Design Principles for Metaheuristics: Problem Types and Avoiding Bottlenecking

Colin G. Johnson
Computing Laboratory
University of Kent
C.G.Johnson@kent.ac.uk

Abstract: *This paper is concerned with an aspect of the design of metaheuristic algorithms, such as evolutionary algorithms, tabu search and ant colony optimization. The topic that is considered is how problems can be* represented *when they are given to a metaheuristic algorithm. A particular difficulty is presented,* viz. *the "bottleneck", where the problem is artificially converted into a new representation in order to fit the standard input to the metaheuristic. Such bottlenecks cause problems in interpreting or trusting the solution given by the metaheuristic. In order to alleviate this problem, we suggest ways in which three types of problem (data-driven, specification-driven and interactive) can be presented to metaheuristics in a bottleneck-free way, and how problems which use multiple solution-types can be tackled.*

Keywords: Metaheuristics, software design, specification, interactivity, bottlenecks, problem-solving.

1 Introduction

The aim of this paper is to discuss an aspect of the design of metaheuristic algorithms. By *metaheuristics* we mean computational algorithms which perform heuristic search over some search space, and which are not tied to a specific problem. Examples include evolutionary algorithms, tabu search, and ant colony optimization. This work fits into a wider project which aims to provide a framework for the sensible design of such metaheuristics—a *design framework*. The eventual aim of this framework is to make clear the choices which need to be made in applying metaheuristics, and provide a set of techniques which support those design decisions. This paper focuses on one such choice, *viz.* the choice of how the problem is represented as an input to the metaheuristic.

2 Describing problems

An important part of solving a problem on a computer is giving some kind of description of the problem to the machine. Clearly in general this encompasses the whole of computer programming. We can narrow this down by asking a more focused question: what (if any) is the best way to describe a particular problem to a metaheuristic algorithm?

One way to approach this is by taking some ill-defined problem in the world and making the problem more "formal" or "exact". Some problems (or aspects thereof) are capable of being described compactly in such a format. Examples: a robot should not be allowed to operate outside a particular area; a timetable should not expect the same person to be in two places at once; a program should identify whether a particular record is in a database or not. Problems which can most naturally be described in these terms can be called *specification-defined problems*.

However not all problems fit into this category of specification-defined problems. Many problems are *essentially* defined by data, as noted by Partridge and colleagues [14, 15, 16]. These can be termed *data-defined* problems:

> *"Many problems, however, are manifest as little more than sets of input-output data. They exist in systems of high complexity where our knowledge of the underlying mechanisms is both crude and fragmentary. Some examples of data-defined problems are: human face recognition; signature recognition; prediction of periodic fluctuations in water consumption of electricity demand for a city; optimal control of chemical processes and manufacturing plants; adjustment of treatment dosage on the basis of bodily response to last dosage, etc. In all of these cases, it is far easier to collect examples of the data (both good and bad examples) than it is to determine in more than a very rough and fragmentary manner precisely how the output depends on the input." [15]*

The danger with such problems is that it is tempting to force them through a *specification bottleneck* (Partridge's term). In this situation the practitioner takes a number of examples of the problem at hand and *constructs* an artificial specification which abstracts from the original data. In some instances of this the process may be good, because in the process of abstraction the practitioner-as-expert adds in expert knowledge which is not easily to extract from the examples alone. Often, however, this is an entirely artificial exercise which can easily remove aspects of the original data which are important but not obvious to the specification-maker.

However there is a converse danger which is less well known, which we shall term a *data bottleneck*. This problem arises when the user trains the metaheuristic using a sample of training data that satisfies the specification, rather than measuring the extent to which the solutions satisfy the specification directly. Again, the problem has been presented to the metaheuristic in a way which does not respect the structure of the problem.

It would seem to be ideal to present both data-defined and specification-defined problems to the metaheuristic so that their fitness is evaluated without forcing the problem through a bottleneck. How this can be achieved is discussed below.

A third type of problem which doesn't fit naturally into either the specification-defined or data-defined category is the class of problems which are *interactive*. Attempts at the "solution" of such "problems" (those terms are rather problematic in this context) are defined not with respect to a predefined notion of quality, but defined via interaction with a user as the individuals are generated.

There are a number of reasons why this interaction might be a core part of how success is defined. One reason, as demonstrated by the examples in [1, 2], is that an aesthetic judgement needs to be made about the objects generated: are they beautiful, are they engaging, do they blend harmoniously with other objects? A second, related, kind of search is a search for individuals which have some subjectively-assessed quality. For example a metaheuristic applied to music synthesis (e.g. [7]) may have the aim of producing a sound which is melancholy in quality. Finally the aim of a particular application may be to exploit the ability of humans to pick out patterns in complex environments. This idea has been applied by Venturini et al. [19] in data mining, where various views on a dataset are provided to a human user and the user interactively evolves those which pick out particular interesting features of the dataset.

Again there are difficulties if problems which are naturally interactively-defined are presented to metaheuristics in a different way. An interactively-defined problem can be forced through a specification bottleneck. For example the user might try and define what a melancholy melody

might be: slow, in a minor key, et cetera. Or in can be forced through a data bottleneck, for example by giving lots of examples and training the machine via some measure of similarity to those examples. Once again, neither of these approaches seems satisfactory; it would seem that the interactively-defined problems are a distinct problem-type.

We have defined three problem-types: data-defined, specification-defined, and interactively-defined. Some applications may involve aspects of each. To summarize this section here are examples of each of the problem types and combinations thereof, drawn from mobile robotics.

Pure specification A robot should move from point A to point B, without hitting any obstacles.

Pure data The robot should move towards a particular person, regardless of where they are in the room.

Pure interactive The robot should trace out a pattern with the pen attached to it which is interesting to an observer.

Specification and data The robot should chase another robot, whilst not leaving a predefined area.

Specification and interactive The robot should trace out a pattern which is interesting to an observer, but never move further than one metre in each minute.

Data and interactive The robot should recognize another robot in its environment and interact with that robot in a style which engages the attention of an audience.

Specification, data and interactive The robot should recognize another robot in its environment and interact with that robot in a style which engages the attention of an audience, whilst not moving out of a metre-wide square on the floor.

3 Design choices for problem-presentation to metaheuristics

It would be desirable to find ways in which each of the three problem types could be tackled using metaheuristic methods without needing to carry out the "bottleneck" transformations. Moreover it is important that *combinations* of these problem-types can be processed by the metaheuristic in a way which allows the various aspects to be processed in the way that is most natural to them. This is one of the reasons why the various bottlenecks are often resorted to: for example a metaheuristic is designed to take problems in the form of training data, so when a user wants to specify something which is naturally a piece of specification (e.g. a safety constraint) they translate that into the input language of the metaheuristic by providing a set of examples.

It is in the assessment of putative solutions to problems by the metaheuristic that these different problem types become important. Such an assessment is part of most metaheuristic approaches, for example the calculation of fitness in genetic algorithms. These assessments typically offer a numeric quality score or (for population-based methods) a ranking of the current attempts.

Data-defined problems fit most naturally into this framework. The data used in defining the problem provides a natural set of *fitness cases* against which the solution-attempt can be assessed.

Specification-defined problems fit less naturally. Typically these are forced through a data bottleneck by using the specification to generate a number of fitness cases which are compatible with the specification. To deal with such problems in a way that respects the structure of the

problem we need to develop methods of directly assessing whether a potential solution is compatible with the specification, and to measure how far such solutions are from the specification. One approach is through the use of *static program analysis* [13] in the determination of fitness in genetic programming. That is, instead of running the generated program on a set of test cases, the fitness is assessed by running an analysis on the program which checks whether the program is compatible with statements in the specification, *regardless* of input data. Thus the specification is never driven through the data bottleneck in being assessed. This has been successfully applied to some simple problems in genetic programming [6, 8, 9].

This technique allows the specification-defined problem to be handled naturally, that is a direct check is made on whether the program satisfies the specification rather than this being checked indirectly. Nonetheless there are problems with this approach. Firstly there is the problem that specification-satisfaction is very often a binary true-false condition: either the program satisfies the condition or it does not. This could potentially make it difficult for a search algorithm within a metaheuristic to get a grip on how close a particular attempt is to satisfying the specification. There are a number of approaches to this. Firstly for some problems there may be many small, easy-to-satisfy-individually specification statements, and it is bringing all of these together into a single satisfactory specification which is the difficulty. In these cases a count of the number of specification statements provides a fitness measure. Secondly it may be possible to devise weaker specification which smooth out the fitness landscape, e.g. by breaking down a complex specification statement into a number of sub-statements. An example of work similar to this is the recent work by Harman et al. [5] on evolutionary testing which smoothes out the fitness landscape by ensuring that rarely visited areas of code are visited more frequently by adjusting the input space.

Finally, interactive problems are superficially easy to fit into a framework of fitness-assessment, but there are a number of issues which need to be considered more carefully. The previous two methods were *consistent*, in the sense that the same individual presented to the fitness-evaluation algorithm would be scored the same (or ranked the same relative to others). This is not true for interactively-defined problems. In these problems the whim of the human assessor or the context of a particular individual in a population can change the assessment of its fitness. Indeed there is some evidence [11] for an implicit *fitness scaling* effect in such systems; initially the user will give a high fitness score to anything that is vaguely like a desired/desirable output. However as the population becomes occupied by "better" individuals, individuals which scored highly early may be scored less well relative to the more converged population.

These are interesting issues, but they are not *difficulties* with the use of metaheuristics for interactively-defined problems. Indeed they are natural ways of interacting with individuals in this situation. For example it is more natural to make aesthetic *judgements* in a comparative fashion (A is better/more exciting/more beautiful/more interesting than B) than to give absolute ratings. Indeed it seems reasonable to say that the above issues are advantages, because they deal with these interactively-defined problems in a natural way, rather than forcing them to be dealt with using concepts such as consistency of evaluation which are more suited to the other two types of problems.

Nonetheless there are difficulties with using metaheuristics for interactively-defined problems. One difficulty, notable particularly with population-based approaches, is that users become bored with making appraisals of many different individuals. One possible solution to this is to embed the evolutionary within a natural context such as a virtual environment [18] or a performance setting [3]. This could be enhanced by the use of affective computing techniques [17] to directly assess user's affective response to individuals in the population.

As briefly discussed earlier, some problems have aspects which belong to more than one category.

For example a problem can easily have its task specified in a data-defined fashion whilst also needing to satisfy some constraints which are given in a specification-defined way. This is where the use of metaheuristics in the way described above can do things which are difficult for other methods. For example multicriterion optimization could be applied to find solutions which are both formally compatible with specification statements whilst solving a data-rich problem. Such techniques have been applied, for example, to the evolution of a robot controller which guaranteedly satisfies a dynamic safety constraint whilst also following another robot [10].

Typically methods of creating software are tied closely to problem type. So for example specifications can be converted into programs which satisfy that specification using a formal method such as refinement [4, 12]. However once we have committed to developing a piece of software in that fashion, it becomes difficult to incorporate aspects of the problem which are naturally defined e.g. in a data-defined way. The metaheuristic approach makes combination of methods much easier to achieve.

4 Conclusions

This paper has introduced the idea that there are natural ways of describing problems, and given three examples of such descriptions (data-defined, specification-defined and interactively-defined) which cover a wide range of real-world problems. The disadvantages of solving problems by putting them through a *bottleneck* and forcing them into a different framework has been outlined and examples given. This classification of problem types has then been used as a way of classifying the inputs to metaheuristics, and ways in which problems of the various kinds can be naturally represented to metaheuristics has been discussed. Finally, the use of multicriterion optimization as a way of tackling problems that contain multiple problem-types as aspects of their solution has been discussed. Some practical examples of all of these approaches are given in the papers cited; this paper attempts to bring these together as a soft engineering design framework.

References

[1] Peter J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann, 1999.

[2] Peter J. Bentley and David W. Corne, editors. *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.

[3] John A. Biles. GenJam Populi: Training an IGA via audience-mediated performance. In *Proccedings of the 1995 International Computer Music Conference*, 1995.

[4] John Derrick and Eerke Boiten. *Refinement in Z and Object-Z*. Springer, 2001.

[5] Mark Harman, Lin Hu, Rob Hierons, Andre Baresel, and Harmen Sthamer. Improving evolutionary testing by flag removal. In W. B. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2002.

[6] Lorenz Huelsbergen. Abstract program evaluation and its application to sorter evolution. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1407–1414. IEEE Press, 2000.

[7] Colin G. Johnson. Exploring the sound-space of synthesis algorithms using interactive genetic algorithms. In Geraint A. Wiggins, editor, *Proceedings of the AISB Workshop on Artificial Intelligence and Musical Creativity, Edinburgh*, 1999.

[8] Colin G. Johnson. Deriving genetic programming fitness properties by static analysis. In James Foster, Evelyne Lutton, Conor Ryan, and Andrea Tettamanzi, editors, *Proceedings of the 2002 European Conference on Genetic Programming*. Springer, 2002.

[9] Colin G. Johnson. What can automatic programming learn from theoretical computer science? In Xin Yao, Qiang Shen, and John Bullinaria, editors, *Proceedings of the 2002 UK Workshop on Computational Intelligence*, 2002.

[10] Colin G. Johnson. Genetic programming with guaranteed constraints. In Ahmad Lotfi and Jonathan M. Garibaldi, editors, *Applications and Science in Soft Computing*, pages 95–100. Springer, 2004.

[11] Tony D. May. Music and computers: The design and implementation of a musical genetic algorithm. Master's thesis, University of Kent, 2000.

[12] Carroll Morgan. *Programming from Specifications*. Prentice Hall, second edition, 1994.

[13] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 1999.

[14] D. Partridge and A. Galton. The specification of 'specification'. *Minds and Machines*, 5(2):243–255, 1995.

[15] D. Partridge and W.B. Yates. Data-defined problems and multiversion neural-net systems. *Journal of Intelligent Systems*, 7(1–2):19–32, 1997.

[16] Derek Partridge. The case for inductive programming. *IEEE Computer*, pages 36–41, January 1997.

[17] Rosalind Picard. *Affective Computing*. MIT Press, 1997.

[18] D. Rowland and F. Biocca. Cooperative design methodology: Genetic sculpture park. *Leonardo*, 35(2):193–196, 2002.

[19] G. Venturini, M. Slimane, F. Morin, and J.-P. Asselin de Beauville. On using interactive genetic algorithms for knowledge discovery in databases. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 696–703. Morgan Kaufmann, 1997.