Optimization Cultures

21 April 2014

Manfred Gilli
Université de Genève and Swiss Finance Institute
manfred.gilli@unige.ch

Enrico Schumann
Aquila Capital Group
es@enricoschumann.net

Abstract

Computational optimization methods can broadly be classified into two groups: classical methods, which require and exploit specific functional forms of objective function and constraints, and heuristics. Those latter methods impose few, if any, restrictions on models, at the price of being more computationally demanding. But because of the growth of computing capacity over the last decades, those methods are now perfectly-practical tools for everyday use. Yet, instead of realizing the advantages of heuristics, users still cling to classical methods. We discuss the reasons for this non-acceptance of heuristics, and argue that the choice of numerical-optimization techniques is as much driven by the culture of the user – field of work and educational background – as by the quality of the method. In particular, we argue that many of the alleged shortcomings of heuristics could be overcome if researchers stopped treating optimization as a mathematical, exact discipline; instead, they should consider it a practical/computational tool.

Keywords: Numerical Optimization; Heuristics; Algorithms

1 Introduction

"I think there is a world market for about five computers." So said, allegedly, Thomas J. Watson, then chairman of IBM, in 1943. It would take another ten years, until 1953, before IBM delivered its first electronic computer, the 701.

Stories and quotations like that abound: people making predictions about computing technology that turn out to be spectacularly wrong.[1] To be fair, Watson probably never made that statement; most of such infamous forecasts are more likely made up or taken out of context.

But in any case, Watson's alleged statement certainly reflects the spirit of the time, and reading it today highlights the steep growth path that computing power has taken:[2] nowadays in everyone's pocket there are devices that perform millions of times faster than IBM's 701.

Of course, it is not only hardware that has improved, but software, too. If people still operated on terminals or had to use punch-cards for storing programs, computing would be much less powerful today.

This evolution, which took place in roughly a man's work-life, has led to a number of consequences.

Less division-of-labor. Because computers have become so fast and easy to use, people in many disciplines turn more and more into polymaths rather than specialists. Or, to put it differently, they become specialists in broader-defined fields. We see this happen in statistics and data analysis, where often a single person can handle data preparation and processing, analyze data, estimate models, run simulations and more. Or think of publishing. Modern computers and software have enabled people to not only write papers and books, but to actually produce them, i.e., create artwork or graphics, define the layout and so on.[3]

Portable software. Software and computing models that rely on specific hardware architectures lose their appeal. For instance, parallel computations that exploit specific communication channels in hardware have become less attractive. Instead of spending the next year with rewriting their programs for the latest supercomputer architecture, people can and should now use their time to think about their applications and write useful software. (Then, after a year, they can buy a better, faster machine.)

Interpreted languages. In the past, implementing algorithms often meant creating prototypes in a higher-level language and then rewriting such prototypes in low-level languages such as C. Today, prototypes written in languages such as Python or Lua are so fast that a re-implementation is rarely needed. As a consequence, implementation times have decreased, and we can much faster explore new ideas or adapt existing programs.

In this essay, we discuss how these powers can be put to good use in a specific area of computation: numerical optimization. We shall argue that the choice of methods and techniques in optimization is as much driven by culture as by merit, and that the mainstream culture has yet to recognize the new powers that are available.

Clearly, preference for one culture or another will be subjective (what language is better, English or French?), and we will not

[1] This applies as well to predictions about technology and to predictions in general. See, for instance, predictions on TV at http://www.elon.edu/e-web/predictions/150/1930.xhtml .

[2] Many examples have been made to provide intuition for the magnitude of this development. One of our favorites is from Dongarra et al. (1998): "Indeed, if cars had made equal progress [as microprocessors], you could buy a car for a few dollars, drive it across the country in a few minutes, and 'park' the car in your pocket!" If applied to computing power in general, that is an understatement: it would now take fractions of a cent to buy a car, less than a second to cross the country – and you might need magnifying glasses to even find your car.

[3] Curiously enough, that had actually been the state of affairs before, as Jan Tschichold (1971) observed: "In der frühzeit des buchdrucks waren drucker und verleger eine und dieselbe person. Der drucker wählte selber die werke aus, die er verlegen wollte; oft war er selber entwerfer und hersteller der typen, mit denen er druckte; er beaufsichtigte selber den satz und setzte vielleicht selber mit. Dann druckte er den satz, und das einzige, was er nicht selber lieferte, war das papier. Nachher benötigte er vielleicht noch den rubrikator, der die initialen einzuschreiben hatte, und einen buchbinder, falls er das werk gebunden auf den markt brachte." [Note that the capitalization is Tschichold's.]

even try to be impartial. Our thesis is this: The mainstream culture today treats optimization as a mathematical, exact discipline. We shall argue that optimization should be considered a practical/computational tool, not a theoretical/mathematical one. Instead of proofs, researchers should emphasize empirical and experimental results. To be clear: we do not suggest that optimization should do without mathematics. We merely suggest a shift in balance.

More specifically, researchers and users of optimization methods should rely less on techniques that impose strong assumptions on the optimization model. Instead, they should prefer a class of simpler, but extremely-powerful, methods: so-called heuristics. These methods give their users much more freedom in specifying a model, albeit at the price of being computationally intensive. Because of the advances in computing hard- and software over the past decades, they are now perfectly useable with everyday computers. In sum, such a change in how optimization is perceived, taught and applied – a shift in culture, if you want – would not cost us much, but we would gain a lot.

As a caveat: most of our experience comes from statistics, economics and finance, with excursions into fields such as computational psychology or environmental modeling. Yet we feel that our arguments will be relevant as well to other disciplines that use optimization.

2 On optimization

Numerical optimization starts with a model, typically stated as

$$\underset{x}{\text{minimize}}\ f(x)\ , \tag{1}$$

in which f is the function that we want to minimize, called the objective function, and x are the decision variables. If we wanted to maximize, we would minimize $-f$ instead. In most models, there are restrictions on how we may choose x.

Such optimization models do not exist in a "vacuum," but are set up for a specific purpose, and their results will, with luck, serve that purpose. Thus, we start with an actual problem, such as "how best to invest my pension savings?", and then translate this problem into a model; finally we proceed from the model to its numerical solution. In this essay, we will mainly be concerned with the second part, from a model to its solution. But we cannot emphasize enough how important the first step is, from problem to model. It may be fun to work on a challenging optimization model, but if the model is not useful than neither is its solution. In Gilli et al., 2011, we discuss this point for applications in finance.

General-purpose computer algorithms for solving optimization models work iteratively. They start with an initial solution, often specified by the user, and then try to improve that solution repeatedly. (Throughout this essay, we use the word solution in the sense of "candidate solution," or "result of running a computer program." In theory, the solution of a model is the set of parameters that minimizes f; thus, a solution is always optimal.)

Heuristics, the techniques that we will advocate in this essay, are simply a class of methods for solving optimization models. But we are getting ahead of ourselves, so back to optimization procedures.

The following pseudocode describes the workings of an optimization method.

1: generate initial solution x^c
2: while stopping condition not met do
3: create new solution $x^n = N(x^c)$
4: if $A(x^n, x^c)$ then $x^c = x^n$
5: end while
6: return x^c

A specific method defines how to generate new solutions (the function N), how to decide whether to accept such a new solution (the function A), and when to stop the search.

Clearly, this algorithm is very general. It would even accommodate a completely-random search: the function N would randomly draw a new solution x^n without regard for the current solution; if this new solution is better than the current solution, it replaces it.

More typically, optimization methods exploit the local behavior of the objective function, that is, they create a new solution that is a variation of the current solution. Thus, the new solution inherits parts of the old solution. As a result, the objective-function values of successive solutions should be positively autocorrelated.

The majority of techniques that is used today belong to a class that we shall call classical methods. Classical techniques concentrate their effort on searching for a candidate solution by constructing and solving convex local models. A new solution is accepted as long as it is better than the previous one (above a certain tolerance). Classical methods are the ideal techniques in a perfectly-convex world.

But in real-life applications, things turn out to be not convex. Multiple optima, nonlinearities, discontinuities due to integer variables or measurement errors, a variety of constraints and other peculiarities shape the objective function and search space in realistic models. See Figure 1 for an example.

Classical methods cannot accommodate such features; the only way to use them is to reshape the model such that it meets the requirements of the technique. That is, researchers need to adjust their models such that they fit the technique. Along the way, the model loses realism.

But with heuristics, there is no need to convexify one's models, since heuristics were specially developed to handle badly-behaved – i.e., realistic – models.

Instead of creating candidate solutions through sophisticated but fragile procedures, heuristics choose candidates randomly in a given neighborhood of the current solution.[4] These solutions are accepted if they are better, but also if they are worse, at least under specific circumstances, which we shall explain shortly. The number of iterations for heuristics is much larger than for classical methods, because we do not stop once we did

[4] The neighborhood of a solution x comprises all solutions that are, loosely speaking, close to x.

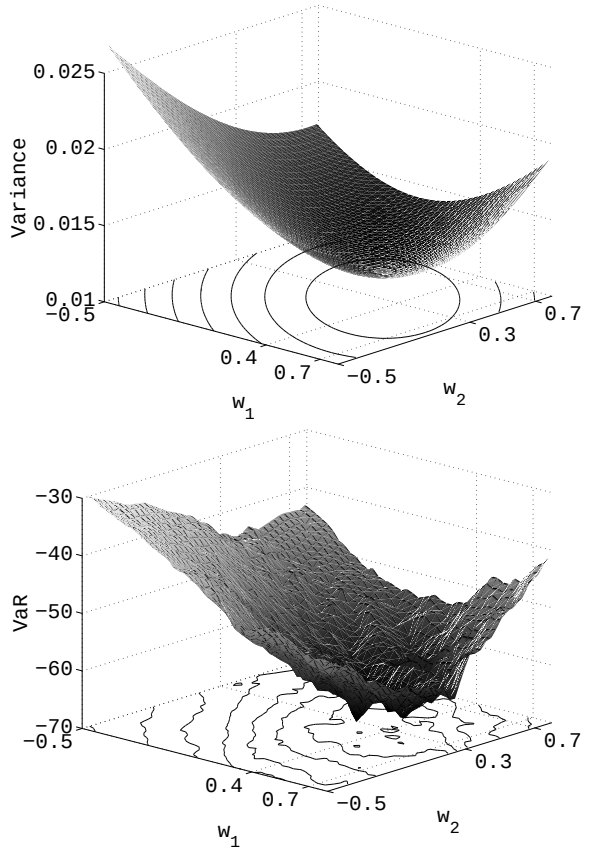

Figure 1: The graphic shows the objective functions of two portfolio-optimization models. In both models, the goal is to minimize the risk of a portfolio of three assets. In the upper panel, we equate risk with return variance. Thus, the function shows the variance of the portfolio for different weights of two assets; the third asset's weight is fixed through the budget constraint (i.e., we cannot invest more wealth than we have). This is the standard model in portfolio optimization, introduced in Markowitz (1952). In order to solve it, a classical optimization technique starts at some point that is specified by the user. Then it moves downhill ("minus the gradient") until at some point the gradient becomes zero: the objective function is flat, and we have arrived at the minimum. That minimum is easily found because the function is smooth and only has one optimum. In fact, Markowitz chose this specification for risk because the function is so well-behaved, not because he considered it the best financial specification. Already in the 1950s Markowitz pondered using downside semi-variance as a measure for risk, but rejected it mainly because he could not find an algorithm to solve the resulting model.
In the lower panel we use the same dataset, but now we define risk as the Value-at-Risk, a quantile of the return distribution. We clearly see that the function is not smooth and has many local minima. A classical method would not be appropriate for such a model, since it would stop at the first minimum that it finds. Heuristics, on the other hand, have been successfully applied to such models. See for instance Gilli and Këllezi (2002) or Dueck and Winker (1992), which is the first application of optimization heuristics to portfolio-selection problems.

not find an improvement.

Classical and heuristic techniques are both based on the same principles. Both types of methods iterate over a sequence of two phases: (i) a creation phase, in which candidate solutions are constructed; and (ii) an acceptance phase, in which it is decided whether the new candidate replaces the previous solution or not. The techniques differ only in the way they implement these phases.

Heuristics spend less effort in creating *a particular* new solution.[5] Instead, heuristics create *many* new solutions; they replace the cleverness of classical methods with brute force. In the past, such reliance on computing power was a severe handicap, but on modern computers – not supercomputers, but the variety that we have on our desktops – that is no longer a problem.

Let us look at two particular techniques.

An example of a classical technique is the steepest-descent method. Given a current solution x^c, we create a new solution x^n by estimating the slope (gradient) of f at x^c, which provides us with the search direction. The new solution x^n is x^c minus a multiple of the slope. Every new solution is accepted and replaces x^c as long as it is better than x^c. We stop when no further improvements in f can be found. (For a more detailed description, see chapter 4 of Gill et al., 1986.)

Using the gradient as a search direction is a reasonable idea, but there are models in which the gradient does not exist or cannot be computed meaningfully. Also, the acceptance-criterion for steepest-descent is strict: if there is no improvement, a candidate solution is not accepted. Yet if the objective function has several minima, such as the function in the lower panel of Figure 1, we will never be able to move away from a local minimum, even if it is not the global optimum.

For such models, we need to give up the idea of "optimal" search directions, and we need to be more lenient when it comes to accepting new solutions. Heuristics do just that.

A well-known heuristic is Simulated Annealing (Kirkpatrick et al., 1983). In this method, the function N randomly chooses a new solution close to the current solution. Thus, we completely dispense with good search directions; any direction will do. If the new solution is better, we accept it. If it is worse, we accept it, too; but only with a specific probability. This probability depends on how bad x^n is: the worse it is, the less likely it is accepted. Also, the probability of acceptance is generally lower in later iterations.

We can think of Simulated Annealing as a biased random-walk through the solution space: the mechanics of the acceptance rule ensure that there is a bias towards better solutions, even though an improvement is not rigidly enforced in every step. But because the routine is repeated often – typically thousands of times –, we are virtually guaranteed a good solution.

3 Optimization cultures

Unlike classical methods, heuristics require essentially no assumptions about the optimization model. They are conceptu-

[5] In this respect, heuristics are similar to a class of numerical optimization methods that were introduced in the 1950s: direct-search methods. This is not to say that there is a historical development from direct search to heuristic methods. Rather, direct-search methods stand somewhere in between classical and heuristic techniques: they are classical because they are still "greedy" and do not rely on any random mechanisms; but they are also simple, make few assumptions about a model, and simply work well in many cases. For an overview, see Wright (1996).

ally simple, which also means that we can easily translate them into computer programs. This flexibility shows itself when we change models, for instance by adding a variable or a new constraint.

Of course, if a model meets all the requirements of a classical method, then using a heuristic will be less efficient – on a relative scale. But computing power is "effectively infinite" (Efron, 2000) when compared with 50 years ago, or at least getting less and less a limitation. Thus, on an absolute scale these efficiency losses often turn out to be small. (In any case, the loss only concerns time-to-compute. See below for other criteria.)

But if heuristics are such powerful methods, how come not everyone is using them? The typical objections against heuristics are these:

Randomness. Heuristics have random elements, and so their solutions are random. We thus cannot know whether we have found the optimum.

Parameters. Heuristics require lots of parameters to be set and adjusted.

Many algorithms. There are so many heuristics, and many seem similar. We thus cannot know which technique is best or at least works well.

Slowness. Heuristics are simply too slow.

Curiously enough, we think that the solution to all these points is the same: experimentation. Let us elaborate, for which we shall discuss those four points in reverse order.

The last objection is easiest to deal with: on modern computers, these techniques simply are not slow. Heuristics may require some programming, such as coding an objective function, and thus the speed with which an algorithm executes may depend on the programming skills of its user.[6]

However, the main misinterpretation that people make is that classical methods require fewer iterations, and hence must be faster. But time-to-compute is only one property of an algorithm. The quality of the model is important, too, and here heuristics are better because they do not restrict the models we can handle.

It also matters how long it takes to develop a tractable model. An example from the financial industry, from Grinold and Kahn (2008, pp. 284–285): The authors describe the implementation of algorithms for an index-tracking problem – finding a portfolio of only 50 stocks that most-closely resembles the S&P500. A specialized, classical algorithm took six months to be developed, and then delivered a solution within seconds. As an alternative, a heuristic (a Genetic Algorithm) was tested as well. Implementing it took two days; the algorithm found similar solutions, but needed two days to compute them. But the example is from the 1990s. Today, on a standard PC, a Genetic Algorithm would compute the solution of such a model in seconds, too. Of course, researchers may have learned since the 1990s, and they may develop models faster today. But it is hard to believe that their performance improvement matches that of computing technology.

In any case, a potential user of heuristics may be puzzled and put off by the large variety of available algorithms. We agree that there is an unhelpful development in academic research to produce ever more new algorithms, which are often only variations of existing ones. (We would prefer more study of existing algorithms.) But the situation is not that bad, actually: there are well-studied and widely-used algorithms available. We have given concrete guidelines elsewhere (Gilli et al., 2011, chapter 12), but the main point to remember is that any quest for a best-performing algorithm is futile. Users should define what they require, then find an algorithm that meets those requirements, and then stop the search. What is important is to gain understanding and intuition about the way the algorithm walks through the search space, by working and experimenting with the algorithm.

But even if a user is willing to experiment with a given algorithm, the troubles seem not to be at an end. For classical methods, including the techniques from mathematical programming, the steps for using them are clearly specified. Once we have a model, there is little more to do than "to press a button." We may exaggerate a bit; but generally no decision about the actual algorithm has to be taken, nor about the parameters, except some tolerances, which almost always are kept at default values. (We do not say that this is good practice, but it is certainly common one; see Altman et al., 2003.) Thus, the user is not embarrassed by any technical choices and the algorithm, in most cases, provides "a solution" when it stops. Also, for classical methods, there is a precise way how to handle constraints, but only those the method can take into account.

All this convenience is missing with heuristics. It is a characteristic of heuristics that they are all based on just a few principles, which is fine if we want to understand how heuristics work. Unfortunately, it is also a downside: think of Simulated Annealing described above, where we said we choose a solution "close to" the current solution. There are clearly many ways to define "close to." Thus, with heuristics, the user has to take many more decisions.

But again, the situation is not that bad. For one, precisely-described algorithms exist, the so-called "canonical versions." Also, many of the newer algorithms, such as Differential Evolution (Storn and Price, 1997), prescribe precisely how new solutions are to be created and require only few parameters to be set. Second, most heuristics are robust with respect to different settings and thus will not fail, even with badly-chosen parameters. And in any case, more computational resources, i.e., more iterations, can typically heal bad choices. Finally, it is actually not that difficult to find good parameter values for a given model – we can run small-scale experiments, and stop when satisfied. In sum: it is true, heuristics are not off-the-shelf methods, and some training is needed. (Such training should start in early education in math classes.) But overall, what is expected from users is manageable, as long as people are not afraid to experi-

[6] This is true for all optimization methods except mathematical-programming. And heuristics allow much more difficult and expensive objective functions in the first place, since they do not restrict the form of the objective function. It is here where experimentation comes in, because producing efficient computer programs is not a mechanical task, but requires repeated experimentation and testing.

ment.

There remains the final – or rather first – objection: heuristics make use of random mechanisms, and thus solutions are random, too. This randomness makes it, allegedly, more difficult to evaluate the quality of solutions computed by heuristic algorithms, as compared with classical methods.

Solutions may be stochastic even with non-stochastic methods. For models that have multiple local minima or are badly-behaved in other ways, repeated runs from different starting points will lead to different solutions. With heuristics, of course, repeated runs even from the same initial solution may give different solutions.

Handling such randomness is extremely straightforward and should not scare anyone who is just remotely acquainted with data analysis. We simply create a sample of solutions by restarting the algorithm a number of times, and then analyze that sample. Obviously, the best result constitutes the overall solution. We can refine the result as much as we want, typically by allowing more iterations. Importantly, through such analysis we can explicitly explore the trade-off between solution quality and computational resources. We stop when the solution is "optimal enough" for the problem at hand.

Indeed, restarting the optimization algorithm is good practice not only for heuristics, but for classical methods as well. The result of a classical method should be validated by restarting the optimization in a neighborhood of the original initial solution. This would confirm that we are in an at least locally-convex region of the objective function. In particular if the model turns out to be non-nonvex, restarting the optimization procedure from various initial values and keeping the best solution is a useful and often surprisingly-successful strategy. (As a side note: such a best-of-N–restarts strategy is a perfect example for our suggestion to treat optimisation as a computational tool, rather than a mathematical one. It would be very straightforward to wrap a classical algorithm – in fact, any algorithm – into a restart-loop and then report the best solution and also the variation among the solutions. But that is rarely done. See also Gilli and Schumann, 2010.)

The deeper causes

The skepticism towards heuristics is only a symptom; the underlying causes run much deeper. For generations, the typical syllabus in education has treated optimization as a purely-mathematical problem. That leads to two views that are in conflict with heuristics. First, mathematics teaches us coherence and that results must be proven. Such formal proofs do not exist for heuristics, or at least not in the usual explicit form we have been used to see in school. As a result, heuristics are instinctively rejected as they are not considered to be based upon solid foundations. (It is remarkable that even for classical methods, we often have only probabilistic guarantees, and sometimes not even that. As an example, think of the workhorse algorithm for linear programming, the simplex method. It has been shown that in the worst case the algorithm's required iterations increase exponentially with problem-size; see Klee and Minty, 1972. But *experience* shows that the algorithm works well.)

The second implication is subtler. Mathematics is an exact discipline, and so there has, apparently, to be an exact solution to an optimization model. This theme is ubiquitous in all disciplines that use mathematics to formulate their ideas, and it leads to the unwarranted intuition that when a mathematical model is used to describe a problem, the model's solution inherits the exactitude of the model. But it does not: as we said before, a model is just a representation, not the actual problem. Thus, the solution, no matter how exact, is the solution to the model and not to the problem. With luck, the quality of the model-solution is correlated with that the of problem-solution, but that is not guaranteed (and rarely investigated). In any case, the correlation must disappear beyond a certain precision threshold. Of course, we do not know this threshold. But we can use the tools of data analysis to explore it. For that, we need to understand the computational part of optimization, but also the actual problem.

4 Conclusions

In this essay we have tried to make the case for preferring heuristics over classical optimization methods. Heuristics allow their users to specify models without restricting the functional form of the objective function or the constraints. Putting it somewhat exaggeratedly, they do so by replacing sophistication with brute force. Yet, as we pointed out, brute force is so cheap and so ubiquitous today that such an exchange entails little downside.

Ripley (1998) nicely summarizes the characteristics of computer-intensive methods in this way: (i) Simple calculations are repeated many times; (ii) assumptions can be relaxed; and (iii) with infinite resources the solutions become exact. This summary also fits heuristics. The third point, however, requires a qualification. Optimization is an applied discipline; optimization algorithms are tools. Thus, "exact" is not required; "good enough" is all that is needed (Gilli and Schumann, 2011). Perversely, a numerically-precise solution to a model may not even add quality, but give an unwarranted feeling of being on the safe side. After all, a numerical solution only relates to the model, not to the actual problem.

How a model and its solution help with the actual problem needs to be explicitly studied. Such analysis typically is difficult and not very precise, but carefully exploring, quantifying and discussing the effects of specific model choices is always better than dismissing such an analysis as "out-of-scope."

Zanakis and Evans (1981) list situations in which using a heuristic is "desirable and advantageous"; the first item on their list being "Inexact or limited data used to estimate model parameters may inherently contain errors much larger than the 'suboptimality' of a good heuristic." Few people would disagree with such a prescription. Yet it is difficult to find examples in the literature in which such an analysis is actually done. We conjecture that this case applies in many scientific disciplines, and if people analyzed their applications properly, they would find that highly-precise solutions are rarely needed, and hence neither are highly-precise methods. Again more reason to move away from classical methods and towards heuristics. The costs of such a shift would be low: we would have to give up some mathematical elegance and specious precision, and replace them with more

computation and experiments. In turn, we would stand to gain a lot: better solutions, not to models, but to actual problems.

References

Micah Altman, Jeff Gill, and Michael P. McDonald. *Numerical Issues in Statistical Computing for the Social Scientist*. Wiley, 2003.

Leo Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16(3):199–231, 2001.

Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Hank A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, 1998.

Gunter Dueck and Peter Winker. New concepts and algorithms for portfolio choice. *Applied Stochastic Models and Data Analysis*, 8(3):159–178, 1992.

Bradley Efron. The bootstrap and modern statistics. *Journal of the American Statistical Association*, 95(452):1293–1296, 2000.

Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Elsevier, 1986.

Manfred Gilli and Evis Këllezi. A Global Optimization Heuristic for Portfolio Choice with VaR and Expected Shortfall. In E. J. Kontoghiorghes, B. Rustem, and S. Siokos, editors, *Computational Methods in Decision-making, Economics and Finance*, Applied Optimization Series, pages 167–183. Kluwer Academic Publishers, 2002.

Manfred Gilli and Enrico Schumann. A note on 'good starting values' in numerical optimisation. *COMISEF Working Paper Series No. 44*, 2010. available from `http://comisef.eu/?q=working_papers`.

Manfred Gilli and Enrico Schumann. Optimal enough? *Journal of Heuristics*, 17(4):373–387, 2011.

Manfred Gilli, Dietmar Maringer, and Enrico Schumann. *Numerical Methods and Optimization in Finance*. Elsevier/Academic Press, 2011.

Richard C. Grinold and Ronald N. Kahn. *Active Portfolio Management*. McGraw-Hill, 2nd edition, 2008.

Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

Victor Klee and George J. Minty. How good is the simplex algorithm? In Oved Shisha, editor, *Inequalities III – Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, September 1–9, 1969*, pages 159–175. Academic Press, 1972.

Harry M. Markowitz. Portfolio selection. *Journal of Finance*, 7 (1):77–91, 1952.

Brian D. Ripley. Computer-intensive methods. In P. Armitage and Theodore Colton, editors, *Encyclopedia of Biostatistics*. Wiley, 1998.

Rainer M. Storn and Kenneth V. Price. Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 341–359, 1997.

Jan Tschichold. Buchherstellung als Kunst. In Richard von Sichowsky, editor, *Typographie und Bibliophilie: Aufsätze und Vorträge über die Kunst des Buchdrucks aus zwei Jahrhunderten*. Maximilian-Gesellschaft, 1971.

Margaret H. Wright. Direct search methods: Once scorned, now respectable. In D.F. Griffiths and G.A. Watson, editors, *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis*, pages 191–208. Addison Wesley Longman, 1996.

Stelios H. Zanakis and James R. Evans. Heuristic "optimization": Why, when, and how to use it. *Interfaces*, 11(5):84–91, 1981.