then a new solution $r = s \cup \{c\}$ can be constructed by *adding c to s*
- The reverse is also possible: if $c \in s$ and $s \setminus \{c\} \in \Omega$, then a new solution $r = s \setminus \{c\}$ can be constructed by *removing c from s*
- This imposes a *neighbourhood structure* on the decision space Ω
- Moving from one solution to another by adding or removing a component is called a *move*
- Solution construction consists in traversing the decision space from the empty solution to any feasible solution by successively adding components
- The construction path from one solution to another may not be unique

Construction rules

The solution construction process is determined by

- The decision space and the associated neighbourhood structure
- Additional restrictions to which paths can be followed from one solution to another

Example

Building symmetric Travelling Salesman tours

Decision space Partial solutions may consist of

A. Degree-2 trees

B. Degree-2 forests

Additional restriction Degree-2 trees may or may not have to have a degree-1 root

Construction rules

Example

Building knapsack solutions

Decision space Any subset of items with total weight below knapsack capacity is a feasible solution

Additional restriction Items may be added

A. In any order

B. Only in a prescribed order

Note All feasible solutions can be constructed in both cases

Solution quality

The original problem definition should suggest a “natural” means of evaluating solutions

- Relational operator (pairwise comparisons)
- Objective function(s)
- Not all objective functions are good for actually solving the problem
  - Computing the objective value may be complicated and/or time consuming
  - *Bottleneck* (or min-max) objectives and simple counts may lead to many ties that make it difficult to gauge progress
  - Objective value *landscapes* may also be *rugged* or even *deceiving*

When the “natural” objective function is insufficient or inappropriate to guide the search, an alternative objective function, or *proxy function*, should be defined.

Evaluation of partial solutions

When partial solutions are *feasible*, the objective function can be used *directly* to guide construction

- Focus on the quality of the *current* (partial) solution
- Improve the solution at each construction step (as much as possible)
- *Short-sighted* approach

Alternatively, and particularly when partial solutions are *infeasible*, the objective function can also be used to guide solution construction *indirectly*

- Focus on the quality of the best possible *feasible* solution that can be still constructed
- Objective function *lower bound* (assuming minimization)
- Avoid degrading the bound value at each construction step (as much as possible)
- *Forward-looking* approach

Evaluation of partial solutions

When partial solutions are *infeasible*, the objective function cannot be used directly to guide solution construction

- Extending the objective function to consider infeasible, partial solutions
  - Complete solutions heuristically in order to obtain *feasible* solutions
  - *Upper bound* values (considering minimization)
  - Alternatively, assign heuristic values reflecting (perceived) solution quality
- Considering the quality of the best possible *feasible* solutions that can be constructed from given partial solutions
  - *Lower bound* values (considering minimization)
  - Also useful when partial solutions are themselves *feasible*

Bounds

Obtaining *upper* and *lower* bounds (under minimization)

- Upper bounds often involve finding feasible solutions, which may itself be difficult
- Lower bounds refer to how good feasible solutions can possibly be, but do not involve finding actual solutions
- Bound quality criteria
  - Closeness to the true optimal value
  - Positive association with the objective function
  - Ease of computation

Lower bounds may be obtained in several different ways

Bounds

Bound functions

- Suppose that the objective function is a combination of terms that depend on the components present in feasible solutions
- Given a *partial* solution, only some of the possible terms in a more complete, feasible solution will be known
- Replace the unknown terms by *optimistic* values such that the resulting objective function value cannot be surpassed by any feasible solution

Example – Travelling Salesman Problem (minimization)

- The cost of any tour is equal to half the sum of the costs of the two incident edges of each vertex
- Plugging in the *smallest* incident edge costs for each vertex still missing edges gives a lower bound

Bounds

Relaxation

- Suppose that optimal solutions would be easy to construct if some problem constraint(s) could be ignored (*relaxed*) during construction
- Solving the relaxed problem *exactly* may lead to *infeasible* solutions to the original problem
- The *optimal* objective value of the relaxed problem is a lower bound for the original problem (under minimization)

Example – 0–1 Knapsack Problem (maximization)

- The optimal knapsack value is difficult to determine because items must fit in completely
- Allowing items to fit in partially (*continuous* relaxation) makes the problem easy to solve

Bounds

Types of constraint relaxation

Continuous relaxation Allowing (some) discrete variables to take continuous values

Simple combinatorial relaxation Simply ignoring some constraints

Surrogate relaxation Reducing the number of constraints by linearly combining some of them

Lagrangian relaxation Linearly combining some constraints with the objective function

In the last two cases, the quality of the resulting bounds depends on the combination weights

Bounds

Restriction

- Suppose that optimal solutions can be easily constructed if an additional constraint is added
- Solving the restricted problem *exactly* may lead to *suboptimal* solutions to the original problem
- If it can be show that the *optimal* objective value of the restricted problem *approximates* that of the original problem to some factor, a corresponding upper or lower bound can be obtained

Example – Steiner Minimum Trees (SMTs)

- Spanning trees are particular cases of Steiner trees with no Steiner points
- Under certain assumptions, minimum spanning trees (MSTs) are a 2-approximation of SMTs
- Half the weight of a MST is a lower bound on the weight of the corresponding SMT

Bounds and construction rules

In combinatorial optimization, solutions are sets of components from the ground set

- Neighbourhood structure on the decision space, where (partial) solutions that differ in a single component are neighbours of each other
- More restricted neighbourhood structures can be considered
  - For simplicity of construction, so that structural constraints can be more easily enforced
  - For simplicity of partial-solution evaluation, so that bounds can be more easily defined or computed

  $\Rightarrow$ Construction rules
- All feasible solutions should be reachable, but not all infeasible (partial) solutions need to be

⚠ Bound values depend on the actual construction rule!

Concluding remarks

- Decision space and solution representation
- Solution structure
- Solution construction tied to
  - The decision space
  - Additional constraints
- Solution quality
- Lower bounds (under minimization)
  - Forward-looking versus short-sighted approaches

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.

COST (European Cooperation in Science and Technology) is a funding agency for research and innovation networks. Our Actions help connect research initiatives across Europe and enable scientists to grow their ideas by sharing them with their peers. This boosts their research, career and innovation.

www.cost.eu