A conceptual model of optimisation problems

Cristina C. Vieira[*]
[*]Faculdade de Ciências e Tecnologia
Universidade do Algarve
Faro, Portugal
cvieira@ualg.pt

Carlos M. Fonseca[*†]
[†]CEG-IST - Centre for Management Studies
Instituto Superior Técnico
Lisboa, Portugal
cmfonsec@ualg.pt

***Abstract*—This paper proposes a conceptual model of optimisation problems to support their implementation in software. Solution representation, search space structure, and solution evaluation, are the central concepts of the proposed model. The desired separation between the problem and the search methods is achieved by recognising that neighbourhood structures are an integral part of a local optimisation problem definition, while considering that optimisers should be formulated independently of specific neighbourhood structures and of their implementation details. The proposed model has been prototyped in Python, as a research and teaching tool. A framework foundation component and an application example based on a genetic algorithm for the N-Queens problem are presented.**

I. Introduction

In the last decade there has been great interest in software environments for optimisation [1]–[8]. One of the main challenges when designing such an environment consists of achieving a clear separation between problems and solvers, so that solvers may be written independently of the problems they will be applied to, and that problems may be written independently of the specific search operators employed by the solvers. This is usually a difficult task, because the implementation of local search operators, including mutation and recombination operators in evolutionary algorithms, typically requires knowledge of how solutions are represented. As a result, search operators are often seen as establishing a connection between problems and solvers, which amounts to placing them in a class of their own [7]. Arguably, a clearer understanding of the problem-specific and solver-specific aspects of search operators would lead to an even greater separation between problem and solver implementation. In turn, this would facilitate deployment, improve the understanding of the optimisers themselves, and enable a fairer assessment and comparison of the performance of different optimisation methods.

In this work, a new model for the software implementation of optimisation problems is proposed. Emphasis is placed on the problem-specific aspects, whereas problem-independent search strategies are left to be implemented by the solver. Solution representation, search-space structure, and solution evaluation, are the central concepts of the proposed model. The desired separation between the problem and the solver is achieved by recognising that neighbourhood structures are an integral part of a local optimisation problem definition. This implies that optimisers should be formulated at a higher level, independently of specific neighbourhood structures and, thus, of their implementation details.

This paper is organised as follows. The basic entities and relationships needed to formulate an optimisation problem are identified in section II, starting from some basic mathematical definitions. In section III, a conceptual model of optimisation problems is proposed. Finally, in section IV, a short description of the model implementation in software is presented. Section V introduces an application example based on the N-Queens problem. The last section discusses the benefits and potential drawbacks of the proposed approach.

II. Problem analysis

Since an optimisation problem exists independently of any method which may be used to solve it, it should be possible to identify, and model, the entities and relationships involved in a complete formulation of an optimisation problem, without referring to any specific optimiser. In order to arrive at a general optimisation problem formulation, the analysis begins with some general mathematical definitions.

A. Optimisation problems

Consider the basic problem definition given in [9]:

*Definition 1 (**Abstract problem**):* An abstract problem Q is a binary relation on a set I of problem instances and a set S of problem solutions.
This definition is very general, but it immediately suggests the decomposition of a problem into two separate entities, a set of instances and a set of solutions. Clearly, an optimisation problem may be seen as a special case of such an abstract problem. Although optimisation problems are usually formulated in terms of numerical functions, a more general definition based on *pre-order* relations [10] may be given, as suggested in [11], [12]. Note that functions are special cases of relations.

*Definition 2 (**Optimisation problem**):* An optimisation problem is an abstract problem where each instance in I is a *pre-order* $\preceq$ on the set of problem solutions S. An element of S is a solution of a problem instance in I if and only if it is a *minimal* element [10] of S under the corresponding pre-order (alternatively, *maximal*).
This definition refines the previous one by specifying the nature of problem instances, namely, that they must consist of binary relations on S and possess certain properties (reflexivity and transitivity), but does not introduce any new entities. It

does, however, make problem instances depend on the solution space, as it happens in practice. Definition 2 encompasses a number of different optimisation scenarios, namely:

Discrete and continuous optimisation:
: Suitable pre-orders $\preceq$ may be defined regardless of whether S is *discrete* (finite or countably infinite) or *continuous*.

Single-objective and multiobjective optimisation:
: *Pareto-dominance* defines a (partial) pre-order, and so do other *preference relations* [13]. Considering Pareto-dominance and n objective functions, one may write:

$$\forall s_1, s_2 \in S,\ s_1 \preceq s_2 \Leftrightarrow \\ (f_1(s_1), \ldots, f_n(s_1)) \leq (f_1(s_2), \ldots, f_n(s_2))$$

where the inequality between vectors must be verified in a componentwise fashion.

Multimodal optimisation:
: The problem of finding *all* minimal elements of S under a given *pre-order* $\preceq$ may be formulated as a new problem, defined in terms of a suitable *pre-order* $\preccurlyeq$ on the power-set 2^S [14], [15]. This includes finding all Pareto-optimal solutions in the multiobjective case.

B. Local optimisation problems

Definition 2 formalises *global* optimisation problems, since the notion of minimal element is taken with respect to the whole set of solutions, S. In practice, global optimisation problems are usually very difficult to solve directly, and *local* optimisation problems are often considered instead.

*Definition 3 (**Local optimisation problem**):* A local optimisation problem is an abstract problem such that each instance in I is a pair $(\preceq, N)$, where $\preceq$ is a *pre-order* on the set of problem solutions S, and $N(s) : S \to 2^S$ is a *neighbourhood function*. An element $s \in S$ is a solution of a problem instance in I if and only if it is a *minimal* element of the subset $N(s) \subseteq S$, under the corresponding pre-order. The neighbourhood function N endows S with a *neighbourhood structure*.

Note the introduction of a new entity, the neighbourhood structure, its direct dependence on the solution set S, and the fact that the local problem differs from its global counterpart only in the mapping between I and S. This makes the neighbourhood structure an integral part of the problem definition, independently of how it may be exploited by any given solving strategy. Also, note that the global optimisation problem remains a special case of the local optimisation problem, where $N(s) = S$, $\forall s \in S$.

III. The Conceptual Model

The model consists of three main components: a *solution space*, an optional *neighbourhood structure*, and an *evaluator*. Each component provides a *representation* of the corresponding mathematical entity, as well as *mechanisms* capable of supporting the optimisation process, such as: (1) Solution initialisation and solution space sampling; (2) Neighbourhood sampling, distance measuring and shortest path generation and (3) Solution comparison and evaluation, including incremental evaluation. Such mechanisms should be made available to the optimiser in a problem-independent way, using a standard interface for all problems.

A. Solution space

The solution space is independent of the remaining components of the model. Its main purpose is to specify how candidate solutions are represented and to provide mechanisms for solution initialisation and, eventually, stochastic sampling of the solution space. Solution initialisation should encapsulate any data needed to describe concrete solutions (including user-supplied initial solutions, regardless of how they were obtained) in such a way that, *once initialised*, solutions may be handled independently of the underlying representation adopted. Stochastic sampling of the solution space is important because it enables an optimiser to obtain candidate solutions in a problem-independent way. Sampling may be implemented with replacement and, eventually, also without replacement. The availability of a stochastic space-sampling mechanism, with and without replacement, is also required to implement random search and exhaustive enumeration, respectively.

B. Neighbourhood structure

The neighbourhood structure depends directly on the solution space component, and must know about how solutions are represented. Its main purpose is to support local optimisation methods and, as such, it should offer a number of important mechanisms. At the most basic level, it should implement the neighbourhood function in some way. Since the value of this function is a set, determining the neighbourhood function may be reinterpreted as sampling the neighbourhood without replacement, at least when the neighbourhood is finite. Additionally, mechanisms to enable sampling with replacement, determining the distance between two solutions (under the given neighbourhood structure), and determining paths from one solution to another should be provided, so that search techniques such as path-relinking [16] and certain recombination operators [17], [18] may be implemented by the optimiser independently of the problem. Where the solution space possesses stronger mathematical properties (as in the continuous case), these may be used to define the neighbourhood *structure* in alternative ways. Finally, the neighbourhood structure may define a representation for the "difference" between two neighbouring solutions, which is usually described as a *move*, in which case it should also provide mechanisms to apply moves to solutions and compute the resulting neighbours. This usually becomes advantageous at the *evaluator* level.

C. Evaluator

The evaluator must allow solutions to be compared with one another, either directly, by implementing a comparison operator, or indirectly, by implementing one or more objective functions. In both cases, the implementation of the evaluator is

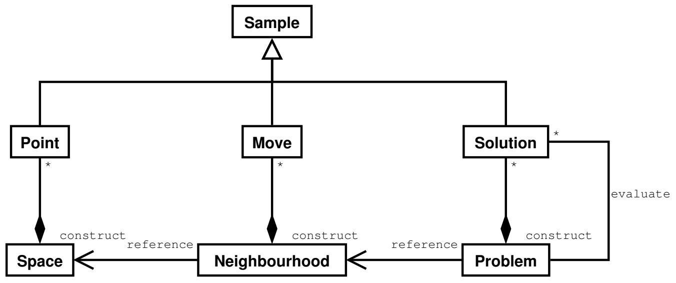

Fig. 1. Metamodel class diagram

intimately dependent on the solution space. Provided that the neighbourhood structure also defines moves, it may become possible for the evaluator to evaluate moves in connection with given pre-evaluated solutions. Incremental, or partial, solution evaluation may be much more computationally efficient than the full evaluation of solutions, and is one of the most important features of local search methods.

IV. Model implementation

The proposed model was implemented in Python as a set of interrelated abstract classes, representing the model's high-level abstraction concepts, and forming the framework *foundation*. Python is a general-purpose aspect-oriented language that promotes the fast development of prototypes for experimentation and application development purposes, and can be used in multiple platforms. Moreover, the availability of numeric package *NumPy* has made it extremely interesting for scientific computing.

The metamodel class diagram is depicted in Figure 1. Concrete spaces must be derived from the abstract class *Space*, which represents all generic space properties. The abstract class *Neighbourhood* represents generic neighbourhood structures and, as with the *Space* abstract class, concrete neighbourhoods must be derived from it. Concrete neighbouhoods must also reference a concrete space.

Concrete problems are derived from the abstract class *Problem*, and must reference a concrete space, possibly through a concrete neighbourhood structure. Problem objectives are implemented as solution evaluators, and must provide solution comparison or objective function evaluation, or both. Optionally, incremental evaluation and/or comparison may also be provided.

Point, *Move* and *Solution* are special classes containing single or multiple elements that only exist in connection with concrete *Space*, *Neighbourhood* and *Problem* class instances.

Space classes provide mechanisms to generate and manipulate single or multiple points, while *Neighbourhood* classes provide mechanisms to generate moves and manipulate single or multiple points and moves, such as applying moves to points and determining the distance between pairs of points. *Problem* classes provide mechanisms to construct Solution instances, often defined in terms of the methods implemented by Space. *Solution* instances combine Point instances with the logic (methods and attributes) needed to implement solution comparison and evaluation, including incremental evaluation. In addition, they are endowed with neighbourhood-related methods, which may often be provided automatically based on the corresponding implementations made available by the Neighbourhood and Move classes. As a result, optimisers may simply rely on the Problem and Solution classes to provide all the funcionality they need.

V. Application example

A. Modelling and implementing the N-Queens problem

The N-Queens problem is a generalisation of the classic 8-Queens combinatorial optimisation problem proposed by chess player Max Bazzel in 1848. The problem consists in placing n queens on a chessboard so that no two attack each other, i.e., no more than one queen may be on the same row, on the same column and on the same diagonal. By representing solutions as permutations, where the ith-value represents the row in which the queen in column i is placed, it becomes sufficient to check for diagonal conflicts, which considerably simplifies solution evaluation. The swap (or exchange) neighbourhood is considered a suitable neighbourhood for this problem [19].

The proposed *NQueens* problem class implementation derives from the abstract class *Problem*, and implements the *constructor* and the *Solution* class *evaluate* hotspot methods. The constructor initialises problem instances with the chessboard length n, and instantiates concrete *Permutation* and *Swap* classes to describe the corresponding representation space and neighbourhood. The *Solution evaluate* method computes the number of pairs of queens on the same diagonal, providing incremental as well as full solution evaluation.

B. A simple genetic algorithm for the N-Queens problem

A simple genetic algorithm implementation (see Algorithm 1) uses the random solution sampling mechanism provided by the problem to generate the initial population in a a problem-independent way. The solution evaluation mechanisms are called whenever the population is queried for its objective values, e.g. by the termination condition. Evaluation results are cached by the solution objects so that subsequent queries for the objective values may be returned without a re-evaluation (e.g., during selection). This also provides the basis for incremental evaluation.

Algorithm 1 *Genetic algorithm*

```
$problem \leftarrow$ **NQueens(n)**
$pop \leftarrow$ **problem.randomSolution(m)**
**while** not termination condition **do**
    $pop \leftarrow select(pop)$
    $recombine(pop)$
    $mutate(pop)$
**end while**
```

Geometric mutation and recombination operators [20] are implemented by calling the relevant solution mechanisms, which are, in turn, provided by the problem neighbourhood instance. The mutation operator is implemented as the application of random moves to individuals, using the *randomMove* solution method and an overloaded in-place addition operator.

Algorithm 2 *Mutation operator*

```
$idx \leftarrow mutateIdx(m, mutationRate)$
**pop[idx] += pop[idx].randomMove()**
```

The recombination operator uses the *distanceTo* and the *randomMoveTowards* solution methods to generate offspring which lie at a random distance between the two parents.

Algorithm 3 *Recombination operator*

```
$idx \leftarrow crossIdx(m, crossoverRate)$
$p1 \leftarrow pop[idx]$
$p2 \leftarrow shuffle(p1)$
$d \leftarrow random(1,$ **p1.distanceTo(p2)/2**$)$
**for** $i = 0$ to $len(p1)$ **do**
    **for** $j = 0$ to $d[i]$ **do**
        **p1[i] += p1[i].randomMoveTowards(p2[i])**
    **end for**
**end for**
$pop[idx] \leftarrow p1$
```

Note how problem-independent aspects (mutation and recombination rates, the pairing of individuals and the amount of information to exchange during recombination) remain clearly separated from the problem-specific, representation related aspects of these operators.

VI. Conclusion and future work

In this paper, a conceptual model of optimisation problems was proposed in which problem-specific aspects are clearly identified as such, and are presented to the solver as problem-independent mechanisms. The main components of the model were identified and the model was implemented in software. An application example was given to illustrate the achieved separation between problem-specific and problem-independent aspects of the recombination and mutation operators.

In contrast with other approaches, the proposed model has the advantage of isolating the application expert from the details of specific search operators, such as mutation and recombination operators. Instead, the problem implementation is only required to provide low-level mechanisms directly related to the concrete problem neighbourhood. The actual search operators can then be implemented at a higher level by the optimiser, as long as they are clearly formulated with respect to an underlying (abstract) neighbourhood. The model also contemplates the caching of intermediate results and incremental solution evaluation as peformance enhancement techniques.

Although the model described in this paper is quite general and should cover many problem formulations not considered here, this must be demonstrated on additional application cases. The development of a complete optimisation framework based on this model is the subject of on-going work.

References

[1] A. Fink and S. Voß, "HotFrame: A heuristic optimisation framework," in *Optimization Software Class Libraries*, D. Woodruff, Ed. Kluwer Academic Publishers, 2002, ch. 4, pp. 81–154.

[2] L. Gaspero and A. Schaerf, "EasyLocal++: an object-oriented framework for the flexible design of local-search algorithms," *Software – Practice and Experience (SPE)*, vol. 33, no. 8, pp. 733–765, 2003.

[3] S. Cahon, N. Melab, and E.-G. Talbi, "ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics," *Journal of Heuristics*, vol. 10, no. 3, pp. 357–380, 2004.

[4] S. Luke, L. Panait, G. Balan, Z. Skolicki, J. B. R. Hubley, and A. Chircop, "ECJ 18 – A Java-based evolutionary computation and genetic programming research system," 2008. [Online]. Available: http://cs.gmu.edu/∼eclab/projects/ecj/

[5] C. Voudouris and R. Dorne, "HSF: A generic framework to easily design meta-heuristic methods," in *MIC'2001: Proceedings of the 4th Metaheuristics International Conference*, Porto, Portugal, 2001, pp. 423–428.

[6] J. A. Parejo, J. Racero, F. Guerrero, T. Kwok, and K. A. Smith, "FOM: A framework for metaheuristic optimization," in *Computational Science – ICCS 2003, Proceedings, Part IV*, ser. LNCS. Springer, 2003, vol. 2660, pp. 886–895.

[7] S. Wagner and M. Affenzeller, "HeuristicLab: A generic and extensible optimization environment," in *ICANNGA: Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms*, Coimbra, Portugal, 2005, pp. 538–541.

[8] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA: A platform and programming language independent interface for search algorithms," in *EMO 2003: International Conference on Evolutionary Multi-Criterion Optimization*, ser. LNCS. Springer-Verlag, 2003, vol. 2632, pp. 494–508.

[9] T. H. Cormen., C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[10] P. Taylor, *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

[11] C. M. Fonseca, "Evolutionary multi-criterion optimisation," in *Soft Computing and Complex Systems*. Coimbra, Portugal: Centro Internacional de Matemática, 2003, pp. 63–75.

[12] C. C. Vieira and C. M. Fonseca, "A unified model of optimisation problems," in *GECCO '07: Proceedings of the Annual Conference on Genetic and Evolutionary Computation*. New York: ACM, 2007, pp. 1537–1537.

[13] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. Part I: A unified formulation," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 28, no. 1, pp. 26–37, 1998.

[14] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *PPSN VIII: 8th Internationl Conference on Parallel Problem Solving from Nature*, ser. LNCS. Springer, 2004, vol. 3242, pp. 832–842.

[15] L. Paquete, T. Schiavinotto, and T. Stützle, "On local optima in multiobjective combinatorial optimization problems," *Annals of Operations Research*, vol. 156, no. 1, pp. 83–97, 2007.

[16] F. Glover, M. Laguna, and R. Marti, "Fundamentals of scatter search and path relinking," in *Control and Cybernetics*, vol. 29, no. 3, 2000, pp. 653–684.

[17] C. R. Reeves, "Genetic algorithms and neighbourhood search," in *Evolutionary Computing, AISB Workshop*, ser. LNCS. Springer, 1994, vol. 865, pp. 115–130.

[18] R. Poli and C. R. Stephens, "Theoretical analysis of generalised recombination," in *CEC 2005: Proceedings of the Congress on Evolutionary Computation*. Edinburgh, UK: IEEE, 2005, pp. 411–418.

[19] R. Sosic and J. Gu, "Efficient local search with conflict minimization: A case study of the n-Queens problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 5, pp. 661–668, 1994.

[20] A. Moraglio and R. Poli, "Topological interpretation of crossover," in *GECCO '04: Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, ser. LNCS. Springer, 2004, pp. 1377–1388.