CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Kent Academic Repository

A Design Framework for Metaheuristics

Colin G. Johnson
School of Computing
University of Kent
Canterbury, Kent, CT2 7NF
C.G.Johnson@kent.ac.uk

Published in *Artificial Intelligence Review*, 29(2), April 2008, pp163–178.

Abstract

This paper is concerned with taking an engineering approach towards the application of metaheuristic problem solving methods, i.e. heuristics that aim to solve a wide variety of problems. How can a practitioner solve a problem using metaheuristic methods? What choices do they have, and how are these choices influenced by the problem at hand? Are there sensible universal choices which can be made, or are these choices always problem-dependent? The aim of this paper is to address questions such as these in the context of a (soft) engineering design framework for the application of metaheuristics. The aim of this framework is to make explicit the choices which a practitioner needs to make in applying these techniques, and to give some guidelines for how metaheuristics might be tuned to problems by considering different problem- and solution-types.

Keywords: Heuristics, Optimization, Artificial Intelligence, Genetic Algorithms, Operational Research, Problem-Solving

1 Introduction

The aim of this paper is to introduce a *design framework* for the application of metaheuristic algorithms; that is, algorithms that aim to provide a heuristic search technique that can be applied to a wide class of problems. The aim of such a framework is to allow a pragmatic engineering approach to the application of metaheuristics to specific problems, by making explicit the decisions which have to be taken in tailoring the metaheuristic to the problem and making clear the different problem types and how they relate to these decisions.

The first main section of the paper reviews the idea of problem-solving using metaheuristics and introduces the idea of a design framework. The next three sections introduce the three main aspects of the framework: problem types, exploration of the search space, and solution types. This is followed by a conclusion which draws the various aspects together.

2 Problem Solving via Metaheuristics

Metaheuristics (Osman, 1996) provide a third approach to problem-solving, alongside exact methods and problem-specific heuristics. By problem-solving we typically mean the solution of problems via a search method over a well-defined search space. Canonical examples are standard operational research problems such as timetabling, scheduling, combinatorial optimization, et cetera. Other, interactive-type problems will also be considered later in the paper.

Many problems can be solved exactly by specific exact methods: for example there are various algorithms for calculating the shortest route through a graph (Wilson, 1996). However for many problems such exact methods cannot be used. Sometimes such methods are not known. Sometimes methods are known, but are impractical, perhaps because the method is computationally intractable when used on problems of realistic size—in such a

case, the computational complexity of exact methods means that other methods must been found that approximate the solution that could be found using exact methods.

Because of such difficulties, many problems are solved using problem-specific *heuristic* methods. Heuristic methods are not guaranteed to find an exact best answer to a problem; they may miss some good solutions. Nonetheless, a good heuristic will find good solutions to most problems most of the time. Clearly "good" and "most" depend on the particular requirements of the problem. One way to consider heuristics is as "rules of thumb" which suggest how to deal with non-pathological problems in a sensible fashion. Often heuristic methods will involve some element of randomness, in which case the performance from run-to-run may vary too.

An example of such a problem-specific heuristic method is the Lin-Kernighan method for the TSP (Lin and Kernighan, 1973). This makes a number of "sweeps" through the graph, each attempting to improve the current solution by a number of local changes. Each sweep through the graph finds the change which makes the best improvement; the nodes involved in this are marked and the sweep begun again to find another local improvement not involving those nodes. This is repeated until the nodes are exhausted, and then the whole process is repeated until no more improvement takes place. This gives a good method for finding such a route, but it can miss out on some larger-scale global changes which could modify the route for the better.

Some heuristic techniques aim to be more generic. For example, combinatorial optimization (Papadimitriou and Steiglitz, 2000) or heuristic methods for constraints (Freuder and Wallace, 1992) provide sets of methods that can be applied to a wide range of problems. One specific strategy that is commonly adopted in the application of these methods is to reduce a new problem to an already solved problem in one of these kinds, and then solve it using the known methods.

Despite the success of such heuristic methods where exact methods are not available, an individual heuristic still needs to be devised for each problem. Whilst much effort is likely to be dedicated to the production of heuristics for important or generic problems, many problems are too specialized to be worth the expenditure of effort to create a specific heuristic. Also one of the aims of computer science is to reduce the amount of individual effort which needs to be put into the solution of particular problems. Whilst computational efficiency is usually taken as synonymous with the amount of computational effort required to solve a problem on the computer, it is often the complexity of setting up the computer to solve the problem in the first place which provides the greatest challenge. For these reasons the creation of *metaheuristics* is important.

Earlier we suggested that problem-specific heuristics are "rules of thumb" for solving a particular problem. From a similar perspective metaheuristics can be seen as general "rules of thumb" for solving a wide range of problems. At a naïve level, at least, it is reasonable to assume that such approaches exist. Examples are easy to produce: take an existing solution-attempt and attempt to improve it by making a small change to it, or by combining it with another; never throw away the best attempt you have found so far; try generalizing an attempt; make an attempt more specific; introduce some random element into the process to redirect thinking; et cetera. Such general metaheuristic ideas have been used by writers such as Polya (1945) and De Bono (1990) in teaching people how to become better problem solvers or creative thinkers.

Metaheuristics have also been created by abstracting from heuristic method. This approach proceedings by analysing a number of heuristic techniques applied in different problems and abstracting out the common features. This then provides a framework that can be applied to a wider range of problems than any of the original problem-specific heuristics from which the metaheuristic has been abstracted.

Another fecund source of inspiration for metaheuristics has been abstraction from the

natural world. Nature exhibits a rich complexity, and there are many systems in the world which deal with that complexity, whether on a small timescale such as the reaction of the immune system to an infection, or on a longer timescale such as the evolution of a species to survive and reproduce (without planning or foresight) within a particular environmental niche. The ways in which these systems adapt and process information about themselves and their environment can be abstracted and used as the basis of a computational metaheuristic; for example Genetic Algorithms (Goldberg, 1989; Mitchell, 1996), swarm search methods such as Ant Colony Optimization and Particle Swarm Optimization (Bonabeau et al, 1999; Kennedy et al, 2001), problem-solving with Artificial Immune Systems (de Castro and Timmis, 2002), Ant Colony Optimization (Blum, 2005; Dorigo and Stützle, 2004) and Tabu Search (Glover, 1989, 1990). The aim of the work in this paper is not to address specific issues concerned with one of these algorithms as such, but to provide a general framework for applying these methods.

One danger with metaheuristic methods is that it is easy to forget that there are many of choices to be made in using such a metaheuristic well. It is easy to think that the process to be followed is to describe the problem in a way which is canonical for the metaheuristic at hand, run a program which implements the metaheuristic, and read off the proposed solution. However, there are many choices to be made in using a metaheuristic, and it is not clear how these choices can be made (though works such as the recent book by Goldberg (2002) attempt to give a theoretically-grounded yet pragmatic attempt at this; a similar point has been made in the context of mathematical programming methods, such as linear programming, by Williams Williams (1999)). In particular the relationship between these choices and problem-types needs to be made clearer.

The aim of this paper is to provide an (extensible) *design framework* for those choices. Such a framework will consist of making explicit those choices and giving pragmatic and theoretically-grounded advice about how these choices might be made. It will also challenge some of the default assumptions where it is usually assumed that there is no choice to be made. The aim of these choices will be to enhance the ability of practitioners to tune metaheuristics to their particular problems, by providing a framework for thinking about these choices and a number of methods to support that decision-making.

The aim of this is to provide a similar framework to that provided by Williams Williams (1999) for mathematical programming methods. In that book, he provides several lists of generic types of entities that play a role in building a model in mathematical programming: for example, a list of constraint types together with advice on how they can be turned into constraints of the kind handled by such methods, and a description of different ways in which models can be combined.

This framework consists of three areas:

Describing problems how problems are presented to the metaheuristic.

Moving through the search space how the metaheuristic explores the search space.

Solution types The different kinds of generic problems that are tackled by metaheuristics

Within each of these areas we explore the decisions that need be made by the "metaheuristic engineer" in adapting an off-the-shelf metaheuristic to their problem.

The approach taken is that of a "soft" engineering approach based on explicit enumeration of the choices that need to be made at each stage in the application of a metaheuristic. This is similar to the process used in methods such as Multiple Criteria Decision Analysis (MCDA) (Belton and Stewart, 2002); however, in the example in this paper the various dimensions of decision criteria are orthogonal to one another, representing decisions taken at different stages in the process of metaheuristic application. Therefore the MCDA methods that attempt to minimize conflicts between different decision criteria are less relevant in this

framework than they would be in a framework with many interactions between the criteria. However, such methods (in particular the Analytic Hierarchy Process (Saaty, 1980)) could sensibly be applied to other areas in metaheuristic engineering that we do not consider here, for example the choice of the core metaheuristic technique to be used in a particular situation.

It is important to note that the framework is *extensible*. A number of choices which seem to be of particular significance have been discussed here. However more choices can be readily incorporated into an extended framework, and the scope of choices within each choice extended.

3 Describing Problems

In the previous section we motivated the notion of metaheuristic using operational research-style problems, as befits the history of the subject. In this section the notion of problem will be broadened to encompass any problem which can be approached by a metaheuristic approach. In particular we will focus on problems where the search space is well defined, and where solving the problem can be viewed as searching that space. Orthogonally, we are interested in those problems which do not admit an exact method of solution, and for which the notion of an approximate or suboptimal solution is valid. Typically, these problems will have some notion either of an objective function whereby a numerical solution quality can be calculated, or at least a way in which to compare pairs of solutions. We will refer to this as *fitness* in this paper.

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 approach to this is to suggest that this process consists of 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: 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 (Partridge, 1992, 1997; Partridge and Yates, 1997; Partridge and Galton, 1995). 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."* (Partridge and Yates, 1997)

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.

For example, in attempting to solve a problem concerned with recognizing handwritten digits, the problem-solver might attempt to solve this problem by specifying the various components that should be recognised by a solution to the problem. For example, the solution should be able to identify a shape that contains two regions, and classifies this as a figure 8. However, the simpler way to present these kinds of problems is simply to gather a training set of images tagged with the number that they represent.

However there is a converse danger when dealing with specification-defined problems, which could be termed a *data bottleneck*. This problem arises when the user takes a specification and trains the metaheuristic using a sample of training data which satisfies the specification; this is a very common practice in machine learning. Again, the problem has been presented to the metaheuristic in a way which does not respect the structure of the problem.

For example, consider a metaheuristic learning to solve a control problem for some industrial plant, where there is a safety constraint on the temperature of some component: for example, it should never have a temperature greater than $60°C$.. The data-defined way to deal with this would be to check whether this constraint is satisfied each time a test case is run. A better way, which would be of particular relevance to safety-critical systems, would be to (automatically) analyse the model to ensure that the constraint always holds; this is an example of what we mean by *specification-defined*.

It would seem to be ideal to present both data-defined and specification-defined problems to the metaheuristic in a way which retains this structure. 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 the class of problems which are *interactively-defined*. 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 an operator 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 recent books by Bentley and Corne (Bentley, 1999; Bentley and Corne, 2002) 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? For example, an algorithm that is meant to evolve a picture or a musical melody.

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 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. (1997) 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 problems 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. However neither of these seem to get at the core of what an interactively-defined problem is.

The discussion of problem types has been concerned with problems which are defined directly by the user of the system (whether in advance or interactively). A further class of problems can be defined via interaction amongst a set of agents. A good example of this is the use of genetic algorithms in a coevolutionary framework where the fitness of one individual depends on the state of others (Hillis, 1990; Potter and De Jong, 1994; Paredis, 1996; Olsson, 1998; Rosin and Belew, 1996). Another related class of problems is given by work on *endogenous fitness* (Menczer et al, 1994; Menczer and Belew, 1996), where there is no explicit fitness function but individuals interact within a complex environment related to the problem.

We have defined three problem-types: data-defined, specification-defined, and interactively-defined. Some problems may involve aspects of each. To summarize this section so far 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 an interesting pattern with the pen attached to it.

Specification and data The robot should chase another robot, whilst not leaving a pre-defined area.

Specification and interactive The robot should trace out an interesting pattern, 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.

It would seem natural that in introducing such problems to metaheuristics each of these three problem-types need to be treated in a way which is natural for that particular problem. 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 solution-attempts in the population.

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. Traditionally such problems are presented to metaheuristics by creating a number of fitness cases which are compatible with the specification. For example, in a safety-critical system, solutions generated by the metaheuristic might be tested against a sample of safe and unsafe scenarios, rather than being verified for safety directly against the specification.

This leads to a number of problems; in particular the solutions can become *overfitted* to the fitness cases and not generalize, and there is no guarantee given to the user that the specification is satisfied outside of the particular fitness cases (Johnson, 2002b). Instead we need to find ways of assessing directly whether a solution-attempt satisfies the specification or not.

One approach to this is through the use of computational techniques such as *static program analysis* (Nielson et al, 1999) (which checks that certain properties of a program are satisfied regardless of input data) or model checking (Clarke Jr. et al, 1999; Huth and Ryan, 2000) (which checks whether certain logical formulae about the entities in the solution and their relationships over time are always satisfied) in the determination of fitness. 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. Such techniques have been applied in the evolution of sorting algorithms (Huelsbergen, 2000), robot control systems (Johnson, 2004) and layout problems (Johnson, 2002a).

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; for example 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. (2002) 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. Thirdly some of the analysis approaches give additional information when they fail to be satisfied. For example a failed attempt at model-checking provides a counter-example to the set of logical statements being checked; these could be subsequently analysed to guide the search.

Finally, interactively-defined 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 problem-types were *consistent* in their fitness evaluation, 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 (May, 2000) 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 (Rowland and Biocca, 2002) or a performance setting (Biles and Eign, 1995; Biles, 1998). This could be enhanced by the use of affective computing techniques (Picard, 1997) to directly assess user's affective response to individuals.

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. 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 (Morgan, 1994; Derrick and Boiten, 2001). 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. For example, such techniques have been applied (Johnson, 2004) in using evolutionary computing to automatically generate programs which solve a robot control task whilst also guaranteeing that the robot will operate within a safe region. This is done by evolving programs using a genetic programming-style system, where the fitness of a program is a combination (in this case a weighted sum, though potentially any multi-criterion optimization method could be used) of a data-driven test for the control task, together with a calculation as to whether the program *always* stays within a safety bound.

This section 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 placed into the context of *metaheuristics*, and the way in which such metaheuristics can treat problems of different kinds has been discussed. In particular it is argued that metaheuristics provide a natural framework in which to solve problems which have a number of aspects which are naturally defined in different ways.

This choice between the various problem-types, and the different ways of evaluating fitness which fit naturally with those problem types, will form the first component of the design framework.

4 Measuring Fitness and Moving Through the Search Space

The previous section discussed how problems can be presented to metaheuristics, in particular focusing on the idea that metaheuristics can be applied in a way which respects the structure of the problem. This argument can be extended from the initial setup of the problem to the way in which the metaheuristic progresses from initialization to solution.

A metaheuristic improves the solution-attempts in its population (which might be a degenerate population with one member as in hillclimbing algorithms) by moving that population to areas of the search space which are inferred to be likely to be better than the current position of the population in search space. The processes which effect these changes can be called *moves* (Tuson, 1999) in the search space.

These moves can take a number of forms. Some move-types consist of changing a single individual. For example in steepest-ascent hillclimbing (Winston, 1992), the individual solution-attempt will be moved by sampling a number of points in its neighbourhood and moving to the point which gives the largest fitness improvement. In genetic algorithms the mutation operator can be seen as a move which makes a number of random changes to the genotype of individuals in the population with the aim of exploring local changes in phenotype space (amongst other things; mutation has a role in ensuring that total convergence at an allele cannot occur, for example). For example, if we consider a system for timetabling within a school or university, a mutation operator would make small changes to the timetable: swapping two classes, exchanging one tutor for another qualified in the same subject. These small changes only affect a small number of students and tutors, and so can be used for "local" exploration. By contrast, sometimes small changes in the representation can make a large change to the result. For example, if the solution to a problem is encoded as a number, then flipping the most significant bit can change the number encoded by the population member hugely (hence the motivation to use representations that have a more uniform effect, such as Gray-coded numbers).

Other move-types involve the combination of information from multiple members of the population. A canonical example here is the recombination move in genetic algorithms, which takes the genotypic representation of one individual and exchanges parts of it with that of another individual. In population-based incremental learning and similar methods (Baluja, 1994; Baluja and Caruna, 1995; Abbattista and Dalbis, 1998; Robillard and Fonlupt, 2000) the whole population is involved in moves: the population is summarized as a vector of weights and new individuals created by regeneration from that vector to give a new population with statistical properties the same as the original population. For example, consider the application of a metaheuristic to design a mechanical device such as a flow nozzle, where the representation is by a sequence of floating-point numbers representing the diameter at n positions along the tube. Each individual population member would be a vector of floating-point numbers; the population summary would be of the same type, with each number representing the average of the diameters at that position across the population members, with perhaps a second vector to represent the variance at each position. This distribution would then be used to recreate a new population of vectors.

It would seem to be important to choose these move operators carefully. Is there a way of doing this so that the moves work well, without needing to design problem-specific moves for each new problem (or at least a way of assessing automatically-generated move operators)?

An important distinction in this discussion is the relationship between the action of a move in the space of encodings of the problem (what in genetic algorithms is called *genotype-space*) versus the effects of that move on the underlying problem (*phenotype-space*). Sometimes the mapping from one to the other is simple, in other cases the connection between the two can be complicated, with e.g. multiple regions of genotype-space mapping

to the same point in phenotype-space. A number of ways of measuring this have been explored by Rothlauf (2002).

Generic move operators such as mutation and crossover in genetic algorithms are defined in genotype space. However it seems implicit that there is some connection between these genotype-space moves and how they transform the population phenotypically. This is reflected in the following description by Goldberg of crossover:

> *"Thus, the action of crossover with previous reproduction speculates on new ideas constructed from the high-performance building-blocks (notions) of past trials."* (Goldberg, 1989, p. 13)

It is interesting here to note the implied equivalence between the genotypic concept of "building-blocks" and the phenotypic concept of "notions". He goes on to describe the crossover process as "exchanging of notions to form new ideas" (Goldberg, 1989, p. 14). There is an implication here that the genotypically-defined crossover process implies some kind of exchange of ideas in the phenotype—though it is recognised that this may happen in "highly nonlinear and complex ways" (Goldberg, 1989, p. 13).

A similar idea can be found for the concept of mutation. Goldberg describes mutation as "an insurance policy against premature loss of important notions" (Goldberg, 1989, p. 14). Another example is given by Rothlauf: "[m]utation operators are important for local search" (Rothlauf, 2002, p. 18). Again mutation is being described in terms of a desired phenotypic trait (avoiding loss of "notions", "local search") as well as in terms of the direct effect it has on the genotype of individuals.

If we want to make a design framework for making problem-centered choices for applying a metaheuristic to a particular problem, then it would seem to be important to consider the effect of the move operators chosen on the phenotype (typically this effect will be indirect, because the moves will be defined genotypically). If this is not considered the statement at the beginning of this paragraph seems meaningless: the problem at hand is defined by the structure of the phenotype-space, and so to make a rational choice of how to move in a way which respects the structure of the problem must involve some consideration of how the moves affect points in phenotype-space. This is not to say that the moves must be *defined* directly in phenotype-space. To the contrary, the moves will still typically be defined in terms of transformations of the genotype; but the choice of which moves to use will be done with respect to the changes the make on the phenotype.

Another way of phrasing the above discussion is in terms of *syntax* and *semantics*. As in conventional programming, the work done by a metaheuristic is the the syntactic domain; the individuals and moves are defined in syntactic terms. Nonetheless the reason for doing this is to have a semantic effect, that is to find an individual which solves the problem at hand. Therefore it seems reasonable that the syntactic representations and moves should correlate with moves which are meaningful in terms of the semantics of the problem.

This emphasis on the semantics of move operators allows us to define those operators primarily in terms of their phenotypic effect. So for example a mutation operator in genetic algorithms can be defined (in a problem-free way) as an operator which makes small exploratory changes to individuals. A recombination operator can be similarly defined as one which brings together aspects of two distinct solutions

It is also important to note that the choices of move operators and the representation of the solutions as members of genotype space are essentially two aspects of the same problem. Any change in representation can be effected by retaining the original representation and changing the move operators so that they have the same effect as the original move operators would have on the new representation.

One way to make such choices is by reasoning about the problem domain and choosing move operators which have a known effect on members of the phenotype. This can be found

e.g. in the work of Martin and Otto (1996) on the TSP, which incorporates local moves known as *4-opt* moves within a simulated annealing framework.

However for many problems this approach is not practical. This may be because the relationship between the genotype and phenotype is complex, so it is difficult to predict the effects of genotypically-defined moves on the phenotype. Alternatively this may be a practical matter: there is not enough human effort available in order to design and test such moves. For both of these reasons methods of automatically designing or choosing move operators and representations is useful.

Three strategies can be employed. The first is statistical analysis of proposed indirect fitness measurements. For example in recent work on metaheuristic counterexample-search in pure mathematics (Johnson, 2003) the fitness cannot be measured directly. Instead it must (by the nature of the problem) be assessed *indirectly* via the calculation of *invariants* on members of the population (which consists of geometrical objects). There is a choice of such invariants, and the problem at hand is a two-criteria optimization problem where the aim is to minimize one fitness criterion whilst whilst maximizing another.

In order to choose between the invariants, each was applied to a large set of randomly chosen examples from the search space. The two fitness criteria were then measured for each of this set of randomly chosen invariants, and each of the two fitness measures calculated for each example. For each invariant correlation coefficient was then calculated between the datasets corresponding to the two fitness measures, and the invariant chosen which had the least correlation between the measures. This was chosen so that the search algorithm can search for both criteria with as little interference as possible between them. Such an approach has potential to be applied more broadly, for example in the choice of representations for multi-criterion optimization problems.

The second strategy is a *target analysis* of proposed move operators and associated representations. A *target analysis* (Laguna and Glover, 1993; Glover, 1986) is a way of tuning (meta)heuristics to a particular problem or set of problems. The target analysis begins with the construction of a *target solution*. If the aim is to solve many similar problems, then the target solution might be a one example from that set of problems, which has been analysed with more effort (computational or human) than can be applied normally. So for example the target solution to the chosen problem might be created via an enumerative search, which might be feasible for a single problem but not for day-to-day use. Alternatively there may be some examples in the set of problems which are solvable using some analytic technique; these solutions could then be used as target solutions. A third type of target solution are solutions to a related, easier problem: for example, in choosing which operators to apply for a hard problem involving a search space structured as a graph, simpler problems which also involve searching a graph might be investigated.

This target solution is then used to tune a metaheuristic to the solution of that problem. For example, the inverse of move operators might be tested on the target solution, to see if the solution is likely to be found by making a move using that move operator. Essentially the metaheuristic is "reverse engineered" from the target solution.

This tuned metaheuristic is then applied to the other problem(s). The hypothesis underlying this is that the tuned metaheuristic will work better on such related problems than an arbitrary metaheuristic would; we have added problem-specific information into the metaheuristic, albeit indirectly and possibly imperfectly via related problems. This has been shown empirically (Laguna and Glover, 1993; Glover, 1986) to have an effect on the ability of metaheuristics to solve problems. Clearly there are a number of caveats to this method—in particular the notion of "related problems" is a difficulty: problems which seem related to the practitioner might not have an underlying structural similarity. Also, the target solution may be untypical, particularly if the reason it can be solved easily also implies that it has some structural uniqueness within the problem-set which is also exploited in

its solution by the metaheuristic—but overall it seems to be a valuable method of putting problem-specific information into a metaheuristic.

A third, more formal, approach is to develop problem-specific operators by exploiting the results of *forma analysis* of heuristics. These ideas have been applied to hillclimbing Tuson (1999), evolutionary Gong and Tuson (2006, 2007) and particle-swarm Gong and Tuson (2008) search methods.

The idea of forma analysis Radcliffe (1991, 1994) is to describe the search space for a particular problem in terms of a set of *equivalence relations*, that is, subsets of the search space that have a feature in common. A search space can then be described by a set of these equivalence relations, called a basis. Such a basis provides a way of describing points in the search space using problem-relevant features.

Radcliffe Radcliffe (1994) describes a number of generic operators that can be combined with a basis to form a problem-specific operator. For example, he defines a mutation operator in terms of a basis that makes a change to the minimum-possible change in the number of equivalence classes that are changed by the operator. Other such operators are provided for moves such as recombination of two individuals. By combining the problem-specific information encoded in the choice of basis with the generically defined operators, problem-specific operators can be automatically generated.

These three techniques—choice of indirect fitness measures, target analysis, and construction of operators via techniques based on forma analysis—provide the beginnings of a rational approach to operator design. Combined with *ad hoc* ideas for operator design

This section has given us another contribution towards a design framework for metaheuristics based on using information about problems and their encodings to guide the choice of moves make by the metaheuristic in its search.

5 Solution Types

An earlier section of the paper discussed the different ways in which problems can be presented to metaheuristics. In this section the final part of the framework is introduced: the choice of the kinds of solution which will be obtained from the metaheuristic.

In the discussion of problem types we discussed the notion of a *bottleneck*. In the problem context a bottleneck is the conversion of a problem into a problem of a different type so that it can be approached via a metaheuristic method. We have argued above that this is a bad thing, and that a preferable alternative is to adapt the metaheuristic so that it more naturally accepts problems of that type.

A similar argument can be made for the output of the process as well as the input. A metaheuristic should deliver the type of solution which is natural for the problem, rather than forcing the user to reorganize the type of solution they require so that it is forced into a particular solution framework.

An example of this is the dominance of *optimization* as a solution-type. Many solutions are optimization-defined: the natural expression of the solution is as an optimal value of some quantity. However many problems are not; nonetheless it is common, when approaching a problem with a metaheuristic such as a genetic algorithm, to ask "what needs to be optimized?".

Nonetheless constructing an entirely new solution-type for every problem is excessive; there is a lot of commonality between problems. At the input end we dealt with this by defining a number of broad problem-types: data-defined, specification-defined, and interactively-defined (leaving scope for additional types to be defined; there is no argument that this is an exhaustive list). Similarly with solution-types we can say that problems can be placed into a number of solution-types: e.g. optimization, search, multimodal optimization. This list is short: would it be possible to add new solution-types to this list which provide good

natural representations for the types of solutions required by some real-world problems? Is it then possible to adapt metaheuristics so that they can hunt for solutions of this type without forcing the solution through a bottleneck?

An example of a new generic solution-type is *qualitative example finding* or *coverage* (Johnson, 2001b,a). The aim of this is to find one concrete example for each value of some property within a large space; for example one route to each of many exits in a complex maze, or one example of each environment which causes an autonomous agent to exhibit one of a range of behaviours. This is not a problem which can be naturally solved using optimization (because as soon as an example as been found there is nothing to optimize) or a focused search (we are looking to cover the search space not narrow down on one solution). For example, such a metaheuristic solution-type could be used to generate test-suites for software, a web-search technique that looked for qualitatively different examples of objects represented by a common search term, or a range of qualitatively different features to use in a classification algorithm.

This ties in with ideas which emphasise that evolutionary algorithms are not primarily optimization algorithms *per se* but are exploration/exploitation algorithms which can search a space in a robust fashion (De Jong, 1993; Harvey, 1997).

This idea of choosing solution-types provides the final part of our design framework. The availability of a number of other "generic" solution types, as typified by the coverage example above, would increase the applicability of metaheuristic methods.

6 Summary and Conclusions

A simplistic view of metaheuristics is that they are *automatic* problem solving machines. In this view, problems are fed into the metaheuristic which processes the problem and gives an output.

The discussion above has shown that this is an *over*simplistic view of metaheuristics. However this additional thinking about how the user of a metaheuristic can involve themselves in the various processes has the potential to improve the performance of those metaheuristics by putting more information about the problem into the process.

This meta-level discussion of metaheuristics has illustrated that the application of a metaheuristic to a particular problem can be seen as involving a number of *choices*. A guiding principle in the above has been to say that these choices should reflect aspects of the problem being solved; the metaheuristic should come to the problem rather than the problem being twisted to fit the metaheuristic. Careful use of a framework for making these choices can avoid the problem of *bottlenecking* where a problem is forced into a representation which is not natural for it before it is presented to the metaheuristic, make clearer the relationship between moves in genotype-space and their consequences in phenotype-space, and broaden the range of solution-types available to practitioners.

The broader scope which is available under this perspective can be applied to the various metaheuristic methods which are available. The influence of this larger view of the possible choices in the context of genetic algorithms is illustrated via the example of genetic algorithms in figure 1. This shows the change from a "traditional" perspective on genetic algorithms to a perspective based on a larger number of choices, allowing the algorithm to be more carefully tuned to the problem at hand.

References

Abbattista F, Dalbis D (1998) The scout algorithm to explore unknown spaces. In: IEEE International Conference on Evolutionary Computation, IEEE Press, pp 705–708

Problem types	Data-defined
Fitness assessment	By measuring the performance of test-runs of individuals against training data
Moves in search space	Genotypically-defined mutation and recombination.
Solution types	Optimization or search.

Problem types	Data-defined, specification-defined and interactively defined
Fitness assessment	By measuring the performance of test-runs of individuals against training data, interactively with the user, by checking how well the output matches the specification, by making one of a number of indirect measures of the fitness.
Moves in search space	Genotypically defined operators which may have been targeted carefully at the problem.
Solution types	Optimization, search, coverage, other types of solution

Figure 1: Broadening the scope of metaheuristics and clarifying design decisions through a design framework, using genetic algorithms as an example.

Baluja S (1994) Population-based incremental learning. Tech. Rep. CMU-CS-94-163, Carnegie Mellon University

Baluja S, Caruna R (1995) Removing the genetics from the standard genetic algorithm. In: Twelfth International Conference on Machine Learning, Morgan Kaufmann, pp 38–46

Belton V, Stewart T (2002) Multiple Criteria Decision Analysis. Kluwer Academic Publishers

Bentley PJ (ed) (1999) Evolutionary Design by Computers, Morgan Kaufmann

Bentley PJ, Corne DW (eds) (2002) Creative Evolutionary Systems, Morgan Kaufmann

Biles JA (1998) Interactive GenJam: Integrating real-time performance with a genetic algorithm. In: Proceedings of the 1998 International Computer Music Conference

Biles JA, Eign W (1995) GenJam Populi: Training an IGA via audience-mediated performance. In: Proccedings of the 1995 International Computer Music Conference

Blum C (2005) Ant colony optimization: Introduction and recent trends. Physics of Life Reviews 2:353–373

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence. Oxford University Press

Clarke Jr EM, Grumberg O, Peled DA (1999) Model Checking. MIT Press

De Bono E (1990) Lateral Thinking. Penguin

de Castro LN, Timmis J (2002) Artificial Immune Systems: A New Computational Intelligence Approach. Springer

De Jong K (1993) Genetic algorithms are NOT function optimizers. In: Whitley L (ed) Foundations of Genetic Algorithms 2, Morgan Kaufmann, pp 5–17

Derrick J, Boiten E (2001) Refinement in Z and Object-Z. Springer

Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press

Freuder E, Wallace R (1992) Partial constraint satisfaction. Artificial Intelligence 58(1–3):21–70

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13:533–549

Glover F (1989) Tabu search—part I. ORSA Journal on Computing 1(3):190–206

Glover F (1990) Tabu search—part II. ORSA Journal on Computing 2(1):4–32

Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley

Goldberg DE (2002) The Design of Innovation. Kluwer

Gong T, Tuson A (2006) Formal descriptions of real parameter optimisation. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, IEEE Press, pp 2119–2126

Gong T, Tuson AL (2007) Enhanced forma analysis of permutation problems. In: Proceedings of the Ninth Annual Genetic and Evolutionary Computation Conference (GECCO), pp 923–930

Gong T, Tuson AL (2008) Forma analysis of particle swarm optimisation for permutation problems. Journal of Artificial Evolution and Applications 2008, article ID 587309

Harman M, Hu L, Hierons R, Baresel A, Sthamer H (2002) Improving evolutionary testing by flag removal. In: Langdon WB, Cantu-Paz E, Mathias K, Roy R, Davis D, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke E, Jonoska N (eds) Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann

Harvey I (1997) Cognition is not computation: Evolution is not optimisation. In: Gerstner W, Germond A, Hasler M, Nicoud JD (eds) Proceedings of the Seventh International Conference on Artificial Neural Networks, Springer-Verlag, Lecture Notes in Computer Science 1327, pp 685–690

Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42:228–234

Huelsbergen L (2000) Abstract program evaluation and its application to sorter evolution. In: Proceedings of the 2000 Congress on Evolutionary Computation, IEEE Press, pp 1407–1414

Huth M, Ryan M (2000) Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press

Johnson CG (2001a) Finding qualitative examples with genetic algorithms. In: John R, Birkenhead R (eds) Developments in Soft Computing, Springer, pp 92–99

Johnson CG (2001b) Understanding complex systems through examples: a framework for qualitative example-finding. Systems Research and Information Systems 10:239–267

Johnson CG (2002a) Deriving genetic programming fitness properties by static analysis. In: Foster J, Lutton E, Ryan C, Tettamanzi A (eds) Proceedings of the 2002 European Conference on Genetic Programming, Springer, pp 298–307

Johnson CG (2002b) What can automatic programming learn from theoretical computer science? In: Yao X, Shen Q, Bullinaria J (eds) Proceedings of the 2002 UK Workshop on Computational Intelligence, pp 89–95

Johnson CG (2003) A design framework for evolutionary algorithms. PhD thesis, University of Kent

Johnson CG (2004) Genetic programming with guaranteed constraints. In: Lotfi A, Garibaldi JM (eds) Applications and Science in Soft Computing, Springer, pp 95–100

Kennedy J, Eberhart RC, Shi Y (2001) Swarm Intelligence. Morgan Kaufmann

Laguna M, Glover F (1993) Integrating target analysis and tabu search for improved scheduling systems. Expert Systems with Applications 6:287–297

Lin S, Kernighan B (1973) An effective heuristic algorithm for the traveling salesman problem. Operations Research 21:498–516

Martin O, Otto S (1996) Combining simulated annealing with local search heuristics. Annals of Operations Research 63:57–75

May TD (2000) Music and computers: The design and implementation of a musical genetic algorithm. Master's thesis, University of Kent

Menczer F, Belew RK (1996) Latent energy environments. In: Belew RK, Mitchell M (eds) Adaptive Individuals in Evolving Populations, Addison-Wesley, Santa Fe Institute Studies in the Sciences of Complexity, pp 191–208

Menczer F, Willuhn W, Belew RK (1994) An endogenous fitness paradigm for adaptive information agents. In: Proceedings of the Third International Conference on Information and Knowledge Management

Mitchell M (1996) An Introduction to Genetic Algorithms. Series in Complex Adaptive Systems, Bradford Books/MIT Press

Morgan C (1994) Programming from Specifications, 2nd edn. Prentice Hall

Nielson F, Nielson HR, Hankin C (1999) Principles of Program Analysis. Springer

Olsson B (1998) A host-parasite genetic algorithm for asymmetric tasks. In: Nédellec C, Rouveroi C (eds) Machine Learning : ECML-98, Springer

Osman I (1996) Meta-heuristics. Kluwer Academic Publishers

Papadimitriou CH, Steiglitz K (2000) Combinatorial Optimization: Algorithms and Complexity. Dover Publications

Paredis J (1996) Coevolutionary computation. Artificial Life 2(4):355–375

Partridge D (1992) Engineering Artificial Intelligence Software. Intellect Books, Oxford

Partridge D (1997) The case for inductive programming. IEEE Computer pp 36–41

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

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

Picard R (1997) Affective Computing. MIT Press

Polya G (1945) How to Solve It: A New Aspect of Mathematical Method. Princeton University Press

Potter MA, De Jong KA (1994) A cooperative coevolutionary approach to function optimization. In: Parallel Problem Solving from Nature III, Springer, pp 249–257

Radcliffe NJ (1991) Equivalence class analysis of genetic algorithms. Complex Systems 5(2):183–205

Radcliffe NJ (1994) The algebra of genetic algorithms. Annals of Mathematics and Artificial Intelligence 10:339–384

Robillard D, Fonlupt C (2000) A shepherd and a sheepdog to guide evolutionary computation. In: Fonlupt C, Hao JK, Lutton E, Ronald E, Schoenhauer M (eds) Artificial Evolution 1999, Springer, pp 277–291, Lecture Notes in Computer Science 1829

Rosin CD, Belew RK (1996) New methods for cooperative coevolution. Evolutionary Computation 5:1–30

Rothlauf F (2002) Representations for Genetic and Evolutionary Algorithms. Springer/Physica-Verlag

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

Saaty TL (1980) The Analytic Hierarchy Process. McGraw-Hill

Tuson AL (1999) No optimisation without representation. PhD thesis, University of Edinburgh

Venturini G, Slimane M, Morin F, Asselin de Beauville JP (1997) On using interactive genetic algorithms for knowledge discovery in databases. In: Bäck T (ed) Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, pp 696–703

Williams HP (1999) Model Building in Mathematical Programming, fourth edition edn. Wiley

Wilson R (1996) Introduction to Graph Theory. Addison-Wesley-Longman, fourth Edition

Winston PH (1992) Artificial Intelligence. Addison-Wesley