# Introduction: Problem Solving, EC and EMO

Joshua Knowles[1], David Corne[2], and Kalyanmoy Deb[3]

[1] School of Computer Science, University of Manchester, UK
j.knowles@manchester.ac.uk
[2] School of Mathematical and Computer Sciences (MACS), Heriot-Watt University, Edinburgh, UK dwcorne@macs.hw.ac.uk
[3] Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology, Kanpur, India deb@iitk.ac.in

**Summary.** This book explores some emerging techniques for problem solving of a general nature, based on the tools of EMO. In this introduction, we provide background material to support the reader's journey through the succeeding chapters. Given here are a basic introduction to optimization problems, and an introductory treatment of evolutionary computation, with thoughts on why this method is so successful; we then discuss multiobjective problems, providing definitions that some future chapters rely on, covering some of the key concepts behind multiobjective optimization. These show how optimization can be carried out separately from subjective factors, even when there are multiple and conflicting ends to the optimization process. This leads to a set of trade-off solutions none of which is inherently better than any other. Both the process of multiobjective optimization, and the set of trade-offs resulting from it, are ripe areas for innovation — for new techniques for problem solving. We briefly preview how the chapters of this book exploit these concepts, and indicate the connections between them.

## 1 Overview

> Intellectual activity consists mainly of various kinds of search.
> *Alan M. Turing, 1948*

When we say that computers can solve problems, it is a sort of half-truth, to be taken with a medium-sized pinch of salt. It is manifestly true that computers solve problems when they almost autonomously carry out everyday tasks involving communication, auditing, logistics and so forth, and computers even act somewhat more 'intelligently' when they do such tasks as controlling an automatic transmission, or making a medical diagnosis, in which they may even exhibit a computerized form of learning. But it is also true that computers are not very autonomous in solving *new* problems. Much human input and human innovation still goes into the process of solving difficult problems (designing a cable-stayed bridge, finding novel drug interventions, brokering international peace initiatives), a state of affairs that is likely — and desirably so — to continue for some time to come.

In this book, we consider an area in computer science which is at the forefront of techniques for solving difficult problems. Evolutionary computation — that is, methods that resemble a process of Darwinian evolution — can be used as a way to make computers 'evolve' solutions to problems, which we, as humans, do not ourselves know how to solve. By giving computers the capability of searching for their own answers to problems — through huge spaces of possibilities, in a very flexible way that goes beyond numerical methods — innovative and intelligent-seeming solutions and actions can be produced.

Much of evolutionary computation is concerned with optimization. For a human to use evolutionary computation to solve a new optimization problem, very little is required. This is where the flexibility of EC comes from: one must only provide the computer with (i) some way of representing or even 'growing' candidate solutions to the problem, and (ii) some function (or method) for evaluating any candidate solution, estimating its goodness on a numerical scale. Enormous varieties of problems can be stated succinctly in this way, from electronic circuits to furniture designs, and from strategies for backgammon to spacecraft trajectories. And the boon of evolutionary computation (though not one hundred percent realized) is that it turns computers into almost universal problem solvers which can be used by anyone with even minimal computational/mathematical competence. Of course, many problems are fundamentally intractable, in the sense that we cannot hope to find truly optimal solutions, but this does not really limit the uses of EC, but makes it more useful, since its strength is in finding the best solution possible given the time allowed.

But, while we just said that optimization is widely applicable, it is but a subset of a larger and even more flexible method of problem solving: multiobjective optimization (MOO). The drawback of standard optimization (let's call it single-objective optimization or SOO) is the requirement for a function that can score each and every candidate solution in terms of a single objective number. Many problems exist that are not so easy to state in terms of a single function like this. Humans are generally much more comfortable with, and used to, thinking in terms of aims and objectives *plural*, when stating a problem. And, while it is possible, in principle, for people to combine their aims in some over-arching function by weighting or ordering them by importance, in practice, different aims and objectives are often not measured in the same units, on the same scales, and it is often nigh-impossible to state the importance of different objectives when one has seen no solutions yet! More fundamentally, many of the methods for combining functions together into a single one, necessarily miss potentially interesting solutions. And many methods are very difficult to use because a small change in weights, gives a totally different solution. Therefore, it would be great if one could exploit the power of evolutionary algorithms, but use them to search for solutions even when the aims and objectives cannot be boiled down to a single function.

We are being purposely obtuse here, of course, because such methods already do exist in the field of *evolutionary multiobjective optimization (EMO)*, and they have been growing more and more effective over the past twenty years or so. An EMO algorithm is, loosely-speaking, one in which objectives are treated independently, and a *set* of optimal trade-offs (called Pareto optima) is sought, rather than a single optimum. But we introduced the field in this roundabout way to emphasise the point, made above, that computers solve problems — difficult problems at least — only in concert with humans. Humans are still the ones who generally own the problems and understand something about what they desire and hope for in a solution. So,

one of the key hurdles to problem solving with computers is to be able to formulate a problem in such a way that a computer can actually solve it, and a human is happy with the solution. EMO has this ability in spades, so rather than being a mere branch of EC, it actually represents a major step forward generally in computer problem solving.

Where this book advances further in terms of problem solving with computers, and problem solving specifically using the tools of evolutionary multiobjective optimization, is in examining the critical area of how to exploit the greater flexibility of search afforded by a multiobjective optimization perspective. While other books and articles on EMO [6, 7, 5] have given a thorough grounding in the development of EMO techniques, and have been formidable advocates of its benefits, it is only relatively recently that a groundswell in terms of researchers confidently exploiting EMO tools to new and innovative ends has really been apparent. It is on this we concentrate.

The new uses of EMO do not represent a step-change, but a gradual realization that there are few hard-and-fast rules in solving problems with the technique. Thus, researchers have begun to ask themselves such things as what would happen if I took away a constraint and treated it as an objective, what would happen if I had a problem where I had one objective but it seemed possible to decompose it into several, what would happen if I had some different functions which were inaccurate proxies for a true ideal objective function? In this book, we see how current research is dealing with these questions and further we see valuable products of this exploration. We see here that, ironically, EMO is very useful in *coevolution*, an area characterized by problems that have *no* formal objective function at all (evaluation occurs only by competitions). We see it helps in traditional SOO problems, where it speeds up search. We see it put to numerous uses in ill-posed problems, especially those in machine learning. Along the way, the chapters also consider the important issue of how to analyse and exploit the *sets* of solutions that are obtained from EMO, both in terms of decision making (i.e., usually choosing one final solution) or of learning from the set of trade-offs. And the development of EMO methods with respect to their scalability to larger problems with more objectives is considered, and supports the ideas proposed throughout the book.

In this chapter, we seek to do two jobs. First, to preview the book, which we have partly done, but which we continue in Sec. 4. Secondly, to cover some bases for any readers who might be unfamiliar with the fundamental concepts whose knowledge is assumed in some of the chapters, we give some appropriate introductory material. To these ends, Sec. 2 recalls the formal definition of a problem, including, in particular, an optimization problem. Hard problems, evolutionary algorithms, and the use of the latter on the former, are then discussed. Sec. 3 deals with multiobjective optimization, giving definitions for Pareto optimality and related issues. Then, Sec. 4 is a rundown of the four parts the book is divided into, and provides summaries of each of the chapters that make it up. Finally, Sec. 5 briefly concludes this introduction chapter.

# 2 Problems and Solution Methods

## 2.1 Optimization Problems

Problems, in computer science, are both abstract and precise. They are abstract in the sense of describing a whole class of *instances*; they are nevertheless precise in the sense that both the inputs and the solution of a problem instance are members of well-defined (mathematical) sets.

Specifically, a problem consists of: (i) a set of instances, where this set can be defined by either listing it exhaustively (enumerating it), or, much more usually, by specifying all the givens that define the form of an instance; and (ii) a set of solutions, being a definition of the entities that comprise a valid solution and the criteria for accepting it.

For example, we could define a sorting problem. A valid instance could be defined as any finite set of positive integers; a valid solution as an ordering of the input set that is strictly increasing.

An *optimization problem* is just a problem where the solution part of the problem is defined in terms of a function (the objective function), which is to be maximized or minimized.

**Definition 1** *An optimization problem is specified by a set of problem instances and is either a minimization problem or a maximization problem.*

From here onwards we will consider only minimization problems. In mathematical parlance, this is done 'without loss of generality', i.e., everything we say is true of both minimization and maximization problems, as long as we replace 'smallest' with 'largest' and such like.

**Definition 2** *An instance of an optimization problem is a pair $(X, f)$, where the solution set $X$ is the set of feasible solutions and the objective function $f$ is a mapping $f : X \to \Re$. The problem is to find a globally optimal solution, i.e., an $i^* \in X$ such that $f(i^*) \leq f(i)$ for all $i \in X$. Furthermore, $f^* = f(i^*)$ denotes the optimal cost, and $X^* = \{i \in X \mid f(i) = f^*\}$ denotes the set of optimal solutions.*

In this definition, 'solution' is being used in a broad sense to mean any well-formed answer to the problem that maps to a cost through the function $f$. The objective is not to find *a* solution, but to find a *minimum cost* one. In this case, therefore, it is meaningful to talk about an approximate solution to the problem, i.e., one that is *close* to optimal.

Once the problem is defined, we commonly express the task of optimization – the fact that we wish to minimize our cost function over a particular set of potential solutions, as follows:

$$\text{minimize } f(\mathbf{x}), \text{ subject to } \mathbf{x} \in X. \tag{1}$$

Here, $f$ is the objective function, and it maps any solution $\mathbf{x}$ that is a member of the set of feasible solutions $X$ to the set of real numbers, $\Re$. We are asked to find an $x$, such that $f$ is minimized.

## 2.2 Hard Optimization Problems

Some optimization problems are easy to solve, and fundamentally so. An easy problem is one where there exists a method that always works — that solves every valid instance — finding an optimal solution in a reasonable amount of time, or number of steps. An example of an easy problem is that of finding a shortest path between two nodes in a network, where the nodes are separated by links with certain lengths. This problem can be solved using Dijkstra's famous algorithm (dynamic programming), an ingenious method that greedily constructs the optimal solution, by considering partial paths through the network and keeping track of the competing alternatives.

Unfortunately, a great many problems that we encounter — and almost all of those in science and industry — are hard. That is to say, there is no known method for solving instances of them exactly and reliably. More than that, these problems are fundamentally hard in that they can be shown to belong to a set of problems, all of which are essentially equivalent in their difficulty. The equivalence means that finding quick and reliable methods for solving one of the problems would result in quick and reliable methods for all of them.

However, no such method has yet been found, and many think that such a method does not exist. Computer scientists call optimization problems that are fundamentally hard in this way, NP-hard [17].

Consider the following list of problems:

- Find a competent, or good, strategy for playing the game, Othello
- Allocate resources for flood protection of the UK
- Define a taxonomy of prokaryotic genes, by their functional activities
- Design a suspension bridge
- Schedule the jobs in a factory, as orders and raw materials arrive periodically.

When defined more precisely, each of these is an example of a fundamentally hard problem. There is no way to go directly to an optimal solution, or to organize a search in a super-efficient fashion that gives reliable results for all instances.

Instead, for problems like these, we can only hope to find good (or approximately optimal) solutions, by a process of searching through alternatives. Further, being fundamentally *hard*, this process is likely to take significant time, even using modern hardware, and even just to find an acceptable, rather than good, solution.

Fortunately, however, we rarely need to resort to a blind, random search through the set $X$. Each of these problems, and realistic problems in general, have some inherent *structure* that can be exploited as we try to devise a strategy for seeking good solutions. This structure is usually apparent in the way that similar solutions are related in terms of the cost function(s). For example, if two suspension bridge designs are very similar, but differing slightly in the distance between the points of suspension, then the performance characteristics of these two bridges are likely to be very similar too. Such structure is partly captured by the *search landscape*; once we have settled on the details of $X$ (having decided how we will encode potential solutions to the problem), the details of $f$, and the specifics of how we will generate new solutions from others previously sampled, this *landscape* comes to life as a high-dimensional mathematical structure. A fundamental aspect of all modern search strategies is to sample points in this landscape (i.e., points in $X$), and attempt to discover (or simply assume) certain properties of the landscape in an attempt to navigate a path through it towards good solutions. One class of such strategies,

called evolutionary computation, is recognized as being particularly successful (in comparison to other methods, at least) at handling the landscapes found in most real-world problems.

## 2.3 Evolutionary Computation

> Algorithms are conceived in analytic purity in the high citadels of academic research, heuristics are midwifed by expediency in the dark corners of the practitioner's lair ... and are accorded lower status. *Fred Glover*

### Origins

Problem solving by simulated evolution has been invented independently several times. Its pre-history can be traced back at least to Butler, who pronounced that that machines might 'become as complicated as us' by evolutionary processes, long before general purpose computers had even been conceived (see Dyson [9], chap. 2). Actual computerized simulations began in the 1950s, one of the early notable examples of problem solving being work by Baricelli [1] to 'evolve' fragments of code that cooperated to play a game called Tac-Tix. Fraser [16], Bremmerman [3], Rechenberg [30], Schwefel [33] and Fogel [14] all experimented with models of evolution in independent work conducted in the 50s and 60s, and were joined by many others, notably Holland [19] in the 70s. These researchers had very different ends in mind, and emphasised different elements of the neo-Darwinian principles of evolution in the models that they investigated (for detailed accounts of this early, pioneering work see [13] and [9]). For some time, the differences were cemented, and the distinct methods known as evolution strategies, genetic algorithms and evolutionary programming developed in isolation. Today, and since the 1990s, evolutionary computation is inclusive of all these areas, and innovations cross the boundaries, using common concepts and abstractions from nature.[4]

### The Basic Evolutionary Algorithm

Despite the high-falutin ideals of evolutionary computation to model and abstract from the complexity and richness of nature's wandering adaptive walks, the basic evolutionary algorithm for optimization has much more in common with a process of selective breeding (as used by Mendel) than it does with adaptation. The optimization problem provides a static goal to which the algorithm is directed, and a *population* of solutions are improved towards this by rounds of evaluation, biased selection, reproduction, variation and replacement. One round or cycle is called a *generation*, and the pseudocode in Fig. 1 captures the central process in practically *every* evolutionary algorithm:

Typically, in evolutionary algorithms, there is made a distinction between the genotype and the phenotype of a candidate solution, with the genotype being the medium of reproduction and variation (step 3.2 in Fig. 1), but with selection being based on the evaluation of the phenotype (steps 2 and 4). There are several alternative schemes for selection, but the basis for them is usually a relative ranking

[4] We might say there is panmictic (all-mixing) evolution.

1. Generate a population of candidate solutions to the problem
2. Evaluate the fitness of each candidate in the population
3. Produce some new solutions from this population:

**while** not done **do**

3.1 Select (preferring the fitter ones) some to be 'parents'

3.2 Produce 'children' (new candidate solutions) from the parents

**end while**

4. Evaluate the fitness of each of the children.
5. Update the population, by incorporating some of the new children, and removing some of the incumbents to make way for them
6. Until there is a reason to stop, return to step 3.

**Fig. 1.** Pseudocode for an evolutionary algorithm

of the solutions; the *fitness* of a solution then refers to the expected reproductive opportunity afforded it by selection.

Reproduction occurs by a combination of replication and mutation events, or recombination and mutation — both lead to variation in the children produced. In recombination, genetic material from two or more solutions are crossed over to yield one or more recombinant offspring. Mutation refers to a small change that is made to the genotype of an offspring, following a recombination or replication event. Evolutionary algorithms are always stochastic, and in most cases, selection, recombination, and mutation are all based on dice-rolls. Moreover, the starting or *initial population* (step 1 of Fig. 1) is most often created by a stochastic process. The offspring population, once created, replaces the population of the previous generation, becoming the new *current* (parent) population.

Typically, the population in an evolutionary algorithm is of a fixed size from one generation to the next. Replacement of the old by the new can again be based on competition (i.e., selection), but it can also be entirely random. One particularly popular replacement scheme is for the best few individuals (the elites) of the parent population to be protected from replacement, so that they survive across generations. This is called elitism, and it is implemented in most evolutionary algorithms because it ensures non-retardation of the best solution(s).

Generally, there is immense variety in the way that individual evolutionary algorithms will carry out each of these steps. To a large degree, the reasons for and nature of this variety relate to the precise problem being addressed — this will dictate, for example, the *encoding* used (how candidate solutions are represented as data structures), which in turn affects the way that recombination and mutation may be done. The more problem-independent aspects of EAs are selection for breeding (step 3.1) and so-called 'environmental selection' (step 5). Many techniques have been tried and tested, but the general lessons that are clear from practice are that a 'low pressure' strategy tends to do well on the more interesting and difficult problems. That is, though we may be tempted to strongly prefer to breed exclusively from the most fit solutions in step 3.1, and be sure to be rid of the poorest solutions

in step 5, it turns out that we get more capable and reliable algorithms when we ensure that these steps are *gently* influenced by fitness.

## Why EC works, and the benefits of EC

Evolutionary computation is very successful, but why? There is continued debate about what underpins its general degree of success, but we first need to clarify what we mean by 'success' in order to address this question properly. What appears to be the case, at least for some important problems, is as follows:

*Performance* For some problems, EC is capable of much better solutions (and achieved in better, or reasonable, time) than all or most other known methods.

*Sufficiency* For some problems, EC is capable of solutions competitive with solutions achieved by all or most other known methods.

*Applicability* EC is applicable to almost *any* optimization problem.

*Accessibility* For some problems, EC has been applied, and works fine, but no comparisons with other methods have been done.

*Opportunity* For some problems, EC is the *only* approach that can be used with any chance at all of success — in other words, with EC we can solve problems that we couldn't solve before.

The *Performance* statement is what most people would assume is meant by "EC is successful". Indeed it is true, but it must be stressed that this situation exists in relatively few cases, usually those in which an EC practitioner has worked hard in configuring the key components of the method – the encoding, the operators, and perhaps other features (such as using a heuristic to provide seed solutions in the initial population). The reason for EC's success in such cases is tied up in the fact that EC provides a framework within which a new approach can be engineered. At heart, it seems plausible that the use of the central evolutionary concepts (Darwinian selection from a population, coupled with a means of variation) is a key element in the success here — i.e., it is a fundamentally powerful all-purpose landscape search strategy. However, it is worth noting that much work (usually) needed to be done to craft the landscape, turning it into a problem more amenable to this strategy.

The *Sufficiency* point is true for a great many problems, and it doesn't seem to characterize EC in a particularly exciting light, yet it speaks to EC's 'dependability'. The key point here is that the 'other' methods tend to have a much larger variance in their success than EC. Given, for example, a set of 100 different real-world problems to solve, an EC approach, crafted with no undue effort in each case, will probably do at least 'OK' on each of them. In contrast, an alternative method such as Simplex, that does well in a few cases, may be inapplicable for all other cases; a graph-based search technique that does well on a certain problem may perform terribly in other cases, and so on. Sometimes, use of EC may be over the top, like using an electron microscope to read the small-print in an insurance policy, and a rival technique will do just as well in far less time. However, that rival may have abysmal performance elsewhere in this set of problems. And so it goes on. As we noted with regard to *Performance*, it seems safe to attribute the success of EC to the notion that the Darwinian principles of evolution comprise a good all-purpose strategy for navigating the kinds of landscapes that spring up once we start to solve a real-world problem. The style of success inherent in *Sufficiency*, adds some weight to

this. We can understand the high variance in the performance of other methods in this context, by suggesting that such an alternative method will be ideally tuned to aspects of the landscape structure of some (maybe very few) problems, while being a manifestly hopeless strategy for other problems. The well-known gradient-descent approach is an obvious example. Essentially, with gradient-descent search, your problem will be solved quickly and optimally if the landscape's structure is that of a single, smooth multidimensional bowl (the optimum being the lowest point in the centre of the bowl); but on almost any other landscape this strategy will miss the optimum, by perhaps a great distance.

Bearing much relation to the last point, the *Applicability* of EC is well-known, and this, in its own right, is a type of success that EC enjoys in abundance. Almost by definition, if we have an optimization problem to solve, then we already have to hand some notion of candidate structures for $X$ and $f$. We need very little more than that before we are then able to at least make a first attempt at using EC to solve that problem. By contrast, other optimization techniques may require additional elements that are either unforthcoming, or painful to arrange — such as necessary features of the differentiability of $f$, or a sensible way to assess the quality of *partial* solutions, or a requirement that candidate solutions be real-valued vectors of a fixed length. An additional feature of EC's general applicability, important to some, is the great ease with which EC can exploit parallel computing resources.

Riding on its ready applicability, combined with its essential simplicity (requiring no particular mathematical or programming prowess, for example), EC is successful partly through its *Accessibility*. This in itself has led to many applications in which EC has been used, pronounced 'good', 'fine' (or whatever), but not actually evaluated in comparison with any alternative approaches. That is, some practitioners, given some problem that they needed to solve, have chosen EC (for one or more of the reasons already discussed), used it, and left it at that. Such cases are valid examples of 'EC successes', and some are in commercial use, but that is not to say that some alternative method wouldn't be (perhaps much) faster, and/or produce higher-quality solutions.

So, many of the applications of EC that we see in the literature, or even in the popular press, may only provide evidence that EC is a highly accessible algorithm, rather than contribute to the evidence that it is the best choice for the problem at han. Nevertheless there is ample evidence that for many important problems it is indeed an appropriate choice; while, in some cases it is arguably the *only* choice. The fact that EC imposes no constraint at all on the nature of the structures in $X$, the set of candidate solutions, leads to some notable achievements for EC when researchers exploit the *Opportunity* this provides. Essentially, there is no candidate in the list of (non-EC) potential methods for optimization that is able to be applied to the problem of finding ideal strategies for a fighter pilot to use during a dogfight. However, EC has been used for this, with notable success [35]. Similarly, though one can think of antenna design as a problem in which standard parameter vectors are manipulated to achieve variants on standard designs, EC provides the opportunity to think of optimizing antennae in a much wider sense; thus, Lohn [23] used EC to optimize a set of *algorithms* for constructing antennae (such an approach, Genetic Programming [21], is a large subfield of EC), enabling a search through a space of possible antenna designs in which existing styles was just a tiny, imperceptible corner. In this and many other cases EC becomes a way (perhaps the only successful, and automated, way) to discover innovative solutions, rather than simply optimize

around standard, prior designs. Many other such examples, as well as examples of more conventional successes, may be found on a visit to the 'HUMIES' awards website at http://www.genetic-programming.org/hc2007/cfe2007.html

We haven't yet quite answered the question of 'why' EC works. But there is no great mystery there. The well-known 'No Free Lunch' theorem [39] tells us that, given an entirely random collection of landscapes (so, think in terms of *all conceivable landscapes*) no single approach is capable of the type of success that we have claimed here for EC. The flip-side of this is that, given some *non-random* collection of problems — a collection in which there is a bias towards certain elements of general structure in the problem landscapes, say — a method may well exist which is generally better than others. It is highly plausible to suggest that the collection of *real-world* problems is highly biased in such a sense. In particular, once we have gone through the process of formalising a problem sensibly, and defined $X$, $f$ and the operators we will use to move within $X$, landscapes that we construct are invariably correlated, in the sense that nearby elements of $X$ tend to have similar cost. Thus real-world problems are highly biased towards correlated landscapes. Meanwhile, the essential Darwinian strategy used by EC, which is to follow 'clues' in a landscape under the assumption of such correlation (apples not falling far from the tree), yet not to overcommit too soon to any particular path or region in a landscape (everyone has some chance to reproduce, rather than only the very fittest), seems well suited to most of the landscapes in this class, while rarely being a particularly *poor* approach.

The basic theory behind the EC search strategy is well known, and exemplified in Price's theorem [27], which basically expresses part of the above in formal terms. In the EC field, specialisations of Price's theorem have been derived [19], [29], [26], which express nuances of the central idea, tied to specific kinds of solution structure $X$. Given these highly general statements, we can be satisfied that the prowess of EC is not a magical ability, but explainable. Meanwhile, similarly general statements for several classes of EC algorithm enable us to be satisfied that an EA will generally make progress in reasonable time [12, 32, 31, 37]. Beyond this, which we need not (and choose not) go into here, the EC literature is replete with incremental steps in our understanding of the many aspects involved in how to best configure an EA for a particular problem class. There remains very much to discover about that very point, but one key theme, which we certainly *will* develop further here, is one which also further evidences EC's traits of *Applicability* and *Opportunity*. Sometimes, with other optimization methods, a problem with two or more objectives can only be addressed if it is first simplified to a single-objective (and hence, a different) problem. But, as we will see, this hurdle is not present with EC.

## 3 Multiobjective Optimization: Why Many Are Better Than One

We have seen in the last section that amongst the benefits of EC is its general applicability in optimization. Yet, an optimization problem defined by a single objective function is itself a restricted class of problems. More generally, an optimization problem may have multiple objectives.

A multiobjective optimization problem (MOP) is typically formalized like this:

$$\begin{array}{ll} \text{minimize} & \{f_1(x), f_2(x), f_3(x), \ldots, f_k(x)\} \\ \text{subject to} & x \in X \end{array} \tag{2}$$

expressing the fact that we want, ideally, a single solution $x$ that minimizes *each* of $k$ distinct cost functions (also called objective functions). These functions may well be conflicting to various degrees, in the sense that a solution $a$ for which $f_1(a)$ is particularly good, may be such that $f_2(a)$ is particularly bad.

At this point, some terminology is necessary. A solution structure $x \in X$ is often called a decision vector (and $X$ is called the decision space), since the values in $x$ tend to encapsulate the design decisions that we need to make. Meanwhile, the vector $(f_1(x), f_2(x), ..., f_k(x))$ is referred to as the vector of objectives, which inhabits the so-called objective space, typically but not necessarily $\Re^k$.

The quality of a candidate solution $x$ is now no longer measured as a scalar, but as a vector. This makes necessary a new way to assess whether or not some solution $x$ is better than some other solution $y$. Previously we might say either "$x$ is better than (worse than) $y$" or "$x$ and $y$ are equally fit". Now, we can still say that they are equally fit, to describe cases in which the objective vectors for $x$ and $y$ are identical, but there are two distinct ways in which $x$ and $y$'s performance on the task may be different. First, we might have "$x$ is better than $y$", as before, to cover cases in which $x$'s objective vector is better than $y$'s in at least one objective, and no worse in all the others. This is called *dominance* and we say that $x$ dominates $y$. Secondly, we may have a case in which $x$ is better than $y$ on some objectives, but $y$ is better than $x$ on other objectives. In this situation we say that $x$ and $y$ are *incomparable*, or we say that they are *nondominated*.

Given a set of multiobjective solutions (such as the current population of solutions during an EMO algorithm run), some of this set will be dominated by others in this set. Those that are not dominated by any others in that set (which may be a single solution, or the whole set) form what we call the *Pareto set*. In objective space, the set of objective vectors corresponding to the Pareto set is called the *Pareto front*.

Commonly, the true, optimal solution to a real multiobjective problem is such a set, containing more than one, and perhaps hundreds or thousands, of nondominated points. Put another way, no single solution in $X$ dominates (or is equal to) all other solutions in $X$; instead, the minimization task is satisfied by a set of distinct, nondominated solutions. Strictly speaking, this set is the Pareto set (and the corresponding objective vectors are the Pareto front), while all other sets of nondominated solutions that we may form from elements of $x$ (such as the nondominated points of the current population during an EMO run) are, at best, 'approximations' to the Pareto set. Commonly, however, papers in the field refer to "Pareto front of the current population", and the precise meaning is usually clear from the context.

Any single point on the Pareto front is called *Pareto optimal*; it is (usually) not optimal in the single-objective sense, since it (usually) does not minimize each of the objectives; however, it represents a compromise, such that if any solution exists that improves upon it on one objective, then that solution will be worse on at least one other objective. Clearly, the Pareto set for any problem contains, for each objective, a point that 'truly' minimizes that objective. That is, if we are trying to find a bridge design that has minimal mass, minimal cost, and whose construction would involve a minimal carbon footprint, then we can expect three

of the solutions in the Pareto set to be, respectively, the best possible solutions in these respects among those members of $X$ that are feasible bridge designs. It is almost always too much to expect, of course, that any single solution will do particularly well on any pair of objectives, or all three at once, especially where there is such obvious conflict. For example, cheaper designs will invariably use less optimal materials in terms of strength/mass ratio, and will typically exploit mass-produced, environmentally questionable sources. Nevertheless, the vector of scores for these three points on each objective, representing the best attainable for each objective, is itself a useful reference point in multiobjective optimization, known as the *ideal point*. Some multiobjective optimization methods use an estimation of this point in order to set target directions for the search. Similarly, the so-called *nadir point* represents the vector of worst values for each objective, for points in the Pareto front (note that, for a problem whose Pareto front shrinks to a single point, the ideal and nadir points are the same).

These concepts are illustrated very simply in Fig. 2, where we see contrived examples of Pareto fronts for two problems. The white circles are supposed to represent the Pareto optimal solutions, plotted in objective space, for a two-objective minimization problem. The circles correspond to actual points (designs, decision vectors, etc.) in the Pareto set. The white square locates the ideal point — a solution that we cannot actually achieve in this problem, but showing the best attainable result for each objective individually. Notice that this particular Pareto front 'bulges' *towards* its ideal point — this is called a *convex* front. More generally, convexity is present in a Pareto front if we can generally draw straight lines between two different solutions, and find that there are solutions on the Pareto front that dominate the points on the line. Alternatively, fronts in some problems may display much concavity — this is the case with the Pareto front represented by black circles in Fig. 2 (these are also used in the figure to illustrate the concept of a nadir point). A particularly interesting aspect of problems with concavities in the Pareto front is that the solutions in the concavity are *not* the optima of any simple weighted sum of the objectives. That is, these may be points that the decision maker (see later in this section) will choose, since they may form an ideal trade-off given various considerations. However, they will invariably be missed in a search based on a single objective weighted sum, since, on such a unidimensional view of fitness, these so-called *unsupported* solutions are bested by other points on the front.

Now, from the viewpoint of the 'owner' of the optimization problem we are trying to solve, we seem to have a difficulty. In the more common approach to optimization, we will typically combine our different objectives into one (for example, adding up a bridge design's scores for cost, mass and carbon footprint) and concentrate on minimizing their sum. This eventually yields a single result – which is the bridge design that achieved the best combined score. Alternatively we may find several solutions that achieve the same best score, but, when using single objective methods, these will invariably turn out to be quite similar, and effectively the same design. Hence, the problem has been defined, the optimization has been done, and we can provide the solution, and move on to the next job. However, if we treat this as a multiobjective problem, and perform a multiobjective search, our tactics for the end game are not immediately clear. The outcome of our search is now a *set* of solutions, and these will typically contain quite a variety of different designs. What do we deliver as the single best design?

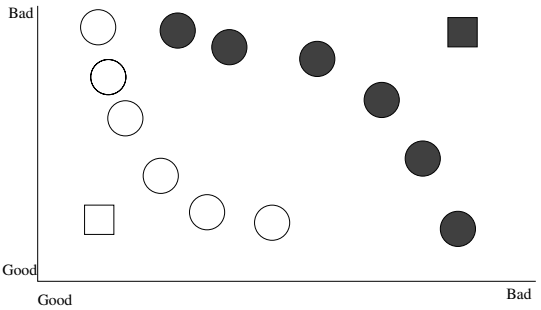


**Fig. 2.** Examples of possible Pareto fronts, convex (light) and concave (dark), showing the ideal point for the convex front (white square) and the nadir point for the concave front (dark square)

Before we answer that, certain notes will be instructive. First, whatever was the optimal point in the single-objective (added up) formulation of the problem is sure to be on the Pareto front of the multiobjective version of the problem. That is, an adequately designed multiobjective search will deliver the 'best' single-objective solution as one of the contents of the returned set of solutions. This is trivial to see, by noting that the solution that truly optimizes a single objective 'added up' formulation must be a nondominated solution. Second, notice that the usual practice, when combining many objectives into one, is to attempt to weight them suitably. So, if cost is more important to us than anything else, we will give this a much heftier multiplier than mass or carbon footprint, in the hope that this will guide our (single-objective) search towards cheaper solutions, but which still show some consideration for the other objectives. However, for the same reasons as before, the optimal solution to this 'new' single objective problem will also be on the Pareto front of our (unaltered) multiobjective search. Indeed, the Pareto front contains the optima for *every* possible weighted-sum based single objective search for this problem.

Thus, one way to view a multiobjective search is as a way to free the problem-solver from the need to specify weights for each objective. It is notoriously difficulty to decide on the correct relative weights in the first place, but you can be sure that the returned Pareto set will contain (at least a good approximation to) the solution that optimizes the 'correct' weighted-sum single-objective formulation (as well as all others), without ever needing to specify the weights.

Suppose, then, that you are an engineer who normally casts your problem as a single-objective weighted sum, but has been convinced, by one of the authors of this chapter, to do a multiobjective search instead. Suppose, too, that beforehand you specify a set of weights, $W$, for each objective, just as you would normally do. Now, proceed to do a multiobjective search, but without actually making use of $W$. At the end, you have a set of solutions — a set of bridge designs, or factory production schedules, or whatever. Faced with this choice of possible solutions, and looking for an easy, swift, automatic way to make the choice, you can simply take the one that minimizes the single objective sum specified by $W$. So, why do it multiobjective at all? Well, the difference is potentially twofold. First, via the multiobjective search you may have found a better result, in terms of the single-objective score found by

$W$, than you would have using a single-objective search method. This is a commonly observed phenomenon. Second, you are presented with a diverse set of solutions that provides information about the trade-offs available to you. Even though the weight set $W$ may represent for you a robust statement of what you require in a design (though usually it doesn't), some solutions on the Pareto front that don't optimize this particular weighted sum may nevertheless grab your attention. You may well discover, for example, that an unexpectedly good saving in mass may be possible for just a slight increase in cost. True, what we are suggesting here is that a decision needs to be made, and in that sense the multiobjective search seems not to have automatically solved your problem for you. But, on anything more than cursory inspection, it becomes clear that the multiobjective search has provided everything that the single-objective search would have provided for you, *plus more*, so this is not an extra decision to be begrudged, it is an extra opportunity, to grasp or ignore as you see fit.

Multiobjective search is therefore viewed as a way of providing the opportunity for a *decision maker* to make informed decisions about the solution based on information about the solutions that inhabit the Pareto front. In contrast, a single-objective formulation and search, when applied to an inherently multiobjective problem, provides a solution that may look appealing in the absence of alternatives, but is otherwise potentially far from what the decision maker may choose given a better supply of possibilities.

When we therefore decide to face a multiobjective problem on its own terms, and apply a search method that supplies a variety of different but equally 'optimal' solutions for a decision maker to consider, there are various ways we can respond to this opportunity. As noted above, if we have a preferred weight vector at hand, we can use that to pick the 'best' one. If instead we are skeptical about this, or any, weight vector, other approaches are available to us, from the long-established field of *multicriterion decision making*.

### 3.1 A Note on Multicriterion Decision Making

> The man who, though exceedingly hungry and thirsty, [is] both equally, being equidistant from food and drink, is therefore bound to stay where he is.
>
> Aristotle, On the Heavens (Book II)

Given that we have used an approach that generates an approximation to the Pareto front, the decision maker is provided with this as both a collection of different solutions to the problem, and a source of information about the conflicts between the objectives, and other aspects of the space of possible solutions. If the decision maker is an expert in the problem domain (which should normally be the case!), she may go into a dark room, and emerge some time later having made her choice, based on perhaps deep consideration of the information at hand as well as other, unformalized (maybe unformalizable) aspects of the probable performance characteristics of the various potential solutions.

But, such decision makers are expensive, and it is therefore desirable to have more formal, automated ways to help decision makers minimize their effort. These

are generally ways to use additional information about the problem or problem domain, which may have been difficult to include in the original search that led to the Pareto set. There are many standard such methods, and the reader may refer to any textbook on multicriterion decision making for further information on the many existing techniques for selecting a final preferred solution (e.g see [10, 25, 34, 38]).

To provide a flavour of the type of method in use, however, we mention first the idea of 'preference articulation'. When an expert is at hand who is able to provide authoritative views on how to balance conflicting measures and goals, this can be exploited by using preference articulation techniques [8, 2, 20], whereby a series of concrete questions about preferences are asked to the decision maker. The answers then determine if it is possible to build one or other type of consistent model of the decision maker's internal utility function; if so, then an automated procedure can potentially be developed for solution evaluation/selection. Note that far more complicated types of model exist for this than a simple weighted sum over the objectives.

## 3.2 Visualization Methods

When tackling an optimization problem, visualization may be used to present various features revealed about the problem, or to present information about the search method being used. Amongst other things, the purposes of visualization include estimating the optimal solution value, monitoring the progress or convergence of an optimization run, assessing the relative performance of different optimizers (including stochastic optimizers whose results form a distribution), and surveying features of the search landscape.

In multiobjective optimization, the above purposes of visualization remain important, but the set-valued nature of the results and the conflicts that exist between objectives mean that additional or dual aspects come into play. These will often include gaining an appreciation of the location and range of the Pareto set/front, assessing conflicts and trade-offs between objectives, and selecting preferred solutions.

In the following, we briefly present some of the visualization techniques used by contributors in this book. For more information on visualization techniques that go beyond those used in this book, the reader is referred to [25] (pp. 239–249), [28] and [24].

A basic task in MOO visualization is to illustrate the Pareto front, or the approximation of it found by an optimizer. A raw plot of the Pareto front approximation for bi-objective (or sometimes three-objective) problems (see Fig. 3) is thus very common. In this book, several of the chapters use this visualization technique to present results or concepts. When used carefully, it is an intuitive and straightforward method which can yield much information in a small amount of space. It is worth remembering, however, that the eye's tendency to interpolate between points is usually not to be trusted when considering points shown in such a plot. The boundary of the region dominated by a set of points is represented by its attainment surface, as shown in Fig. 3. This is the representation used in the chapter by Handl and Knowles in their visualization of the Pareto fronts obtained by multiobjective clustering.

Many problems of interest go beyond two or three objectives, of course. To gain an understanding of the range of the Pareto front and the conflicts between

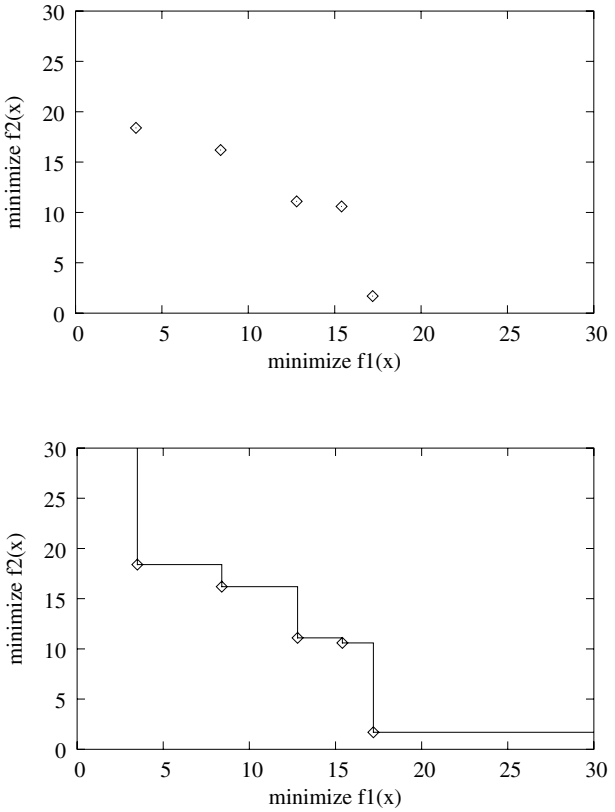


**Fig. 3.** (Top) A standard two-objective plot of a Pareto front approximation. (Bottom) The corresponding attainment surface represents the family of tightest goals that are known to be attainable as a result of the points found

objectives (and to help select preferred solutions), a parallel axis plot (also known as a value path) [18] is one method that has much to recommend it (see Fig. 4): it can handle relatively many objectives; all objectives are represented simultaneously; the value of each objective is shown by position along a standard numeric axis; and the conflicts between pairs of adjacent objectives is represented by the angles of lines. By adding interaction, allowing a user to change the order of presentation of the objectives and to set acceptable values for objectives, it is possible to learn much about the Pareto set. Parallel axes plots, or variations of them, can be seen in use in the chapters of Rodriguez and Fleming, Brockhoff et al, and Parmee et al in this book.

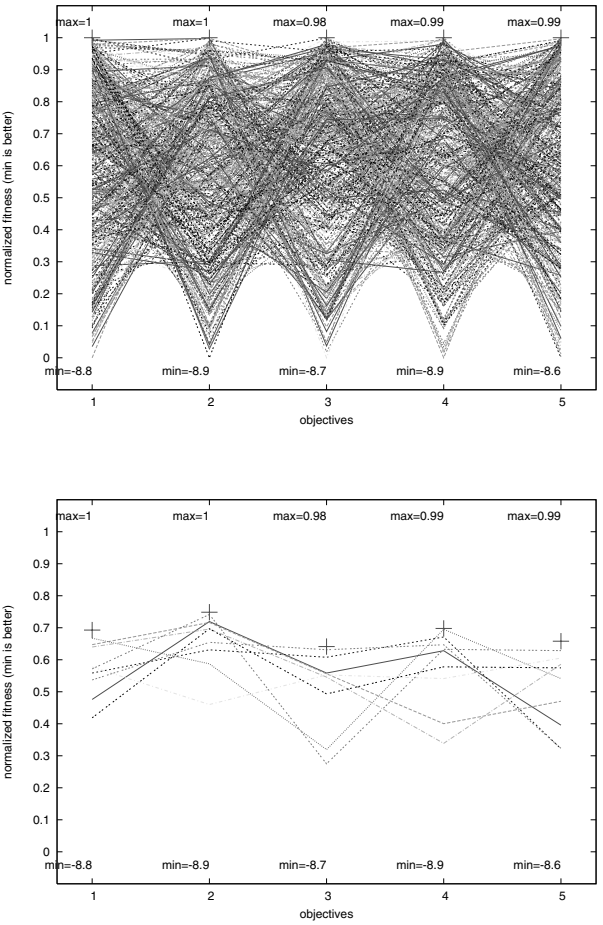


**Fig. 4.** (Top) A parallel axes plot showing a whole approximation set. (Bottom) A subset of the approximation set, arrived at by interactively setting acceptable levels for each objective (shown by the cross-hairs)

To assess the progress of an optimizer, a plot of best function value against iteration (or time) is the standard method used in single-objective optimization. In multiobjective optimization, this simple and very useful visualization device cannot be used. However, some authors in the book use an appropriate one-dimensional measure of progress in place of best function value, and plot this instead. Cutello et al do this in their chapter, showing how progress towards a known gold standard is made over time.

Similarly, it is often useful in single-objective evaluation to plot best function value achieved (after convergence) against some metaparameter, like a parameter of the search method used. Jin et al and Branke et al in this volume both plot one-dimensional performance metrics against meta-parameters to show the effects of the latter on search performance.

### 3.3 From EAs to EMOAs

Our primary interest in this book is the use of evolutionary multiobjective optimization algorithms (EMOAs) — that is, evolutionary algorithms that work directly with fitness vectors, rather than scalar fitness values. The benefits and merits of EAs, as discussed above, carry over directly to EMOAs, while the steps we need to take to operate with fitness vectors rather than scalars are relatively straightforward, as we shall briefly discuss.

Before launching into an introduction to EMOAs, however, we do not dismiss alternative optimization strategies. Though not our focus, non-EMOA multiobjective optimization predates EMOA, and continues to thrive. The field of operations research (OR) is the primary alternative to EMO in this respect.

OR boasts a rich and growing multiobjective optimization literature. Many of the problems considered in OR are multiobjective versions of convex problems, e.g., minimum spanning tree, where the single-objective version is polynomial-time solvable by convex optimization methods; to find Pareto optima of the multiobjective problem, one can scalarize the objectives, and apply the same convex methods (so finding one Pareto optimum is 'easy'). However, for most of these problems, it can be shown that the number of Pareto optima is exponential in the number of input variables, and hence finding the Pareto optimal set is intractable for large problems (see [11]). For these reasons, some of the work reported in the OR literature is based on mathematical programming techniques supplemented with different schemes for approximately sampling the Pareto front via scalarizing methods. Methods also based on traditional optimization techniques, but using decision-making before or interactively during optimization to reduce the number of Pareto optima to find, are also popular.

In parallel, the OR field has also developed methods for nonconvex multiobjective optimization problems. These rely mainly on approximate optimization algorithms such as simulated annealing and tabu search. To adapt them for use in multiobjective optimization, two main approaches are possible. One is to adjust the core functions of the algorithm, such as the acceptance function, so as to base decisions on dominance relations between solutions (or other factors, such as proximity of solutions in objective space). The other is to leave the basic algorithm in its original form, and rely on scalarizing techniques (as used in the mathematical programming approaches discussed above) to build up an approximation of the Pareto optimal front.

Meanwhile, in recent years, some OR researchers have begun to experiment with evolutionary algorithm approaches, encouraged by the rapid development of the field of evolutionary multiobjective optimization. This brings us to our main topic.

There are comprehensive and authoritative recent texts available providing reviews of and descriptions of the field of EMO (e.g., [5, 15, 7]). The central difference to the single objective case is the assignment of fitness — obviously, each individual

in the population now has its performance characterized by a fitness *vector*, rather than a single value. Perusal of Fig. 1 suggests that the only way this need affect the operation of an evolutionary algorithm is in the selection steps — i.e., steps 5 and 9, in which we are making decisions that require us to compare candidate solutions in terms of relative fitness. This is indeed the case, and EMO algorithms tend to be characterized by how these steps are performed.

The primary style of approach, often referred to (see several of the coming chapters) as Pareto ranking, is to give each point in the population a single score based on the degree to which it dominates, or is dominated by, other points in the population. In one of the most celebrated such approaches, *nondominated sorting* [36], the score assigned to a candidate solution reflects the 'depth' to which it dominates other candidates in the current population. Specifically, all of the nondominated solutions in the population are given the best rank (say, rank 1). These are then marked as 'ranked', and we proceed to find the nondominated front among the *unranked* remainder. These are ranked 2, and the process continues until all candidates have been ranked. An individual's rank is then converted into a selective probability in any one of numerous ways. Often, for example, practitioners will borrow the simple *Tournament Selection* method common in single objective EAs, in which some (usually) small number from the population are randomly picked, and the best of these (highest ranked, breaking ties randomly) becomes selected as a parent. As for the other selection step (step 5 in Fig. 1), the typical approach in EMOAs is to ensure that the Pareto front of the merged parent and child population is preserved, and any remaining space in the population is filled by relatively highly ranked members. It is common, however, for the Pareto front size to be larger than the population size — when this happens (and more generally too, e.g., in considering which of the non-front candidates to keep, when this is an issue) — selection decisions are often based on density in objective space. So, we might prefer to maintain points that are in relatively sparsely populated parts of the front, and don't mind losing solutions that are in crowded sections. These are certainly not the only approaches to selection, and meanwhile we have skipped over many issues that are the topic of hot research subareas in the field. However, the reader is again referred to any of several accessible papers and texts that present the main techniques, and others that describe ongoing research areas.

Meanwhile, the remainder of the book attempts to characterize and focus on a certain category of subareas, in which EMO is considered more widely, as a framework in which a variety of novel approaches to problem solving can be devised and implemented.

## 4 New Directions in Problem Solving with MOO Techniques

Rather than regarding problems as immutably divided into single-objective and multiobjective, and basing such distinctions purely on the properties required of a solution to the problem, EMO scientists, in practice, are finding reasons to blur these distinctions, co-opting the EMO mechanisms for handling individual objectives for related but distinct purposes that enhance the optimization process. This book is largely about the most prominent and successful of these new problem-solving

approaches. It is not a book of EC techniques or of isolated applications, but rather concentrates on the concepts guiding the use of EMO in *broad* problem classes. In particular, it shows how EMO

- can be used to understand and resolve ill-defined problems;
- helps in dynamic optimization environments, in problems with constraints, and in various learning problems where quality is not always directly measurable or free from biases;
- can eliminate a measurement bias or other confounding factor in the optimization of an objective;
- can combine information from multiple sources;
- can change the landscape of a problem, making it easier to search;
- supports proper progress and convergence in search problems where the objective is not explicit, but instead based on tests or competitions (e.g., in co-evolution);
- and, may be employed in the reverse-engineering of artificial and natural systems, where it can contribute to the quest for new principles of design.

Among these and other lessons, we also learn more about the decision-making step for choosing a 'final' solution that is associated with more traditional uses of multiobjective optimization, and explore how this need not be an entirely subjective matter. In some uses of EMO presented in this book — for example, when applying it to traditional 'single-objective' problems, like constrained optimization — the 'best' solution is not a function of decision-maker preferences, and its identification can be automated.

## 4.1 Chapter Summaries

### Part I — Exploiting Multiple Objectives: From Problems to Solutions

Part I of the book is a collection of chapters about problem formulation. It shows how *broad classes* of problem, usually formulated with a single objective to optimize, can be re-cast as multiobjective problems, with various beneficial and sometimes even dramatic effects. In some cases, the effect achieved is an improvement in the efficiency of searching the problem space. In other cases, there is a more profound effect, so that different and better solutions can be accessed asymptotically, i.e., the ultimate potential outcome of the search is improved. In yet other cases, the new problem formulation yields a greater understanding of a problem, with its competing goals and objectives, and this can help to re-evaluate the problem, possibly leading to a more conscious refinement of it.

Using several objectives to help solve what are traditionally considered 'single-objective' problems may raise the spectre of 'relaxing' the problem in some people's minds, resulting in a set of trade-offs, when only one solution is really wanted. However, as is shown throughout the chapters in Part I, this does not turn out to be a difficulty: the single-objective formulation (if it exists in well-defined form) can always be invoked *post hoc* to select the best solution; and where this is found inappropriate, it is because the multiobjective problem formulation has revealed a weakness, or a hidden assumption, in the original problem definition — one that should be externalized and dealt with appropriately. This issue of selecting the final solution is discussed at some level in most of these chapters, and what is found is

an interesting contrast to the usual view that multiobjective optimization always implies a phase of (human) decision making.

Ficici (Chap. 2) considers co-evolutionary algorithms (CEAs), a thriving area of research that has the potential to make valuable contributions to the breadth of the whole problem-solving domain. The defining characteristic of CEAs is that the fitness of an individual (a candidate solution) is defined, implicitly, by interactions with other individuals. From this, it follows that CEAs can be used to solve problems for which no known (explicit) objective function exists: problems that would be impossible to tackle using traditional optimization methods, such as finding optimal strategies in two- or multi-player games. Ficici's chapter shows how Pareto optimality can be used as an organizing principle in CEAs, with each individual being viewed as a potential objective for optimization. From this idea, several long-standing difficulties associated with CEAs can be better understood and largely circumvented. Moreover, the multiobjective framework allows the general co-evolutionary learning problem to be handled in such a way that monotonic improvement of solutions is ensured. This idea relates to elitism in MOEAs, and to the use of archives of non-dominated solutions, a theme which is touched on in several other chapters in the book, particularly de Jong and Bucci's. The issue of decision making is raised in the chapter, and its relationship to the concept of *refinement*, and the *equilibrium selection problem* in game theory is described, with some tantalizing prospects for future developments.

One of the earliest uses of multiobjective methods for solving single-objective problems was their application in constrained optimization, an area thoroughly reviewed in Mezura-Montes and Coello's work (Chap. 3). Constrained optimization problems represent a large and important chunk of real-world problems, especially in engineering, but they still pose a challenge to traditional evolutionary algorithm methods. In this chapter, the common EA approach of using penalty functions is compared, conceptually, with multiobjective formulations of the problem. Two benefits of the latter are posited: that weights do not have to be selected in order to balance the different importance or ranges of the constraints; and, that the number of improving paths to the optimum is much greater, which increases the possibility of approaching good solutions. Work that both supports and criticizes these assertions is considered, and an empirical study is used to compare some of the current state-of-the-art methods. In constrained optimization, it is shown that decision making never enters as a matter of DM preferences, even in the multiobjective formulation. In other words, there is an automatic way of selecting the objectively best solution from the Pareto front in all cases.

Another considerable area of research in problem solving concerns optimization in dynamically changing contexts, as explained in Chap. 4., Bui et al. investigate using a secondary helper objective for this class of problems, aimed at maintaining diversity in the evolving population, and thus a readiness for sudden or periodic changes in the optima. The use of a secondary objective, and application of standard MOEAs, is a simple approach to handling dynamism, and what is more it does not introduce any issues related to decision making. Comparisons made with existing EA mechanisms for dynamic problems, namely hypermutation and random immigration, show the MOEA approach already gives the most consistently good performance, while there remains much room for further development of the technique.

Cutello et al (Chap. 5) apply multiobjective EAs to the traditionally single-objective problem of predicting a protein's native structure, a pressing and massively

significant problem in the biological sciences. The rather specialized nature of this problem belies the fact that it may serve as an archetype for others in which the objective function is not really objective or final, but a proxy used to help find solutions. In structure prediction, it is an energy function that is minimized, and this is essentially a guess made up of several components of energy; the ultimate arbiter of quality, however, is not the objective function, but the distance to the observed, real structure (which is not available at the time of optimization in real instances of the problem, however). Taking a multiobjective approach to the problem (here by decomposing the energy function into its components) is a process of learning how to *align* the objective functions with the ultimate measure of solution quality. Here, the flexible nature of multiobjective search is being used as a way to improve the models on which the optimization is based.

Neumann and Wegener's interest (Chap. 6) is in the possibility that for well-defined problems a multiobjective formulation could be straightforwardly faster to solve for an evolutionary algorithm than a single-objective one. Taking two classic problems from combinatorial optimization, the single source shortest path problem and the minimum spanning tree problem, they demonstrate that this possibility is not a fiction. For the MST, the asymptotic expected optimization time is derived for simple multiobjective and single-objective EAs, indicating the superiority of the former. Experimental results on different instance types also show a performance advantage of multiobjective algorithms for some classes of minimum spanning tree. In all cases, the problem formulations used here directly yield unique solutions to the original problem and no extra step of decision making is involved.

Handl and Knowles (Chap. 7) express and develop a view of problem-solving-via-MOO that concords with some of the ideas expressed in the preceding five chapters. They believe that in practical problem-solving applications, MOO is used in a variety of subtly different ways, called *modes*. The modes capture the specific reason why the problem has been formulated with multiple objectives, and what job each of the objectives is doing. Handl and Knowles identify five different modes and provide examples of each from their own research. They also show how some modes require no decision making for solution selection, while others reveal useful trade-off information that would normally be hidden, but which must be accounted for to select a final operational solution. For the latter case, however, some automatic and semi-automatic methods of decision making have been successfully devised, notably those based on the shape of the Pareto front and the consideration of control distributions.

## Part II — Multiple Objectives in Machine Learning

Part II of the book concerns the application of MOO to different problems in machine learning. The chapters collected here, like those in part I, emphasise the reasoning behind the multiobjective formulations presented, and demonstrate that core difficulties in machine learning can be understood and alleviated by the multiobjective approach. Common themes in machine learning, and in these chapters, are the trade-off between accuracy and model complexity; conflicts between training, validation and testing errors; and the combining of rules or classifiers. More unusual issues that are also highlighted include competing errors in multi-class problems, program bloat in genetic programming, and the value of promoting models that humans can understand in system identification.

Fieldsend et al (Chap. 8) consider the supervised learning paradigm, in which the output class or value of a datum must be predicted from its inputs, following a period of training on a random i.i.d. sample of example data. It is well known that the supervised learning problem is about generalization performance, which is difficult to assess during training, and hence different terms are often added to the basic training error objective to achieve regularization or model selection. Fieldsend et al consider multiobjective approaches to this central issue, and show some graphical methods for identifying solutions that best balance accuracy vs model complexity. The chapter also identifies a number of supervised learning problems where competing error terms are inherent, and a balance must be struck between them. One such is the different costs of misclassifications in multi-class data, most notably in disease diagnosis. Some groundbreaking methods in this problem area are presented.

The first of two consecutive chapters on genetic programming is Bleuler et al's (Chap. 9). Genetic programming is a form of computer program induction, based on evolutionary algorithm principles (see [22]). Bleuler et al focus on the problem of 'bloat' in GP, whereby evolved programs have the tendency to grow larger and larger, containing more and more useless code. This problem with GP has been a bugbear for several years, and several methods for counteracting it have been proposed and studied. In recent years, several multiobjective approaches have been tried, with considerable success. In this chapter, the reason behind the success of the Pareto-based approach to reducing bloat is investigated, following a thorough review of this area.

Rodriguez-Vázquez and Fleming (Chap. 10) concentrate on the use of genetic programming in system identification, specifically for non-linear dynamical systems. They show how a multi-stage process, which involves going back and forth between steps of structure selection, parameter estimation and validation can be compressed into a one-step process through the use of a multiobjective formulation. Moreover, human understanding of generated models is identified as an important issue which can be further enhanced by including objectives that control the type and complexity of model components used.

Rule mining is a method of classification, often for large databases, based on two processes, (i) extracting useful rules and (ii) combining them. Ishibuchi et al (Chap. 11) investigate both processes, exploring what is meant by a Pareto optimal rule and a Pareto optimal rule set, and how these can be approximated. They uncover interesting relations between accuracy and complexity, which echo the 'switchback effect' shown in Fieldsend et al's earlier chapter. They also show that Pareto optimal rule sets are not necessarily comprised entirely of Pareto optimal rules, and this is more the case as the ruleset size is allowed to grow.

## Part III — Multiple Objectives in Design and Engineering

In design and engineering, it is quite widely understood and accepted that problems invariably have competing objectives, and that problem solving is about finding good balances, or spotting niches where a different type of solution might be attractive for the first time. Part III of the book is about this area, particularly open-ended design, where it is almost a given that problems are multiobjective, when viewed at some level. Instead of explaining why and how multiple objectives arise here, the chapters rather focus on how to support understanding, learning and invention in a multiobjective space, and also how the same principles that are used for design

might also help when analysing and seeking to understand existing natural systems, which have inevitably evolved under several and various selection pressures.

Deb and Srinivasan (Chap. 11) suggest a systematic procedure of using two or more conflicting objectives (usually minimization of size and maximization of performance) to unveil salient knowledge about properties which when present in a solution would make it an optimal solution corresponding to the underlying objectives. The argument works as follows. Since Pareto-optimal solutions are no ordinary solutions in the search space, but rather correspond to optimal solutions of certain trade-offs among objectives, a series of such solutions is expected to possess some common properties that can provide a practitioner with important knowledge about 'what makes a solution optimal?'. This process of 'innovization' — the creation of innovative knowledge through multiobjective optimization — is illustrated through a number of engineering design problems.

Parmee's focus (Chap. 12) is on methods to support the human designer as she goes about her business, particularly in the area of conceptual design. Advanced methods of visualization, interactive evolution and machine learning are described, all aimed at taking away the drudgery of evaluation, and freeing the designer to make more insightful and high-level choices and inferences, based on an understanding of the multiobjective nature of the problem space.

Moshaiov is interested in analogies that can be drawn between artificial and natural systems (Chap. 13). He explores how and why such analogies have been useful because of what they can tell us about the design process, and about natural (evolved) phenomena. From this historic background, he moves on to consider why, in cybernetics, artificial life and evolutionary biology, the concept of trade-off is known, but a multiobjective view is rarely taken. Moshaiov explains how such a view could be made acceptable to both biologists and engineers, and considers what the consequences of this broader outlook might be.

## Part IV — Scaling Up Multiobjective Optimization

Much evidence for the potential of multiobjective optimization to deliver new and powerful solutions to problems, from classic combinatorial optimization problems to open-ended design problems, is provided in the first three parts of the book. To turn this into a reality, there is, of course, a continuing need for the development of effective multiobjective optimization methods. Of great concern to the field in recent times has been the scalability of the algorithms and concepts we use — scalability to increasing numbers of objectives and to larger design spaces. Part IV of the book presents some of the latest developments in the design of scalable multiobjective evolutionary algorithms.

Jin et al (Chap. 15) consider how the relatively low-dimensional manifold in which Pareto optimal solutions reside can be modeled and projected back into the much higher dimensional parameter space. Such an approach promises to achieve great scalability in parameter space dimension, provided certain base assumptions are valid. Jin et al show excellent performance of their techniques for problems with up to 100 real-valued parameters.

The first of three chapters concerned with methods capable of handling problems with many objectives is provided by Hughes (Chap. 16). He provides a background to the issue and reviews the capabilities of several existing Pareto and non-Pareto

multiobjective EAs at handling problems with four or more objectives. He considers both the issues of convergence to the Pareto front, and of controlling the distribution of points along it.

De Jong and Bucci (Chap. 17) are concerned with a particular class of problems that results in optimization problems with many objectives, easily tens or hundreds of them. This class is where the objectives are defined in terms of 'tests', an approach that may be taken when other methods of evaluation are not possible. Although the space of tests is usually very large, de Jong and Bucci show that tests need not all be arranged on orthogonal axes, but they can be grouped together, depending on how they affect the ordering of candidate solutions. This enables a significant reduction in the number of effective objective dimensions considerably.

Brockhoff et al (Chap. 18) also seek to reduce the number of objectives presented, to a lower number, which is easier to handle effectively. The first method for doing this has similarities with de Jong and Bucci's method for tests: it depends on inspection of the orderings induced by the objectives on sampled sets of solutions, and whether or not these orderings can be preserved when some objectives are removed from consideration. The second method does not consider orderings per se, it is based on principal components analysis on the objectives to establish the redundant ones. Both techniques are demonstrated on test instances, and unifying concepts are discussed.

## 5 Concluding Remarks

There are many interesting topics vying for inclusion in a book on multiobjective problem solving. Our choice has been guided by the desire to present the emerging concepts and methods that are being used to tackle important and long-standing classes of problems; and as a result some things have necessarily been left out. We have not touched on the use of other natural analogies in multiobjective problem solving, such as methods based on social insect behaviour, or flocking — methods that can trump evolutionary algorithms in some domains. We have not considered multiobjective optimization for problems where the number of solution evaluations possible is severely restricted, an area which is gaining rapid prominence because of the take-up of EMO in engineering and science, where evaluations are often costly. In machine learning, we covered several topics, but left out the contribution MOO is beginning to make in searching for ensembles of neural networks [4], though ensembles of rules are considered in the chapter by Ishibuchi et al.

And you, the reader, may well consider that we have overlooked something else — and you are surely right. But what remains in the book is, we hope, the beginnings of a synthesis that shows the contributions MOO is making to the core activity of problem solving with computers. Rather than an esoteric technique at the fringes of evolutionary algorithm research, or a specialist's area in operations research, MOO is now being used to help people solve all sorts of problems by offering genuinely new approaches to them. It goes beyond a method that allows engineers to balance out different criteria — though that is an important aspect of it. It can transform a problem, reaching solutions that were not possible before, or allowing monotonic progression where cycling previously occurred. It can take us to solutions, even to classic problems, more quickly than before. And we have seen that the algorithms

for achieving this are being developed anew, with fresh concepts to take us on to a new scale of problem-solving ability.

## Acknowledgment

JK gratefully acknowledges the support of the Biotechnology and Biological Sciences Research Council (BBSRC), UK.

## References

[1] N. A. Barricelli. Numerical testing of evolution theories Part II: Preliminary tests of performance, symbiogenesis and terrestrial life. *Acta Biotheoretica*, 16(3–4):99–126, 1963.

[2] V. Belton and T. J. Stewart. *Multiple Criteria Decision Analysis: an Integrated Approach*. Springer-Verlag, Berlin, Germany, 2002.

[3] H. J. Bremermann. Optimization through evolution and recombination. In *Self-Organizing Systems*, pages 93–106. Spartan Books, Washington, DC, 1962.

[4] A. Chandra and X. Yao. Ensemble Learning Using Multi-Objective Evolutionary Algorithms. *Journal of Mathematical Modelling and Algorithms*, 5(4):417–445, 2006.

[5] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[6] D. Corne, K. Deb, P. Fleming, and J. Knowles. The Good of the Many Outweighs the Good of the One: Evolutionary Multi-Objective Optimization. *IEEE Connections Newsletter*, 1(1):9–13, 2003.

[7] K. Deb. Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design. In K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, chapter 8, pages 135–161. John Wiley & Sons, Ltd, Chichester, UK, 1999.

[8] J. S. Dyer, P. C. Fishburn, R. E. Steuer, J. Wallenius, and S. Zionts. Multiple criteria decision making, multiattribute utility theory: the next ten years. *Management Science*, 38(5):645–654, 1992.

[9] G. Dyson. *Darwin Among the Machines*. Penguin Books, 1999.

[10] M. Ehrgott. *Multicriteria Optimization*. Number 491 in Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, 2000.

[11] M. Ehrgott and X. Gandibleux. An Annotated Bibliography of Multi-objective Combinatorial Optimization. Technical Report 62/2000, Fachbereich Mathematik, Universität Kaiserslautern, Kaiserslautern, Germany, 2000.

[12] D. Fogel. Asymptotic convergence properties of genetic algorithms and evolutionary programming: Analysis and experiments. *Cybernetics and Systems*, 25(3):389–407, 1994.

[13] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.

[14] L. J. Fogel. Autonomous automata. *Industrial Research*, 4(1):14–19, 1962.

[15] C. M. Fonseca and P. J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[16] A. S. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 10:484–491, 1957.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[18] A. M. Geoffrion, J. S. Dyer, and A. Feinberg. An Interactive Approach for Multi-Criterion Optimization, with an Application to the Operation of an Academic Department. *Management Science*, 19(4):357–368, 1972.

[19] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[20] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Cambridge University Press, Cambridge, UK, 1993.

[21] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[22] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, 2002.

[23] J. Lohn, G. Hornby, and D. Linden. An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission. In *Genetic Programming Theory and Practice II*, pages 13–15. Springer, 2004.

[24] A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev. *Interactive Decision Maps: Approximation and Visualization of Pareto Frontier*. Kluwer Academic, 2004.

[25] K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[26] R. Poli and W. Langdon. Schema Theory for Genetic Programming with One-Point Crossover and Point Mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[27] G. R. Price. Selection and covariance. *Nature*, 227:520–521, 1970.

[28] A. Pryke, S. Mostaghim, and A. Nazemi. Heatmap visualization of population based multi objective algorithms. In *Evolutionary Multi-Criterion Optimization (EMO 2006)*, volume 4403 of *LNCS*, pages 361–375. Springer, 2006.

[29] N. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4):339–384, 1994.

[30] I. Rechenberg. Cybernetic solution path of an experimental problem, 1965. Library Translation 1122, Royal Aircraft Establishment, Farnborough, UK.

[31] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.

[32] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 50–54, 1996.

[33] H.-P. Schwefel. Kybernetische Evolution als Strategie der experimentellen Forschung in der Stromungstechnik. *Master's thesis, Hermann Föttinger Institute for Hydrodynamics, Technical University of Berlin*, 1965.

[34] Y. Siskos and A. Spyridakos. Intelligent multicriteria decision support: Overview and perspectives. *European Journal of Operational Research*, 113(2):236–246, 1999.

[35] R. E. Smith, B. A. Dike, B. Ravichandran, A. El-Fallah, and R. Mehra. Discovering Novel Fighter Combat Maneuvers in Simulation: Simulating Test Pilot Creativity. In *Creative Evolutionary Systems*, pages 467–486. Morgan Kaufmann, 2001.

[36] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, India, 1993.

[37] D. Thierens and D. Goldberg. Convergence models of genetic algorithm selection schemes. In *Parallel Problem Solving from Nature — PPSN III*, volume 866 of *LNCS*, pages 119–129, 1994.

[38] P. Vincke. *Multicriteria Decision-aid*. Wiley, New York, 1992.

[39] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.