A Personal Perspective on Evolutionary Computation: A 35-Year Journey

Zbigniew Michalewicz[*] zm@complexica.com
Complexica, Level 2, 9 Charles Street, West Lakes, SA 5021 Australia
www.complexica.com

https://doi.org/10.1162/evco_a_00323

Abstract

This paper presents a personal account of the author's 35 years "adventure" with Evolutionary Computation—from the first encounter in 1988 and many years of academic research through to working full-time in business—successfully implementing evolutionary algorithms for some of the world's largest corporations. The paper concludes with some observations and insights.

1 The Discovery (1988–1990)

Let's start at the very beginning, a very good place to start.[1]

During the academic year of 1987/88, I took a sabbatical leave from the Victoria University of Wellington (I was lecturing mainly on database systems, which was my main research area at that time) and joined for a year the faculty of the Department of Computer Science at the University of North Carolina at Charlotte (again, teaching mainly database courses on graduate and undergraduate levels). However, halfway through my sabbatical, around January 1988, I attended a seminar where the speaker talked about *genetic algorithms*. It was the first time in my life that I heard this term ... *and I loved it!* First, I learned that this was the best technique ever for search, optimization, and machine learning. It can be applied successfully to any problem. Second, the name of the technique, genetic algorithm, was absolutely fantastic; it triggered the imagination and piqued curiosity.[2] Whoever came up with this term was a brilliant marketer!

Anyway, all this sounded too good to be true, so on my return to Victoria University in June of 1988, I decided to run some experiments to check the claims I heard during the seminar. For some reason (which I cannot recall today), I selected one of the classic operation research problems, a transportation problem, for my experiments.

The transportation problem is a relatively simple combinatorial optimization problem; however, it includes several constraints. We must determine the overall minimum cost of the transportation plan for a single commodity from several sources to

[*]Also, at the Polish-Japanese Academy of Information Technology, ul. Koszykowa 86, 02-008 Warsaw, Poland; the Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland; and the School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia.

[1]Readers of my generation may recognize this sentence from the movie, *The Sound of Music*.

[2]Only years later, I found out that similar techniques were already experimented with for a couple of decades, but the terms used were not that appealing. Indeed, even in science, "marketing" is important; we experience it all the time, for example, watching transformation of "statisticians" into "data analysts" and then into "data miners" and finally into today's very sexy "data scientists."

Manuscript received: 29 December 2022; accepted: 29 December 2022.

several destinations. The problem specifies the level of supply at each source, the level of demand at each destination, and the transportation cost from each source to each destination.

There is only one commodity, so any destination can receive its required demand from one or more sources. The objective is to find the "best" transportation plan, which is represented as the amount to be shipped from each source to each destination such that the total transportation cost is minimized. If the cost of transport is directly proportional to the amount transported, then the transportation problem is *linear*; if the cost of transport isn't directly proportional to the amount transported, then the transportation problem is *nonlinear.* Linear transportation problems can be solved through linear programming methods; however, nonlinear problems lack a general solving methodology.

To illustrate a simple transportation problem,[3] we can consider a small example (Michalewicz, 1992) with three sources (rows) and four destinations (columns):

	5.0	**15.0**	**15.0**	**10.0**
15.0	4.1	1.9	7.3	1.7
25.0	0.5	12.1	7.4	5.0
5.0	0.4	1.0	0.3	3.3

where the boldface numbers in the first column represent the levels of supply across three sources, the boldface numbers in the first row represent the levels of demand across four destinations, and the twelve numbers in the body of the matrix represent the transportation plan: the amounts to be transported from each source to each destination. In the above example, 7.4 units are transported from the second source (second row) to the third destination (third column). There is also a set of twelve cost functions that represent the various transportation costs between every source and every destination. These cost functions are used to evaluate possible transportation plans, which are the candidate solutions. My goal was to develop a genetic algorithm to solve such a basic transportation problem. However, at that time I had only one resource available (Davis, 1987) as a reference. A few articles (written by different authors) that were included in this edited volume were extremely interesting (Grefenstette, 1987),[4] but they didn't answer several questions I had. In particular, the main issue I was struggling with was the issue of constraints: this transportation problem was constrained on supply (you can't supply more than you have) and on demand (you must deliver the required amount), and I was not sure how to deal with these constraints.

Only some months later, I got a copy of the first text on genetic algorithms (Goldberg, 1989). I was eagerly searching the book for some answers; however, I found only a couple of paragraphs (spread over pages 85 and 86, out of 432 pages of the whole book) on how to handle constraints:

Thus far, we have only discussed genetic algorithms for searching unconstrained objective functions. Many practical problems contain one or more constraints that must be also satisfied. In this section, we consider the incorporation of constraints into genetic algorithm search.
. . .
At first, it would appear that inequality constraints should pose no particular problem. A genetic algorithm generates a sequence of parameters to be tested using the system model,

[3]This transportation problem is *balanced*: the supply and demand totals are the same.

[4]This chapter was of particular value. I believe it gave me some initial insights at that time. However, the chapter considered the Traveling Salesperson Problem for which a vector of integers was used.

objective function, and the constraints. We simply run the model, evaluate the objective function, and check to see if any constraints are violated. If not, the parameter set is assigned the fitness value corresponding to the objective function evaluation. If constraints are violated, the solution is infeasible and thus has no fitness. This procedure is fine except that many practical problems are highly constrained; finding a feasible point is almost as difficult as finding the best. As a result, we usually want to get some information out of infeasible solutions, perhaps by degrading their fitness ranking in relation to the degree of constraint violation. This is what is done in a *penalty method*.

In a penalty method, a constrained problem in optimization is transformed to an unconstrained problem by associating a cost or penalty with all constraint violations. This cost is included in the objective function evaluation.

In other words, a constrained optimization problem:

optimize $f(\mathbf{x})$

subject to $c_i(\mathbf{x}) \geq 0$ for $i = 1, 2, \ldots, k$

where $\mathbf{x}$ is a vector $<x_1, x_2, \ldots, x_n>$

is transformed into unconstrained form:

optimize $f(\mathbf{x}) + r \times \Sigma_{1 \leq i \leq k}\ \Phi[c_i(\mathbf{x})]$

where r is a penalty coefficient and Φ is a penalty function.

Then the concluding paragraph of the section on incorporation of constraints into genetic algorithm search in David Goldberg's book was:

A number of alternatives exist for the penalty function Φ. In this book, we usually square the violation of the constraints, $\Phi[c_i(\mathbf{x})] = [c_i(\mathbf{x})]^2$, for all violated constraints i. Under certain conditions, the unconstrained solution converges to the constrained solution as the penalty coefficient approaches infinity. As a practical matter, r values in genetic algorithms are often sized separately for each type of constraint so that moderate violations of the constraints yield a penalty that is some significant percentage of a nominal operating cost.

The advice sounded reasonable; so finally, I got my answer. At this stage I had everything I needed to start experimenting. A binary string (010001011001010001 . . . 0100111) was used[5] to represent a transportation plan (this was the easy part of arranging all floating-points in the transportation matrix into a sequence and converting each number into a binary string, with some assumed precision). For my experiments, I also created a few sets (each set consisted of some linear and nonlinear transportation cost functions) that would contribute to the evaluation score. Then, to deal with the problem-specific constraints, I built a family of penalty functions that penalized the evaluation score for violations of constraints, and I ran many experiments with various weights for penalty coefficients. To make the story short, the results were quite disappointing. In general:

[5]In his book, David Goldberg presented a strong argument as to why bit-strings are the best method of encoding parameters in genetic algorithms (this is also known as the principle of "minimal alphabet").

- √ Large penalties resulted in a low-quality solution; basically, the first feasible solution found was selected as the final solution. The algorithm was unable to escape a feasible solution and search for a better one, as this required crossing infeasible areas of the search space, and these areas were penalized heavily.
- √ Small penalties resulted in a "funny" outcome: usually the optimal transportation plan was "all zeros"—do not transport anything anywhere! Indeed, you cannot do better than that from the perspective of cost, and it was also a reasonable plan as penalties for constraint violation were relatively small.

I also experimented with dynamic penalties, where penalty coefficients changed every number of iterations of the algorithms (without much improvement), as well as with repair algorithms to return to feasibility after the standard operators were applied; however, the effort to "repair" a solution was significant and after a while I abandoned this approach as well. Note that a single mutation (i.e., change of a single bit in a binary string) applied to a feasible solution would trigger a series of changes in different places of the string (at least in three other places) to preserve the feasibility of the solution. The situation was even more complex when I tried to repair a solution that was an offspring that resulted from crossover, as such offspring would violate numerous constraints.

Only then (out of desperation, I think) I considered the following idea: what would happen if we departed from binary representation? Clearly, the main consequence of such a decision would be the need for a new set of operators transforming one (or more) solutions into new solutions. Fortunately, this task (finding new feasibility-preserving operators) in the case of the transportation problem was relatively straightforward. With two feasible matrixes, M_1 and M_2, a crossover operator was introduced that resulted in two offspring:

$$\mathrm{O}_1 = c * \mathrm{M}_1 + (1 - c) * \mathrm{M}_2$$

$$\mathrm{O}_2 = (1 - c) * \mathrm{M}_1 + c * \mathrm{M}_2$$

where $0 < c < 1$ was a parameter of the operator. Note that if M_1 and M_2 are feasible, then O_1 and O_2 are also feasible.

After some experimentation, two mutation operators were defined. The first mutation explored the boundary of the feasible search space by introducing as many zero entries into the matrix as possible (still preserving feasibility). The second mutation followed the original idea of mutation: it made a small change. It selected two random rows and two random columns from the original parent matrix (thus creating a mini 2×2 matrix), recording its totals form both rows and both columns, and re-initializing all four values in such a way that the totals remained the same. These four new values replaced the old values in the original matrix, and of course the resulting solution was also feasible.

The experimental results were "interesting" (at least for me at this stage). First of all, my nonstandard genetic algorithm worked. The system was converging nicely on the optima (which is known in the case of the linear version of this problem) or outperforming other nonlinear programming methods for the nonlinear versions of this problem. However, I wasn't sure whether the developed system deserved the name "genetic algorithm." First, a matrix representation was used. Second, the usefulness of crossover was negligible with respect to mutation operators—contrary to my belief that crossovers were the "main" operator (responsible for mixing building blocks of the solution) while mutation was only a "background" operator (guaranteeing that no bit is “lost”). I couldn’t imagine reporting any experimental results with the probability of crossover set to zero (i.e., running the system without crossovers). However, to my great relief, the system generated the best results when the probability of crossover was greater than zero—only 0.05, but still—so I could report experimental results with both operators present. Third, there were some additional features of this nonstandard system that emerged from experimentation. For example, in each generation I applied just one operator and the offspring replaced one individual in the population (later this feature was labelled as *steady state*). Also, it was frustrating to observe that in some generations the system moved “backwards,” in the sense that the best solution found thus far was lost (replaced by a weaker offspring), so I modified the system to keep the best individual no matter what (later this feature was labelled as *elitism*).

At this stage, the lessons and insights that emerged for me from these early experimentations (which shaped the next few years of my research) were:

- √ It is possible to develop an evolutionary system (meaning: a population-based method) for a particular problem, where candidate solutions are modified by some unary (mutation) and binary (crossover) operators. Such an evolutionary method could be quite powerful, and, if applied correctly, could yield good results. The knowledge of the problem is incorporated in the system by means of data structures used to represent candidate solutions and operators which modify them.

- √ It seems that the representation of a solution is the key; this is somewhat equivalent to “understanding the problem” that we’re trying to solve.

- √ The variation operators are important; again, these transformations of various solutions should (possibly) incorporate some characteristics of the problem. For example, in the experiments I conducted on the simple transportation problem it was useful to introduce a special mutation operator (a couple of years later labelled: *boundary mutation*) that explored the boundary of the search space rather than its interior. This was an interesting concept, because for many real-world problems the best solutions are located at the boundary of the feasible search space (as constraints often represent limitations in resources and at least one resource is usually pushed to the limit to generate the best solution).

- √ Finally, it is hard to overstate the importance of the constraint handling technique used for dealing with constrained optimization problems. There are hardly any real-world optimization problems that aren’t constrained in some way, so “processing constraints” is very important. If we don’t handle this issue correctly, the chances are that the developed system would waste too much time on sampling infeasible solutions. Only later did I discover a wealth of constraint-handling techniques: dynamic and adaptive penalties, repairs (including Baldwin’s effect, Lamarckian evolution, Davis’s 5% rule), feasibility-preserving operators, decoders, etc.

Later in this paper, I’ll refer to the above points and, in a sense, these insights are as important for me today (when building real-world applications), as they were 30 years ago!

Let me conclude this section by describing one event that took place in September 1989, just two months after I moved from the University of Wellington in New Zealand to the University of North Carolina at Charlotte (the place of my earlier sabbatical). One of my friends from George Mason University invited me to give a talk at his department (Computer Science), so I thought that I would take this opportunity to discuss genetic algorithms and my first experiments: my very first talk in my new research area! However, minutes before my talk I found out that famous[6] Ken De Jong with his PhD students would attend my talk (I wasn't aware of that earlier and not sure whether I would have accepted the invitation if I knew). Anyway, the talk went very well. I got a few very useful comments and suggestions and since that time we have been staying in a close touch. It was one of the nicest experiences in my professional career.

2 Early Research Years (1991–1993)

The next three years were very important. First, I was already in the United States, so I was much closer to all events related to evolutionary algorithms. In July 1991, in San Diego, I attended the Fourth International Conference on Genetic Algorithms (ICGA), which was my first conference in this area. I met the top researchers at that time and could finally put faces to the authors of research papers that I had been reading. Hans-Paul Schwefel attended this conference, and so I also "discovered" evolutionary strategies, an evolutionary technique developed earlier in Germany. I was thrilled to see that evolutionary strategies used "natural representation": a vector of floating-point numbers for numerical optimization. I was convinced this was "the way to go." After all, why would we convert a floating-point number into a binary string, triggering some issues (e.g., the precision) for processing? Further, I was pondering about this very question for some months now; at this conference one of my papers (Janikow and Michalewicz, 1991) compared the efficiency of binary vs. floating-point representations for numerical optimization; the other paper (Michalewicz and Janikow, 1991) presented my thoughts on constraint-handling methods other than penalty functions (which were so disappointing in my early experiments). Also, in 1991, my first two journal papers on genetic algorithms were published (Vignaux and Michalewicz, 1991; Michalewicz et al., 1991), which described my early experiments on transportation problems. At that time, as I explained earlier in this paper, I wasn't sure whether the systems I developed deserved the name "genetic algorithms" (hence the term "nonstandard genetic algorithm" in the title of the second paper). After all, I departed from binary representation and the operators used were also nonstandard, as they included, for example, a repair mechanism to stay in the feasible part of the search space. On top of that, the usefulness of crossover operators was minimal, and at that time I was convinced that crossovers "must be there" as the main operator of the method.

Another landmark of this period was creation of the GENOCOP system (for GENetic Optimizer for COnstrained Problems[7]). In 1991, I made the source code of the system available to all researchers, and the original version (which was followed by many additional versions) is still available today at my University of Adelaide website

[6]Ken De Jong's doctoral dissertation from 1975, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, (together with John Holland's *Adaptation in Natural and Artificial Systems* published the same year), is often considered as a birthday of genetic algorithms (De Jong, 1975; Holland, 1975). My seminar talk at George Mason University happened fourteen years later!

[7]Clearly, I was influenced by a 1987 movie, *RoboCop*.

(https://cs.adelaide.edu.au/~zbyszek/).[8] I believe the system has outlived its usefulness today, but at that time, it had some merit. It was the first system (as far as I was aware) in which the user could define several floating-point variables, define the domain of each variable (from–to), define the objective function, and define several linear constraints. No need to worry about penalties, and no need to worry about repairs. There were also some parameters that the user could set, like the population size, the total number of iterations, probabilities of operators used (there were six operators in GENOCOP), as well as a parameter to define the strength of the selective pressure. An article on GENOCOP was submitted for publication to the *Communications of the ACM* in 1991 and accepted the same year, but due to some ACM policy changes at that time, only a short abstract was published after five years of delay (Michalewicz and Janikow, 1996)!

During the next few years, I created additional versions of GENOCOP and put them into the public domain—versions with improved operators, with improved initialization methods and selection routines. The best version of this family of systems (note that only linear constraints were allowed) was GENOCOP 3.0, a mature version that's been used by hundreds[9] of researchers.

However, the highlight of this period 1991–1993 was the publication of my first book, *Genetic Algorithms + Data Structures = Evolution Programs* (Michalewicz, 1992). The title of the book was inspired[10] by a well-known text from 1976, *Algorithms + Data Structures = Programs* written by Niklaus Wirth (Wirth, 1976). Wirth's book covered some of the fundamental topics of computer programming with an emphasis on the point that algorithms and data structures are inherently related. I thought that the systems I was developing shared a similar theme: the operators used in these "genetic algorithms" and the data structures representing candidate solutions were also inherently related.

At around that same time, the first journal related to evolutionary algorithms was also established. I was lucky to submit my paper then (Michalewicz, 1993), which was accepted for the first issue[11] of this new journal.[12] The paper included some experimental results (again, on transportation problems) together with some insights. The abstract of this paper was as follows:

In this paper we present the concept of evolution programs and discuss a hierarchy of such programs for a particular problem. We argue that (for a particular problem) stronger evolution

[8]This website also includes the original source codes used for experimentation on linear and nonlinear transportation problems.

[9]This is just an estimation based on the number of e-mails I was getting around that time; even this year I got two inquiries about GENOCOP!

[10]Eight years later, I published (with D. Fogel) a textbook, *How to Solve It: Modern Heuristics* (Michalewicz and Fogel, 2000). Here, the inspiration for the title also came from an outstanding book: *How to Solve It* (1945) by George Pólya (Pólya, 1945). In this book, Pólya identified basic principles of problem solving; later I dedicated one of my books, *Puzzle-Based Learning* (Michalewicz and Michalewicz, 2008), to him.

[11]This is the reason behind this invited paper: to reflect on how the field has developed with respect to that original contribution in the first issue over the last 30 years.

[12]Four years later, in 1997, the second journal in evolutionary algorithms was established: *IEEE Transactions on Evolutionary Computation*. Again, I was lucky to get my paper accepted for the very first issue (Xiao et al., 1997). In a few years it will be the 30th anniversary of that journal—so it might be that I will be invited to write another paper on evolutionary computation.

programs (in terms of the problem-specific knowledge incorporated in the system) should perform better than weaker ones. This hypothesis is based on a number of experiments and a simple intuition that problem-specific knowledge enhances an algorithm's performance; at the same time, it narrows the applicability of an algorithm. Trade-offs between the effort of finding an effective representation for general-purpose evolution programs and the effort of developing more specialized systems are also discussed.

The paper summarized my earlier experiments on transportation problems and expanded on my simple intuition that problem-specific knowledge (in terms of the data structures used, operators, and constraint-handling techniques) enhances an algorithm's performance. The "more" problem-specific knowledge is incorporated into the algorithm, the "better" the results. The paper presented five versions of evolutionary algorithms with an increasing amount of problem-specific knowledge incorporated into each algorithm (hence the term "hierarchy" in the title of the paper); indeed, the results for "higher" versions were better than those for "lower" versions. The argument was largely intuitive, as it is difficult to measure the "amount" of problem-specific knowledge that's incorporated into an algorithm. Nevertheless, this simple observation that was presented in this paper influenced my research for the rest of my career. Later, I found this research useful for various commercial applications.

3 Academic Life (1994–1999)

During the next six years, I continued with standard academic life: attending conferences, publishing papers, supervising students. An important milestone was reached in June 1994: the First IEEE Congress on Evolutionary Computation (Orlando, June 1994) took place (as a part of the IEEE Congress on Computational Intelligence), and I was the general chair of the evolutionary part. This congress set the foundation for the *IEEE Transactions on Evolutionary Computation* (the first issue appeared three years later, in 1997). Similarly, some discussions during this congress with editors of the Oxford University Press and Institute of Physics resulted in a very interesting publication (Bäck et al., 1997), where around 100 researchers contributed to this large volume (around 1,000 pages). This Handbook represented a milestone for the field of evolutionary computation; a quote from the Foreword of this volume explained it all:

However, within the field there was a growing sense of the need for more interaction and cohesion among the various subgroups. If the field as a whole were to mature, it needed a name, it needed to have articulated cohesive structure, and it needed a reservoir for archival literature. The 1990s reflect this maturation with the choice of *evolutionary computation* as the name of the field, the establishment of two journals for the field, and the commitment to produce this handbook as the first clear and cohesive description of the field.

Further, I continued my work on additional versions of the GENOCOP system. Again, the general idea was to keep the objective and set of constraints as independent entities, so we could add or remove constraints at will without affecting other components of the algorithm; later, within business environments, this line of reasoning proved very useful. Still, the next two versions of the system that were put into the public domain provided improvements in terms of operators and types of variables:

√ GENOCOP 3.1 provided self-tuning probabilities of operators. Note that all previous versions of the system required setting the frequency for all six operators (these were called uniform mutation, boundary mutation, non-uniform mutation, arithmetical crossover, simple crossover, heuristic crossover) which wasn't convenient. This research later resulted in a publication on parameter control in evolutionary algorithms (Eiben et al., 1999).

√ GENOCOP 4.0 added the possibility to declare integer/Boolean variables. This was missing in all previous versions and many test cases required this feature.

However, all versions of GENOCOP presented so far handled linear constraints only, so it was time to turn attention to nonlinear constraints. Two versions of the GENOCOP system were released (referred to as GENOCOP II and GENOCOP III) that handled nonlinear constraints, and a bit later, the final version of the system was placed in the public domain[13]:

√ GENOCOP 5.0 allowed for the handling of nonlinear constraints through the use of a decoder.

The issue of constraints in evolutionary algorithms remained one of my main research topics; during that period, I worked closely with Marc Schoenauer and this cooperation resulted in a few papers (Michalewicz and Schoenauer, 1996; Michalewicz et al., 1996; Schoenauer and Michalewicz, 1996, 1997, 1998).

Some of these publications dealt with situations where the optimum solution lies on the boundary between feasible and infeasible parts of the search space; again, from the perspective of real-world problems, this was a very fruitful research direction. Fifteen years later, with two other researchers, I returned to this topic, and our paper pointed out that this area is significantly under-explored in research (Bonyadi, Wagner et al., 2014). I believe that we should analyze the impact of constraints more in the future, as it also goes towards decision support, albeit more strategic than operational. There was also one additional paper (Michalewicz et al., 2000) worth mentioning in the context of research on constraints in evolutionary algorithms, as it discussed a test-case generator for constrained parameter optimization techniques that could create various test problems with various characteristics—for example, test problems with different relative size of the feasible region within the search space, problems with different number and types of constraints, problems with convex and non-convex objective functions (possibly with many optima), problems with highly non-convex constraints consisting of (possibly) disjoint regions. The idea was to get some insights into why some constraint-handling techniques (e.g., penalties, decoders, repairs) performed well on some constrained problems but not others.

The paper on the test-case generator for constrained parameter optimization techniques was completed during the academic year of 1998/99, which I spent at Aarhus University in Denmark (sabbatical leave). During this stay (great place, great crowd of graduate students) I was also working on a new book (Michalewicz and Fogel, 2000). This work, in some sense, closed the period of 12 years during which I was fully immersed in research on evolutionary algorithms in the context of academic life, and little did I know about "real-world applications." Of course, there were a few publications on real-world applications of evolutionary algorithms; almost every conference had a session on real-world problems and almost every paper on evolutionary algorithms talked about applicability of these algorithms to problems in real-world

[13]This version was based on a method described in a paper by Kozieł and Michalewicz (1999).

settings (usually within the introductory section of the paper). Even myself, I co-edited an application-oriented book (Dasgupta and Michalewicz, 1997) and also wrote a few papers where evolutionary algorithms were applied to problems in real-world settings (the early papers on the transportation problem also fell into this category). And so I published many papers where evolutionary algorithms were applied to evolving trading rules (Ghandar, Michalewicz, Schmidt et al., 2009), portfolio management (Ghandar, Michalewicz, and Zurbruegg, 2009), forecasting economic time series (Wagner et al., 2008), strategic decision support (Johnson et al., 2005), material flow in supply chains (Vergara et al., 2002), control tasks in dynamic systems (Ursem et al., 2002), joint replenishment problem (Khouja et al., 2000), design of electromagnetic devices (Wieczorek et al., 1998), economic lot and delivery scheduling problem (Khouja et al., 1998), and modeling of ship trajectory in collision situations (Śmierzchalski and Michalewicz, 2000). All these papers had a real-world flavor; however, they were not really "real-world applications."

I began to realize that the term "real-world application" was used in the evolutionary computation community somewhat arbitrarily. Some researchers just experimented with applications (e.g., numerical optimization, constraint-handling, multiobjective optimization) that had the label "real-world." Other researchers experimented with applications that were tested on some known model (e.g., TSP, VRP, JSSP) of a real-world problem. And some other researchers tested their applications on some real data (e.g., taken from a hospital, city council, a particular business organization). However, it was very hard to find any application based on evolutionary algorithms that was used in some business or industry on a daily (regular) basis.

At that time, I was thinking about large-scale software systems where evolutionary algorithms could play a significant role, systems that could be used on a regular basis by staff within large organizations and where the recommended "optimal solutions" were considered in the final decisions, systems that could successfully compete with software provided by big players (e.g., Microsoft, SAP, or Oracle). In other words, I was after evolutionary algorithms that were used in "real action" as opposed to, for example, some off-line applications for making a one-time decision (such as a design decision). I thought that the two sisters[14] of evolutionary computation (artificial neural networks and fuzzy systems) broke into the mainstream of industry much earlier, and it would be good if evolutionary computation joined them. Anyway, my desire of doing something "for real" was growing stronger and stronger, and the opportunity presented itself in early 2000.

4 From the Halls of Academia into the Skyscrapers of Business (2000–2012)

In January 2000 I co-founded my first business, NuTech Solutions. The timing seemed to be good. The dot-com boom of the late 1990s facilitated a massive increase in start-up companies, and between 1995 and its peak in March 2000, the Nasdaq rose 400%.[15] It was boom times for technology companies. The general idea behind NuTech Solutions was to apply science (in particular, evolutionary methods) to solve hard optimization

[14]Traditionally the three main pillars of Computational Intelligence (understood as the theory, design, application, and development of biologically and linguistically motivated computational paradigms) have been neural networks, fuzzy systems, and evolutionary computation.

[15]Only to fall 78% from its peak by October 2002, giving up all its gains during the bubble.

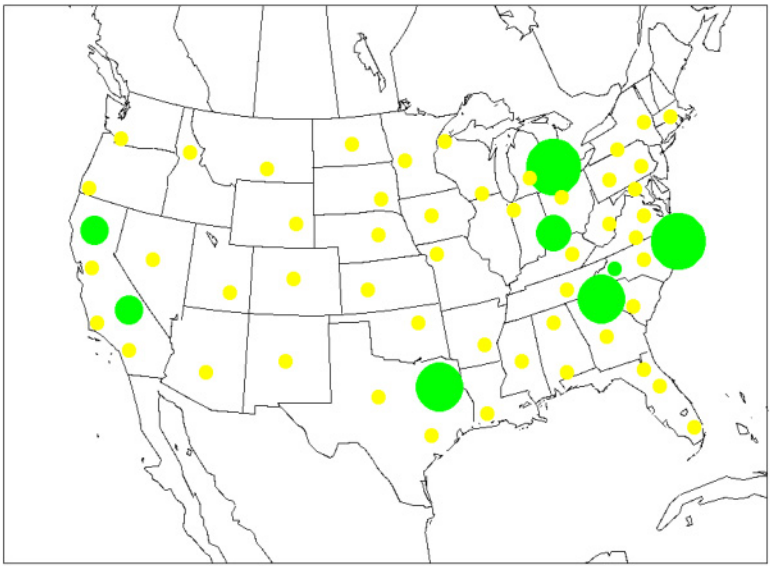


Figure 1: Off-lease cars and auction sites on one particular day.

problems for industry. Little did I know about running a business at that time, but it was a great adventure and a great learning experience.[16]

This event also started my gradual transition from academia into business. On the one hand, during the next twelve years, 2000–2012, I was still in academia (at the University of North Carolina at Charlotte and later, from 2005, at the University of Adelaide, Australia), but on the other hand, during these twelve years there were periods of time when I worked either part-time in my own company or was on leave from university. After all, it is quite difficult to maintain your academic activities and run a business at the same time.

Without going into too much detail, let me present one project (from 2001) that provided me with many observations and insights.

4.1 Car Distribution System

The project was for GMAC, a car financing organization in the United States that was leasing around one million cars each year to consumers, organizations, and rental agencies (see the case study in Michalewicz et al., 2005). When a car lease agreement expired, which could be from one to five years, the car was either returned to GMAC or purchased by the lessee (in either case, these cars are called *off-lease cars*). GMAC didn't need to worry about the purchased off-lease cars, but it needed to sell the returned off-lease cars at one of many available auction sites located across the United States. Each of these returned cars was different in its make, model, body style, trim, color, year, mileage, and damage level, and the overall number of cars leased each year translated into approximately 5,000 returned off-lease cars each day. Figure 1 illustrates a

[16]Some years later we described our experiences in *Winning Credibility: A guide for building a business from rags to riches*, (Michalewicz and Michalewicz, 2006). The book may serve as a guide on how to start and run a business; it was used as a text in business schools at a few universities.

particular day, where green circles represent the returned off-lease cars and yellow circles represent the 50 auction sites at which GMAC sells its cars.

The larger the green circle, the more cars were returned at that location, with the sizes and locations of these circles varying from one day to the next (as different people and organizations returned their cars at different locations). The yellow circles, on the other hand, represent the designated 50 auction sites where the returned off-lease cars were sold. The locations of these auction sites were fixed.[17]

GMAC's task was to distribute this daily intake of approximately 5,000 cars to the 50 designated auction sites; in other words, to assign an auction site to each off-lease car. For example, if the first car were located at a dealership in Northern California, GMAC would consult some reports[18] on what the average sale price for that car was at different auction sites (after adjusting for mileage, trim, damage level, etc.), and then ship that car to the auction site with the highest average sale price. Of course, GMAC also needed to estimate the transportation cost to each auction site (the longer the distance, the higher the cost, and longer transportation times also resulted in higher depreciation and damage risks). Although straightforward, this approach for distributing off-lease cars didn't allow GMAC to capture the full value of each off-lease car. Because the entire process was based on manual analysis and individual, car-by-car decisions, any small mistake that resulted in a net reduction of "only" \$50 per car would cost GMAC \$250,000 in a single day.

As such, GMAC defined their business problem and objective as:

Maximize the aggregate resale value of all returned off-lease cars by optimizing the distribution of individual cars to individual auction sites.

This was a difficult business problem to solve and objective to realize, for the following reasons:

1. *Number of possible solutions.* There were 50 possible solutions for each individual car, as GMAC could ship a car to any of the 50 auction sites; for two cars, there were 2,500 possible solutions (50×50); for three cars, 125,000 possible solutions ($50 \times 50 \times 50$), and so on. For 5,000 cars, however, there were approximately $50^{5,000}$ possible solutions (50 multiplied by itself 5,000 times)! Nevertheless, GMAC had to make daily decisions for these cars, irrespective of how complex the problem was or the number of possible solutions.
2. *Transportation costs.* When GMAC shipped an entire truckload of cars from one location to another, it would realize a better price per car than when it shipped only one car (or few cars), which lowered the overall logistics cost. This occurred because the cost of transport was primarily tied to individual trucks and drivers, with the number of cars on each truck being of secondary importance.

[17]Although the locations of the 50 auction sites are fixed, GMAC may, from time to time, change the auctions it does business with by dropping some sites and adding new ones (thereby changing the location of the 50 yellow circles). This may happen if cars are routinely damaged at some sites, auction fees go up, or some other reason arises. However, these decisions raise several additional questions, such as: *How do we evaluate the monetary impact of dropping some sites and adding others?* and *Can we increase profits by replacing some auction sites with others?* We will address these important questions later.

[18]Many reports are available for estimating the auction price of cars, including *Black Book*, *Kelley Blue Book*, the *Manheim Market Report*, and others.

3. *Volume effect.* Although GMAC wanted to send each car to the auction site where the highest price could be realized, sending too many cars of same color, make, and mileage to the same auction site would trigger the volume effect. For example, if GMAC sent 45 white Chevrolet Camaros to the same auction site (which might have all been returned from a rental agency on the same day), then these cars were likely to sell for the minimum opening price, because with 45 identical cars for sale, there wouldn't be enough buyers to bid the price up on each car (meaning there was a limit to how much supply could be absorbed by each site). On the other hand, if GMAC sent only five Chevrolet Camaros to the same auction site, then these five cars would fetch a higher price because the same number of buyers would be bidding on a smaller number of cars.
4. *Price depreciation and inventory holding costs*. To further complicate matters, every auction site had a fixed day for selling cars (e.g., every second Friday at 10 am). Because of this, if GMAC shipped 100 cars to an auction site and the cars arrived one or two days *after* the auction day, then these cars would sit until the next sale day, incurring depreciation and holding costs. Because of this, GMAC needed to check the exact auction day and inventory levels across all 50 auction sites before making any new distribution decisions.
5. *Price changes*. Used car prices change over time, and these changes may be slow and subtle (over many years as consumer preferences change), sudden and dramatic (as was the case in March 2020 when the COVID-19 panic set in), or region specific (e.g., convertible cars become unpopular in northern states during the winter months, and consequently, they fetch lower prices, which is part of the "seasonality effect"). GMAC also had to deal with next year's models entering the market during August and September, causing older models to drop sharply in price (also part of the seasonality effect). During this time of year, it was better to ship cars nearby and sell them quickly, rather than shipping them longer distances to more lucrative auction sites. Additionally, new body style models were introduced every few years, causing an even bigger drop in price for older body styles.

Coming up with a daily decision of where to send the returned off-lease cars wasn't easy, as the decision needed to consider all these factors.

Furthermore, the process of transporting a car to a specific auction site could take up to two weeks, as the truck would have to drive to the pick-up location, load the car, pick up some additional cars (possibly somewhere close by), and then finally deliver the cars to the designated auction. Because of this, GMAC had to consider the sale price for each car a couple of weeks ahead of time. For example, for a car located in Jacksonville, Florida, GMAC might consider sending this car to an auction site in Georgia, Pennsylvania, or California. The price prediction for these three auction sites would be different, because GMAC would be predicting the sale price five days into the future for the Georgia auction site, ten days into the future for the Pennsylvania auction site, and fifteen days into the future for the California auction site. The differences in time were due to the transportation distance. However, to predict these prices, GMAC needed to consider the seasonality effect, price depreciation, volume effect, and inventory levels. In making the decision of Georgia vs. Pennsylvania vs. California, GMAC would also need to weigh the possibility of a better price in California against the higher transportation cost, higher depreciation, and higher overall risk.

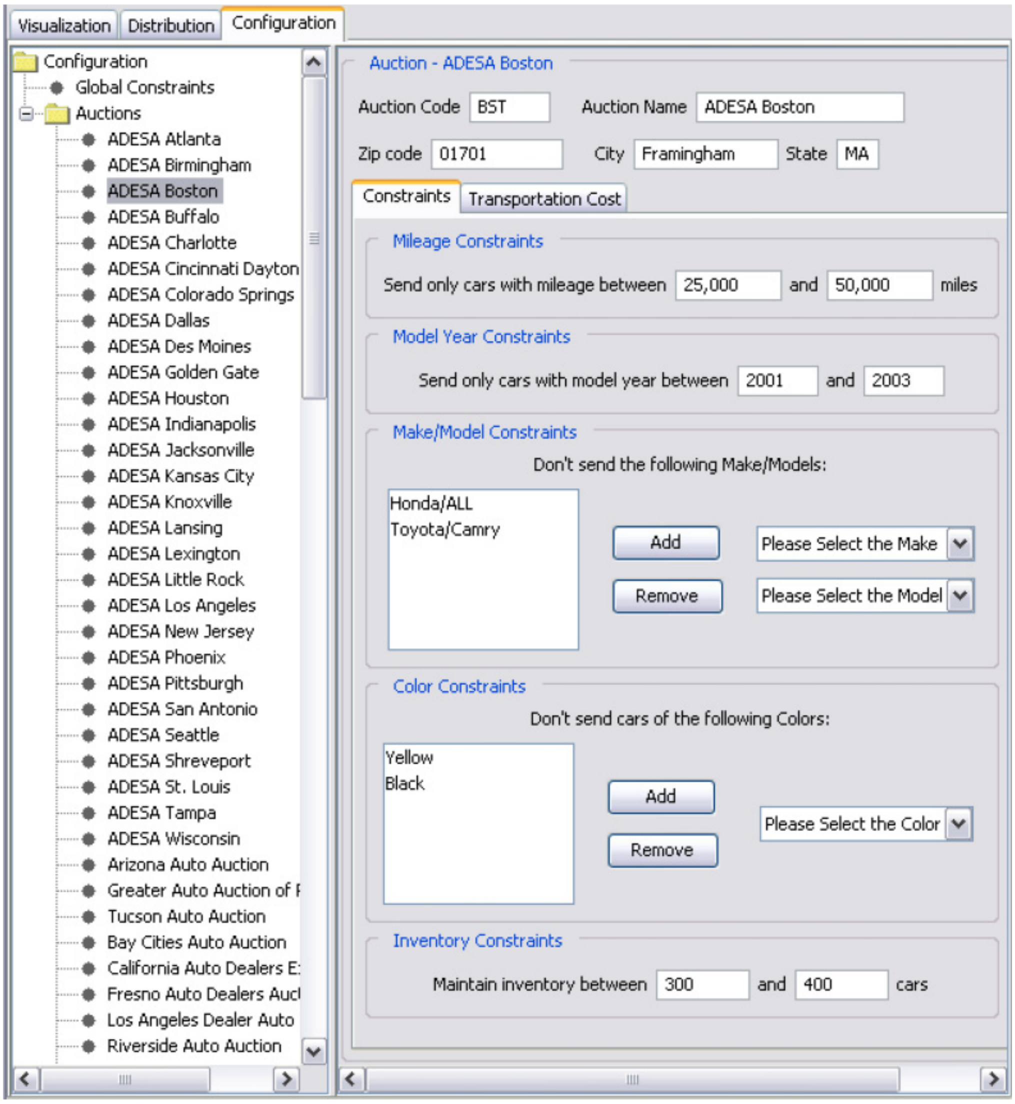
One of the main features of the developed system was a separation of the objective (maximization of the aggregate resale value of all returned off-lease cars) and constraints (in the spirit of my experimental GENOCOP systems). Consequently, the users of the system had the additional option to add, modify, or delete various constraints and business rules. Constraints that were applied to all auction sites were regarded as global constraints (e.g., "maximum transportation distance" which limited the transportation distance of all cars) and GMAC could also implement a large variety of local, auction-specific constraints within the system, such as:

- *Mileage constraints*, which defined the upper and lower mileage of cars that could be shipped to a specific auction site. An example of this constraint would be "only ship cars that have between 30,000 and 70,000 miles to the ADESA Atlanta auction site."
- *Model year constraints,* which specified a range of model years that could be sent to a specific auction site. For example, GMAC could specify that a particular auction site could only accept model years between 2002 and 2004.
- *Make/model exclusion constraints,* which specified certain makes/models that were to be excluded from specific auction sites.
- *Color exclusion constraint*s, which specified certain colors that were to be excluded from specific auction sites.
- *Inventory constraints,* which specified the desired inventory level at each auction site. For example, GMAC could specify an inventory level between 600 and 800 cars for an auction site at any particular time.

Each auction site could have different constraint settings, which represented the business rules that GMAC wanted to operate under for that site. For the ADESA Boston auction site, the constraints represented the following business rules (as shown in Figure 2):

- "Send only cars with 25,000 to 50,000 miles"
- "Send only 2001, 2002, or 2003 model years"
- "Do not send any Honda or Toyota Camry cars"
- "Do not send any yellow or black cars"
- "Keep the inventory between 300 and 400 cars"

Except for the inventory constraint, all these constraints were defined as hard constraints. If the system had to break a hard constraint, it would mark this recommendation with the notation "constraint violation." Inventory constraints, on the other hand, were defined as "soft" constraints and a penalty was assigned to solutions that violated these constraints. The penalty for violating a soft constraint would grow exponentially, and so instances where this constraint was violated in a significant way were rare. However, if the system had to process a very large number of cars on a single day, then the inventory constraint might have been violated at almost every auction site. In such cases, the exponential penalty function would make these violations uniform. For example,



Auction - ADESA Boston

Auction Code BST Auction Name ADESA Boston

Zip code 01701 City Framingham State MA

Mileage Constraints

Send only cars with mileage between 25,000 and 50,000 miles

Model Year Constraints

Send only cars with model year between 2001 and 2003

Make/Model Constraints

Don't send the following Make/Models:

Honda/ALL
Toyota/Camry

Color Constraints

Don't send cars of the following Colors:

Yellow
Black

Inventory Constraints

Maintain inventory between 300 and 400 cars

Figure 2: Screenshot showing local constraints set for the "ADESA Boston" auction site.

in a case where all auction sites have a maximum inventory constraint of 300 cars but the current number of cars to be distributed would increase this inventory level to an average of 400 cars per auction, then the penalty for violating this soft constraint would be evenly distributed across all sites (so that they have the same degree of violation).

Because these constraints allowed GMAC to set various business rules (e.g., "do not send any red cars to Florida"), the configuration screen served as a link between GMAC and the system. GMAC could also use this configuration screen to investigate various "what-if" scenarios, such as "what would be the distribution of cars if the maximum transportation distance was limited to 500 miles?" Because 300 auction sites were configured in the system and only 50 were "active," GMAC could also activate or deactivate any auction site, and then re-run the optimization process to test a specific what-if scenario, such as “what would happen to the aggregate resale value of all cars if we used 60 auction sites instead of 50?”

GMAC could also use different what-if scenarios to investigate different transportation cost options available from different suppliers. The system calculated the transportation cost from any distribution site to any auction for any number of cars, and two factors influenced this cost: (1) the distance between a distribution site and auction, and (2) the number of cars being shipped.

The optimization model generated a variety of possible distribution plans that served as input to the prediction model. This input provided a destination assignment (i.e., auction site) for each off-lease car, which the prediction model then used to generate a predicted sale price. The optimization model then summed all these predicted prices (i.e., the output data) to evaluate the quality of the distribution plan—the higher the sum of the predicted sale prices, the better the distribution plan (hence, there was a strong relationship between the prediction and optimization models). An evolutionary algorithm based on indirect representation was used, where all available auction sites were sorted by *distance* from a particular car. In other words, auction 1 was the closest (distance-wise), auction 2 was the second closest, and so forth. Hence, each solution was represented by a vector of auction site indices (relative to a particular car), and the length of the vector was equal to the number of cars being distributed:

$$\begin{array}{|c|c|c|}\hline 3 & 4 & 4 \\ \hline\end{array} \quad \ldots \quad \begin{array}{|c|c|}\hline 1 & 1 \\ \hline\end{array}$$

The vector above represents a solution where the first car is shipped to the third closest auction (for this particular car), the second car is shipped to the fourth closest auction (for this particular car), the third car is shipped also to the fourth closest auction (note, however, that the second and third car are most likely shipped to different auction sites, as the fourth closest auction for the second and third car need not be the same), and so on, with the last two cars being shipped to the closest auction sites. In this implementation of evolutionary algorithms, the optimization model applied the elitist strategy, which forced the best solution from one generation to the next, as well as various mutation and crossover operators that were discovered through experimentation.

To enable learning within the system, the prediction model updated itself with the arrival of new data. The prediction model contained numerous parameters (different values for various adjustments) that were automatically updated to capture changing trends in the used car market at regular intervals.

When used in a high-volume setting, where thousands of cars were returned off-lease each day, the system generated a significant lift in the aggregate resale value. This was one of the most successful projects at that time—a project where evolutionary algorithms played the key role and where my earlier experience (on separation of objective and constraints) proved to be very useful.

4.2 Adaptive Business Intelligence

The GMAC project also provided me with a very interesting observation: namely, that a significant gap existed in business between having the necessary domain-specific knowledge and being able to use it to make the best decision. Because knowledge is an essential component of any decision-making process (as the old saying goes, *knowledge is power*!), many businesses viewed knowledge as the final objective. But the GMAC project demonstrated that knowledge wasn’t enough. GMAC knew *a lot* about their operation—they had hundreds of graphs and charts that visualized every bit of

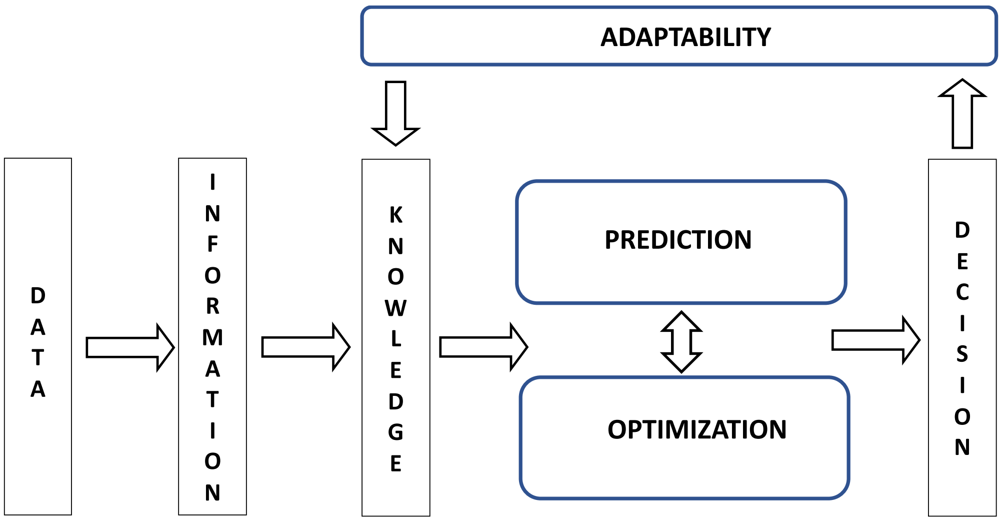


Figure 3: Components of adaptive business intelligence.

data—but the decision-makers weren't able to apply this knowledge effectively to optimize the daily distribution decisions! In short, all the knowledge in the world wouldn't guarantee the right or best decision for GMAC or any other organization.

So, around 2003 or so, I came to the realization that the future of *business intelligence* would be in systems that could recommend optimized decisions (rather than just providing reports, analytics, and insights). This is how a new term, *adaptive business intelligence*, was born. While *business intelligence* was defined as "a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data," the term *adaptive business intelligence* was defined as "the discipline of using prediction and optimization techniques to build self-learning decisioning systems." As illustrated in Figure 3, in addition to performing the role of traditional business intelligence functions (i.e., transforming data into knowledge), adaptive business intelligence also includes these crucial decision-making components: prediction, optimization, and learning (called "adaptability" at that time).

In hindsight, the GMAC project resulted in a new concept for me, and a few years later the idea of adaptive business intelligence was presented in book form (Michalewicz et al., 2007).

Towards the end of 2004, after 16 years in the United States, I moved to Australia (School of Computer Science, University of Adelaide) and a couple of months later (February 2005) I co-founded my second business, SolveIT Software. Again, I will skip the description of the first (painful) years of starting a new business on a new continent, and instead describe another "discovery" made while analyzing some real-world business problems that we encountered during that time.

4.3 Multicomponent Problems

An observation that emerged from one of our commercial projects at SolveIT Software was the following: real-world problems are sometimes comprised of several "components" that interact with each other; organizations realize that these components are related and affect one another, and what is most desirable is that a solution for the *overall* *problem* takes into account all these subcomponents.[19] For example, scheduling production lines (e.g., maximizing the efficiency or minimizing the cost) is directly related to inventory costs, labor costs, and service levels to customers, among other business metrics, and shouldn't be considered in isolation. Moreover, optimizing one component may negatively impact another. For these reasons, organizations could unlock more value through "globally optimized" solutions that consider all the components together, simultaneously, rather than just a single component at a time.

One of the projects that displayed this multicomponent characteristic was related to optimizing the transportation of water tanks. In this case, a manufacturer produced water tanks of different shapes and sizes based on customer orders. The total number of customer orders per month was approximately 10,000, which varied in delivery locations. Each customer ordered a water tank with specific characteristics (including size) and expected to receive it within a period of time (usually one month). These water tanks were then transported by a fleet of trucks (operated by the water tank company) that had different characteristics and some were equipped with trailers. A subset of orders was selected and assigned to a truck and the deliveries were then scheduled. Because the tanks were empty and of different sizes, they could be bundled inside each other to maximize the truck's load. A bundled tank had to be unbundled at special sites, called "bases," before the tank's delivery to the final destination. There might be several bases close to the various customer locations where the tanks were going to be delivered, and selecting different bases affected the best overall solution. When the tanks were unbundled at a base, only some of them fit back onto the truck, as unbundled tanks required more space. So, the truck was loaded with a subset of these unbundled tanks and delivered them to the customer, while the remaining tanks were kept in the base until the truck returned to continue the delivery process.

The goal of the optimization algorithm was to group the customer orders into subsets of tanks that were bundled and loaded onto trucks for delivery (possibly with trailers), and then determine the best base for unbundling these tanks before delivery to each customer, so that the overall delivery cost was minimized. Each of the mentioned procedures in this problem (tank subset selection, base selection, delivery routing, bundling and unbundling) was just one component of the problem and finding a solution for each component in isolation didn't lead to the optimal solution for the overall problem.

For example, if we just focused on the subselection of tanks (based on customer orders), there is no guarantee that there exists a feasible bundling solution that would allow this subset to fit onto a truck. Also, by selecting tanks without considering the location of customer destinations and location of bases, the best solution might not be a high-quality one, as there might be a customer destination that required a low-cost tank, but that location was far from any base, making delivery very expensive. On the other hand, it is impossible to select the best route for customer destinations before selecting the tanks, because without first selecting the tanks, the best solution (lowest possible cost and distance) was to deliver nothing. Thus, solving each component of the problem in isolation didn't lead to an optimized overall solution, especially when we added additional considerations, such as the rostering of drivers (who often had

[19]There are similar concepts to multicomponent problems in other disciplines, such as operations research and management science, with different names such as integrated systems, integrated supply chains, system planning, and hierarchical production planning.

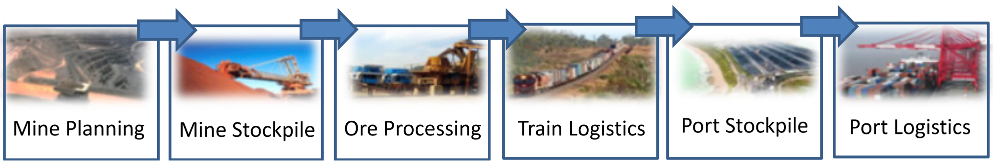


Figure 4: Steps in mine-to-port supply chain.

different qualifications), fatigue factors and labor laws, traffic patterns on the roads, usability of different trucks for different segments of roads, maintenance schedules, and so on, which further complicated the problem as well as the interaction between its various components.

Another project that displayed this multicomponent characteristic was related to optimizing the mine-to-port supply chain. In such operations, a mining company tries to satisfy customer orders by providing a predefined amount of product (e.g., coal, iron ore, etc.), graded to a specific quality (such as the percentage of iron within the ore) on a particular due date when the product must be ready for ship loading. The supply chain would have multiple stages, starting with planning the extraction of ore at the mine, all the way through to planning the rail corridor and when each ship will be berthed and loaded (see Figure 4). Each of these steps is significantly complex by itself, such as coming up with an optimized ore extraction sequence, or the most efficient train schedule (which must consider maintenance, the availability of various drivers, the number of junctions in the network where trains can pass, etc.), representing hard optimization problems to solve even on their own.

Apart from the complexity within each component, optimizing a single component in isolation will not lead us to a globally optimized solution[20] for the whole supply chain. As an example, scheduling trains to optimality (moving as much product as possible from the mine to port) might result in too much product at the port or even the wrong product at the wrong time, because we're only worried about maximizing the asset utilization of the train network, rather than what's optimal for the mine or port. Solving each component individually and then assembling the solutions together is unlikely to result in an overall, globally optimum solution; and if all solutions are local, then many business opportunities may be lost. Hence, solving an individual component of the mining supply chain provides us with a local optimum as far as the whole problem is concerned, rather than the global optimum, especially given there are strong dependencies between all components of the supply chain (e.g., some decisions on which blocks to mine directly impact crushing and blending activities, which in turn impact train logistics).

[20]There is some confusion related to terminology used in academia and industry with respect to "local optimum" (or local solution) versus "global optimum" (or global solution). In academia, the highest peak in a landscape is called global optimum and possibly many lower peaks, which are the local optima. So this concept of local vs. global optima in academia is strictly "vertical," where the quality measure score determines the height of a peak, and by having all the measurements we can easily identify all local peaks. In business, however, the term "global optimum" has a different meaning. Note that when we're dealing with multicomponent problems, we can get a locally optimum solution for each individual component by solving each component separately, and then try to put the components together. Still, the result is likely to be suboptimal.

A global optimization algorithm, on the other hand, would simultaneously address the objectives of each component of the supply chain. However, the implementation of such algorithms for global optimization is scientifically challenging, and there are two general approaches: (i) a centralized approach, where a global "agent" is responsible for "supervising" local agents (that are dealing with the individual components of the overall operation) and tries to synchronize their activities and recommended decisions, and (ii) an approach based on distributed optimization, where local optimization algorithms "talk" to each other in order to synchronize their actions.

With this in mind, I suggested a new and possibly interesting research direction for the evolutionary computation community at that time, and with two co-authors put together a paper (Bonyadi et al., 2013) on a travelling thief problem (this was a new term at that time; I did not find this term in existing literature). The idea followed the structure of problems that I encountered in business and which I described earlier: take two well-known components and merge them together in such a way that these two components, when solved separately, wouldn't yield the optimal solution to the overall problem. So the travelling thief problem was a combination of two well-known optimization problems: the travelling salesman problem and the knapsack problem. So, there are n cities, and the cost matrix for moving from one city to another is given. Also, there are m items; each of them has some value and a weight. There is a thief who is going to visit these cities exactly once and pick some items from the cities and fill his knapsack (the maximum weight for the knapsack is given). The aim is to find a tour that visits all the cities exactly once and gets back to the starting city, optimizing objective function(s) while the total weight of the knapsack is not violated. Note that the objective function(s) might be related to the time of the travel and/or the total theft value from picking the items. However, the values in the cost matrix for moving from one city to another depend on the current weight of the knapsack, that is, the heavier the knapsack, the larger the cost. The travelling thief problem became popular in the evolutionary computation community. Apart from several research papers published at various journals, there have been competitions on the travelling thief problem organized at various EC conferences (including this year GECCO '23). I couldn't resist either, and a few years after leaving academia I was still involved in some research on multicomponent problems (Przybyłek et al., 2018).

4.4 Puzzle-Based Learning

I was also involved in another activity during this period. I found some similarities between "solving" real-world problems and solving puzzles, and I realized that in academia we teach a variety of subjects, except one: how to think! So, in 2008, I wrote a textbook on problem-solving (Michalewicz and Michalewicz, 2008) and amazingly, hundreds of universities all over the world were introducing puzzle-based learning courses in their curriculums. Such courses are still taught today (judging from emails I am getting on a regular basis), and five years later, an editor at Springer convinced me to write a sequel to the original textbook with emphasis on *how to teach* such courses (and how to evaluate the performance of a student). This is indeed far from trivial; imagine an exam during which a student gets a puzzle to solve and either solves it or not! So, with the assistance of a few co-authors (who were already involved in teaching puzzle-based learning courses at their universities), a new text (Meyer et al., 2014) was published, where we shared our experiences. Some additional information (including videos and software illustrations of selected puzzles) is available at http://www.puzzlebasedlearning.edu.au/ and lecture materials are available on request.

4.5 Departure from Academia

However, around 2011–2012, I realized that there is a significant gap between theory and practice in evolutionary algorithms. For example, in 2011, with Frank Neumann and Andreas Ernst, we placed a call for papers for the special issue on "Heuristic Search Methods for Large Scale Optimization Problems in Industry" for the *Evolutionary Computation Journal*. In the call for papers, we inserted the following phrase: "*This special issue solicits novel high-quality contributions on heuristic methods for large, applied optimization problems that have been used in practice*." You can probably guess how many papers we received . . . and so the idea of this special issue was abandoned.

Many other changes were also happening in 2012. I found myself enjoying various business applications and challenges more and more and discovered that being a "business owner" gave me greater pleasure than being a professor. Furthermore, I read an essay (Ullman, 2009) where Jeff Ullman (my hero from 40 years ago—I used his excellent textbooks on data structures, algorithms, and databases in the pre-evolutionary period of my life) placed some of his thoughts on advising students. He wrote:

> Look at the last section [of some paper], where there were always some "open problems." Pick one, and work on it, until you are able to make a little progress. Then write a paper of your own about your progress, and don't forget to include an "open problems" section, where you put in everything you were unable to do. Unfortunately this approach, still widely practiced today, encourages mediocrity. It gives the illusion that research is about making small increments to someone else's work. But worse, it almost guarantees that after a while, the work is driven by what *can* be solved, rather than what *needs* to be solved.

and

> People write papers, and the papers get accepted because they are reviewed by the people who wrote the papers being improved incrementally, but the influence beyond the world of paper-writing is minimal.

I couldn't agree more. That year I wrote my two (and only) essays (Michalewicz, 2012a, 2012b) that relate to the last phrase of Ullman's essay, "*but the influence beyond the world of paper-writing is minimal,*" where I tried to share my experiences (theory vs. practice) with the evolutionary computation community.

In January 2013, I left academia for good. A few months earlier SolveIT Software[21] was acquired by Schneider Electric and I did not plan to stay there much longer after the acquisition. But I had also lost my taste for academic research. So, I resigned from my university position, and then resigned from all editorial boards and program committees. Since that time, I have not reviewed any papers (I keep reading some, but no reviews, thank you!)—and life without reviewing is pretty good.

5 Back to Business (2013–2023)

It was difficult for me to leave academia for good. Still, I was involved in some paper-writing activities with my former students or students who were in the process of completing their degrees (I could not leave them halfway through their programs), so I contributed to their research efforts and assisted them in putting together some papers

[21]SolveIT Software became the third-fastest-growing company in Australia in 2012, as ranked by Deloitte; the company won numerous awards and counted among its customers some of the largest corporations in the world, including Rio Tinto, BHP Billiton, and Xstrata. SolveIT Software won all major mining tenders competing against the largest companies of the world.

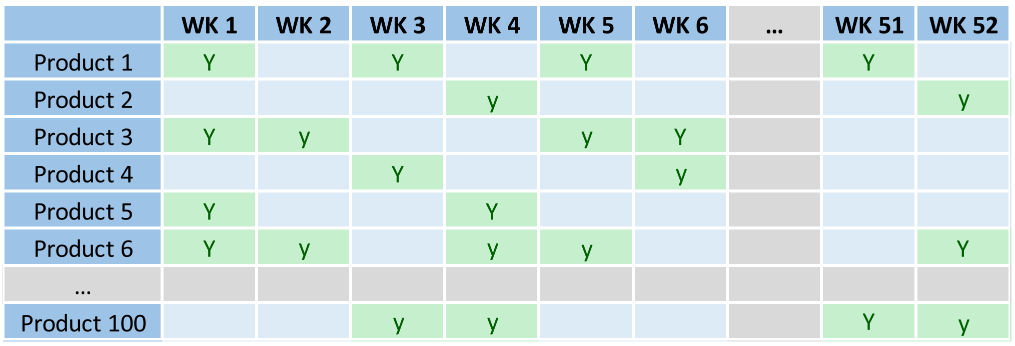
on particle swarm optimization (Bonyadi and Michalewicz, 2014a; 2014b; Bonyadi, Li et al., 2014; Bonyadi and Michalewicz, 2016a, 2016b, 2017a, 2017b), wheat blending (Li et al., 2014), car racing (Bonyadi et al., 2016), finance (Ghandar et al., 2016), and scheduling (Abello and Michalewicz, 2014a, 2014b, 2014c; Zhang et al., 2016; Weise et al., 2014), but my heart was already somewhere else.

On July 30, 2014, I co-founded my third (and I believe, the last) company, Complexica. Again, many interesting problems poured in, but I found out that the same principles would apply: truly intelligent systems that could make meaningful recommendations required a predictive module, optimization module, and learning component. Besides, great care needed to be taken while incorporating constraints and business rules. Probably it would be the best to illustrate all these important issues by discussing one application, where the necessity of incorporating problem-specific knowledge is also very apparent. The application is for *promotional planning and pricing,*[22] which has all the hallmarks of a super complex business problem worthy of discussion.

We've all experienced product promotions (e.g., *Sale! 50% off! Buy one, get one free!*), which manufacturers and retailers use to drive foot traffic into stores, increase volume and market share, and build awareness for new products (have you ever bought a product on promotion, liked it, and then switched to that product permanently?). These promotional activities are typically funded by both the retailer and participating manufacturer and can account for more than 50% of the revenue of such companies. Hence, promotions are a big deal. And not only is the problem of promotional planning inherently complex—where it's difficult to make "good" decisions that generate real improvements in revenue, margin, and overall profitability—but it's also an inherently *high-value problem,* where the difference between good and bad decisions can mean millions of dollars and many percentage points of market share.

Now, let's consider the structural elements of the problem. There's a certain length of time for which we must plan our promotions, and let's assume that we plan for the entire year, so our planning horizon is 52 weeks. Let's also assume that the promotional period is one week, which provides us with the plan's granularity and the level of detail we must plan to. We can represent this granularity with a promotional calendar, called a *slotting board,* where each column is a week, and each row is a particular product, allowing for promotions to be *slotted* into each column/row combination. For example, the slotting board in Figure 5 is just for one product category (e.g., wine, while other categories may include spirits, beer, etc.) in just one area of operations (e.g., a state). Note also that the structure of the matrix in Figure 5 is not flat; usually products are organized into subcategories (e.g., category of "wine" is split into subcategories of "white wine," "red wine," sparkling," etc.). And if there are only 100 individual products for us to plan in a category, then the simple decision of whether to promote a particular product for any given week requires 5,200 individual "yes" or "no" decisions (52 weeks x 100 products). If we ignore other elements of the problem (such the promotional price, promotion type, ancillary marketing, holidays and seasonality, catalog placement, and so on) the number of individual binary yes/no decisions still implies an astronomical number of possible plans (1 followed by 1,565 zeros!). Just for one category of products in just one area.

[22]There are several related terms to the area of promotional planning, including *promotional programming, trade promotions, trade promotion optimization (TPO),* and so on. For the sake of simplicity, however, we'll use "promotional planning" throughout this paper even though at times there may be a more fitting term.



	WK 1	WK 2	WK 3	WK 4	WK 5	WK 6	...	WK 51	WK 52
Product 1	Y		Y		Y			Y	
Product 2				y					y
Product 3	Y	y			y	Y			
Product 4			Y			y			
Product 5	Y			Y					
Product 6	Y	y		y	y				Y
...									
Product 100			y	y				Y	y

Figure 5: Slotting board for one product category in one area of operation.

The promotional plan must also adhere to business rules for individual products. For example, business rules may prevent the promotion of specific products for less than four weeks or more than twelve weeks during any twelve-month period, known as *minimum and maximum frequency*. Furthermore, these business rules may apply to the "gap" in between promotions for the same product, known as the *minimum and maximum promotional gap*, so that promotions don't happen too often or too infrequently (such as promoting the same product for nine consecutive weeks and then not promoting it for the rest of the year). Such business rules also apply to prices, where the minimum promotional price might be set at no less than 50% of the shelf price (i.e., the non-promoted price) and not more than 90% and must move by some price step increment (e.g., 50 cents or one dollar, to avoid awkward prices like $8.13). We need to also bear in mind that different geographic regions may have their own business rules, which adds further complexity to the problem. There may also be specific rules tied to the promotional period itself, such as the minimum and maximum number of products on promotion at any given time (which may differ from week to week as we consider holidays or other events). For example, during most weeks of the year, it may be permissible to have 30% of our products on promotion at the same time, but for certain weeks of the year, such as before major holidays like Easter, Christmas, and New Year's, it may be appropriate to increase this percentage.

These business rules may be quite complex, and yet, we've only considered products in isolation, and not thought about products being constrained by the promotional activity of other products. Hence, we might need to extend our business rules to cover different pack sizes of the same base product, different varieties or flavours of the same product (e.g., where all go on promotion or none at all), or different products altogether (e.g., where we can promote a subset of our products in a category at any given time, but not all of them within the same promotional period). And lastly, the promotional plan, as a whole, must meet certain KPI thresholds, such as volume or volume growth, revenue or revenue growth, and gross profit (among others), which we must also define as business rules. When we consider all these additional factors, it seems that the job is quite challenging! Just creating a single feasible plan—one that doesn't violate any business rules or constraints—is already a Herculean task, most likely involving a multitude of spreadsheets and endless hours of evaluating new plans against historical promotions to "guess" the likely outcome.

One of the challenges is constructing an evaluation function for the optimizer. Clearly, we are talking about a predictive model that would estimate the outcome of any promotional plan. Using historical data, we can investigate many factors that affect the outcome, most important of which is *price elasticity* (or simply *elasticity*). Elasticity has its roots in Economic Theory and is part of the Law of Demand, which states that demand for any given product will go up as its price goes down, and vice versa (with a few exceptions, such as some luxury goods and other limited circumstances). All products have an elasticity curve; some of these curves are steep, where a small change in price causes a large change in demand, and some shallow, where a large change in price causes a small change in demand.

We can also increase our knowledge by trying to understand *why* a reduction in price caused an increase in sales. For some products, a change in price may have affected a consumer's decision on whether to buy the product at all, so the promotion resulted in a real change in consumption. However, some products (such as ice cream and champagne) may experience a real increase in consumption when placed on promotion, because people not only buy more, but they also consume more as well. Some products are highly seasonal, and we may find seasonality to be the dominant factor driving sales (with our analysis possibly revealing that any increase in sales wasn't attributable to the promotion). Furthermore, we may discover long-term trends for particular products, brands, or even entire categories (as some categories might be experiencing growth while others are in a state of decline), providing us with even more knowledge of why demand increased or decreased at certain points in time. And lastly, when a product is on promotion, we can expect consumers to stock up on this product, therefore "bringing forward" future purchases. This is called the *pull-forward effect*, which influences non-perishable products (those that consumers can easily store). The pull-forward effect results in a fall in demand to below baseline levels after the promotion has ended and is something we need to understand and consider when planning future promotions.

When we're planning future promotions, one of our goals is to maximize the "gain" from competitor products (so that consumers switch from a competing product to our own) and minimize the "loss" from our own product range. If an increase in promotional sales comes at the cost of another one of our products, this is called *cannibalization,* which means that consumers have switched from one of our products they regularly buy to the one on promotion. Through data analysis, we can create a *cannibalization matrix*, which is a table that outlines the expected cannibalization effect of certain products when placed on promotion. However, the practical challenges of constructing such a table are significant. Without applying any domain knowledge or business rules, it might be necessary to look up each individual product and calculate its cannibalizing effect on every other product in our range. For most manufacturers, which may sell hundreds or even thousands of products, this would result in a table with tens or even *hundreds of thousands of values.* While it's possible to calculate this automatically, there's no easy way to validate these values without going through them line by line. And finally, an increase in sales could be due to external factors; for example, sporting events can increase demand for beer and snacks, while hot weather is positively correlated with higher ice cream sales. Not only is this type of knowledge important for decision making, but it also forms the basis of our predictive modelling efforts.

Prediction models use past data to make forward-looking predictions, in effect answering the question: *Based upon what we know about the past, what's likely to happen in the future?* A predictive model to "evaluate" a plan by predicting an outcome, such as *Volume* (the total unit quantity of products sold in whatever measure is used), *Net revenue* (the total revenue generated, based on the promotional sales price and volume), *Retailer gross profit* (the total retailer gross profit, typically calculated as the difference between the promotional price and the retailer's cost of the product plus promotional funding, multiplied by the total units sold), or *Manufacturer gross profit* (the gross profit, typically calculated as the difference between the net revenue and total product cost, including all production and freight costs, as well as any promotional co-funding costs).

Once a prediction model is developed for evaluating our promotional plans, we then need to create several plans and find the "best" one through a process of optimization. First, however, we need to define what "best" means to us, which in this case might be "any promotional plan that maximizes overall volume while satisfying our business rules and constraints." Note that some of these business rules and constraints might be for the entire category, while others are just for individual products. In one category, as an example, we may have the following business rules and constraints:

- No fewer than 30 products and no more than 60 products on promotion in any given promotional period (soft)
- The overall minimum net revenue should be $1,000,000 (hard)
- The overall gross profit growth over last year should be 3% (hard)
- The overall minimum retailer margin growth over last year should be 2% (hard)

whereas for Product 43, we may have some additional business rules and constraints:

- Minimum promotional price of $4.00 and maximum price of $7.00 (soft)
- Price or discount step: $0.25 (soft)
- Minimum of five and maximum of eight promotional frequencies (hard)
- Maximum of three consecutive promotional periods (hard)
- Maximum of five consecutive non-promotional periods (hard).

Such business rules and constraints typically reside in the minds of human experts within each organization, and it's often a significant undertaking to extract and document them; however, such a process is highly beneficial, because it reduces key man risk, provides visibility of the rules and constraints under which decisions are made, and allows for "testing" of each rule and constraint to ensure ongoing relevance.

There are other important considerations related to optimization. Usually, we start the search for the best plan from some starting position, for example, last year's plan or a new promotional plan that we manually create. Sometimes it's desirable to restrict the type of changes made to the original plan (e.g., whether the optimization algorithm can switch between two products, removing one from promotion and adding another) or restrict the number of changes, perhaps due to retailer requirements, or to assist in software adoption and implementation, as people can become disheartened if the slotting board they've worked on for three days comes back with 150 changes! This is also important from the perspective of *explainability* of the optimizer (we refer to this issue in the following section).

Altogether, the problem of promotional planning and pricing is super difficult.[23] The search space is enormous, the structure of a solution (with individual products, their categories and subcategories, and many different geographical areas) is very rich (a set of multi-layered matrices), there are tens or hundreds of business rules and constraints of different weights and complexities, that apply to all levels of the solution (varying from individual products through categories and subcategories, to various geographical areas), a few conflicting objectives (e.g., volume vs. margin), and a very complex predictive model that serves as an evaluation function for the optimizer. Think about the complexity of the data structure that captures the structure of the problem; think about tens of operators that introduce changes to promotional plan considering all business rules and constraints; think about possible constraint-handling methods. It is a lot of fun to deal with problems of such caliber! And what an opportunity for evolutionary algorithms!

And yet, the promotional planning and pricing problem is like the transportation problem that I studied at the beginning of my career, in a sense that many issues that I was dealing with at that time are also present; the difference is mainly in scale. It's because the main theme remained the same: how to incorporate problem-specific knowledge into a genetic algorithm in terms of data structures, operators, and constraint-handling techniques to create a successful implementation? This was the case for the transportation problem, and it's also the case for the promotional planning and pricing problem, as well as most real-world problems that I have been dealing with.

Note also that there is an additional challenge in developing such system for any given organization: the finished system should be easily adjustable to different organizations that face of promotional planning and pricing decisions. Different organizations may have promotional plans of different structures (e.g., sub-subcategories on the top of categories and subcategories), possibly slightly different objectives, and definitely different sets of business rules and constraints. So again, as it was done within the GENOCOP system with constraints, it's important to provide end users with flexibility for activating/deactivating/modifying their set of business rules and constraints without changing the optimizer itself.

6 Conclusions

Let me summarize my last 20+ years of business experience by making a few (more or less) general observations in no particular order of importance. By the way, these observations relate to discrete problems; I have never experienced a continuous type of problem within business environments. Don't get me wrong—most of my academic "training" was done on continuous domains; all constraint-handling approaches were experimented with on continuous domains. Further, there are many important real-world problems—for example, engineering design problems—that require numerical optimization; the only thing I am saying is that I didn't experience these types of problems during my business life.

My observations are limited to evolutionary algorithms only (as opposed to the whole software systems) because the topic of how to implement a *commercially successful* application is very broad and complex (and outside the scope of this paper). I would like to share just one thought on this topic, and the following well-known story illustrates this point. A scientist invented a new type of food for dogs, and he proved that dogs

[23] A detailed discussion on this problem (together with descriptions of other real-world problems) is included in Michalewicz et al. (2021).

would benefit from this food by never getting sick, having shinier fur, stronger teeth, etc. The list of benefits was long and compelling. However, there was one serious problem with his dog food invention: no dog wanted to eat it. And the same applies to software systems. The software may include powerful science components, super-precise predictive models, first-class multiobjective optimizers, etc.—and yet, people within the business may not wish to use it. We can look at this from another perspective, which might be summarized by the so-called Anna Karenina principle[24]: "All happy families are alike; each unhappy family is unhappy in its own way" (Tolstoy, 1877).

In other words, happy families share a common set of attributes that lead to happiness, while any variety of attributes can cause unhappiness. The same concept applies to software projects. All commercially successful projects are alike: they have users that see value in the software and actively use it. However, software projects can be unsuccessful for many reasons (slow response time, insufficient business benefit, too complex user interface, inadequate training, lack of change management, etc.) with one outcome: no one in the organization uses the software!

So, let's turn our attention to evolutionary algorithms. The lessons I've learned over the preceding years are the following:

1. To implement a commercially successful evolutionary algorithm, the data structure used to represent a candidate solution is key. This observation is consistent with my first paper published in the *Evolutionary Computation Journal;* nothing has really changed since then. But the richness of the data structures used often implies enormous size of the search space; often the number of possible solutions is beyond astronomical.

2. In most systems, the task of the optimizer (e.g., evolutionary algorithm) is not really to find the optimal solution, but rather to improve on a given solution, which very often is generated in a manual way by a team of experts. Clearly, the degree of improvement is one of the key measures in the success of the project. Further, the running time of the algorithm is also essential; even for longer-term decisions (like a yearly promotional plan) end users often want to see an "improved plan" within minutes!

3. A complex data structure also requires specialized operators that transform current solutions into new solutions (offspring). The design of such operators is of fundamental importance. Note that the search space is usually enormous and heavily constrained (recall the last example of promotional planning and pricing). There is no time for experimenting much with infeasible solutions; all changes to the original solution should be meaningful. Most of the problem-specific knowledge is incorporated in these operators and the number of possible operators might be quite substantial. For example, referring to the last example of promotional planning and pricing, some operators may work on the subcategory level (e.g., swapping promoted products within a subcategory), some may adjust the length of a particular promotion, while others may delete or insert a new promotion. These operators must have some "understanding" of the business rules and constraints and suggest meaningful changes to the current plan.

4. Further, we should separate the constraints and objectives in such a way that authorized users are able to activate/deactivate business rules and constraints (in

[24]The name of the principle derives from Leo Tolstoy's 1877 novel, *Anna Karenina*.

addition to selecting the objective), effectively providing the organization with what-if scenario analysis. Note that activation/deactivation of business rules and constraints may relate to the activation/deactivation of some families of operators.

5. Many researchers have studied evolutionary algorithms within dynamic settings (i.e., time-changing environments) and many methods have been considered (e.g., niching, sharing, introduction of a memory) to enhance diversity of the population of individual solutions, as diversity of the population was perceived as a hedge against a dynamic (or noisy) environment. Researchers have been considering changes in the evaluation function or some constraints (as the result of some changes in real environment) and ways to address them. However, it seems to me that the most important component of this line of research was somehow overlooked: in most optimization projects (e.g., car distribution system, promotional planning and pricing) a predictive model serves as the evaluation function: it would assess the merit of candidate solutions. There are a few consequences of this approach. First, noise is inherent, as any predictive model carries (sometimes significant) error, so the optimizer should be robust enough to handle variability. Second, new data arrive at regular intervals (e.g., sometimes daily, like new sales results from the previous day), and these new data sets may require (automatic) modification of the prediction model, hence a change in the environment. Third, with some frequency, the predictive model compares predicted values vs. actual values, and again, this may call for an update of the predictive model. It seems that the relationship between evolutionary algorithms and predictive models (as their evaluation function), with all new data arriving at regular intervals, with all comparisons and measures of error, and updates of the predictive model, present an interesting research direction for cases of dynamic environments. On top of this, it's often necessary to develop two separate predictive models: an approximate one (that evaluates a candidate solution quite fast) and another that's precise for the final evaluation. This is because very often there are severe limitations on the running time of the optimizer (users are not willing to wait hours to see results), and the precise model used as an evaluation function will not do.

6. An additional aspect of research that's often overlooked is the explanatory feature of an optimizer. On the one hand, such features may seem unnecessary, as it may seem that there's very little to "explain"! After all, the optimizer returns a feasible solution with the best objective score; what else is there to explain? Actually, a lot! Quite often, end users in most organizations would not just "accept" the best recommendation of any optimizer. They need to understand why such a recommendation was generated. So often it's necessary to either develop an "explanatory box" where a few points are displayed providing some reasons for the final recommendation (usually in connection with business rules, constraints, and the objective values of "similar" solutions) or to provide a mechanism that can be used to "investigate" the recommendation (e.g., in promotional planning, the ability to make a very small number of changes to the original plan). Note again that this point is important for practical reasons: as discussed earlier, the main goal is to get the software accepted and used in the organization.

7. The final observations are the following: As discussed earlier, real-world problems are usually comprised of some "components" that interact with each other,

and algorithms should search for a solution to the *overall problem* that considers these components. Further, some problems require optimizers that can handle different time horizons; for example, schedulers should be able to optimize for a day as well as for the week, and the synchronization between such optimizers is far from trivial. Finally, I found that only a limited number of organizations require true multiobjective optimization algorithms; multiobjective aspects of their problems are usually handled (I believe, for simplicity reasons) by selecting just one objective (out of a few) and by defining goals for the remaining ones.

Point #3 from the above list is probably the most important, and also consistent with the observations I described in my paper published in the *Evolutionary Computation Journal* 30 years ago; recall two sentences from the last paragraph of Section 2 of this paper:

The paper summarized my earlier experiments on transportation problems and expanded on my simple intuition that problem-specific knowledge (in terms of data structures used, operators, and constraint-handling techniques) enhances an algorithm's performance. The "more" problem-specific knowledge is incorporated into the algorithm, the "better" the results.

To conclude, I would like to say that over the past 35 years I was simply *lucky*:[25] the first experiment on the transportation problem from 1988 set a particular trajectory for my research interests and instead of spending too much time on other important topics (e.g., convergence of evolutionary algorithms, population structures, selection mechanisms, self-adaptation, hardware realizations) I just concentrated on the issue of incorporating problem-specific knowledge into algorithms (in general) and on constraint-handling techniques (in particular).

This observation is consistent with one paragraph that I found in an excellent book written by the founder of Nike (Knight, 2016):

Luck plays a big role. Yes, I'd like to publicly acknowledge the power of luck. Athletes get lucky, poets get lucky, businesses get lucky. Hard work is critical, a good team is essential, brains and determination are invaluable, but luck may decide the outcome.

Acknowledgments

I would like to thank Emma Hart for her invitation to write this paper for the 30th anniversary issue of the *Evolutionary Computation Journal*; it was a great pleasure to contribute—so many memories came back! Also, this is an excellent opportunity to thank all my 200+ co-authors and co-editors, as well as other scientists from NuTech Solutions, SolveIT Software, and Complexica, who, at different stages, worked with me exploring various research ideas over this 35-year journey and/or developed evolutionary engines for challenging business applications. I am grateful to the IEEE Computational Intelligence Society for the 2019 Evolutionary Computation Pioneer Award,

[25]One of my friends (Raja Sooriamurthi) made me aware of a popular anecdote (often attributed to either Niels Bohr, Albert Einstein, or Enrico Fermi) about a journalist who visited the home of a prominent physicist. The visitor was surprised to find a horseshoe above the front doorway of the scientist's abode (tradition asserts that a horseshoe acts as a talisman of luck when placed over a door). The visitor asked the physicist about the purpose of the horseshoe while expressing incredulity that a man of science could possibly be swayed by a simple-minded folk belief. The physicist replied: "Of course, I don't believe in it, but I understand it brings you luck, whether you believe in it or not!"

which recognized my contributions to some concepts and sustained developments in the field of evolutionary computation. Further, I would like to thank Leonardo Arantes, Łukasz Brocki, Andres Colombari, Krzysztof Krawiec, Henri Luchian, Juan Martin Sanguinetti, Raja Sooriamurthi, and Markus Wagner for their great comments and suggestions on the first draft of this paper. Finally, my special thanks go to Matt, my son, who not only co-authored four books with me and reviewed the earlier draft of this paper, but above all he assisted me with my transition from academia to industry!

References

Abello, M., and Michalewicz, Z. (2014a). Multi-objective resource-constrained project scheduling with time-varying number of tasks. *The Scientific World Journal*, special issue on Recent Advances in Information Technology, Article ID 420101.

Abello, M., and Michalewicz, Z. (2014b). Implicit memory-based technique in solving dynamic scheduling problems through response surface methodology—Part I. *International Journal of Intelligent Computing and Cybernetics*, 7(2):114–142. 10.1108/IJICC-12-2013-0053

Abello, M., and Michalewicz, Z. (2014c). Implicit memory-based technique in solving dynamic scheduling problems through response surface methodology—Part II. *International Journal of Intelligent Computing and Cybernetics*, 7(2):143–174. 10.1108/IJICC-12-2013-0054

Bäck, T., Fogel, D., and Michalewicz, Z. (Eds.) (1997). *Handbook of evolutionary computation*. Oxford University Press and Institute of Physics.

Bonyadi, M., Michalewicz, Z., and Li, X. (2014a). An analysis of the velocity updating rule of particle swarm optimization algorithm. *Journal of Heuristics*, 20(4):417–452. 10.1007/s10732-014-9245-2

Bonyadi, M., and Michalewicz, Z. (2014b). A locally convergent rotational invariant particle swarm optimizer. *Swarm Intelligence*, 8(3):159–198. 10.1007/s11721-014-0095-1

Bonyadi, M., and Michalewicz, Z. (2016a). Analysis of stability, local convergence, and transformation sensitivity of a variant of particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 20(3):370–385. 10.1109/TEVC.2015.2460753

Bonyadi, M., and Michalewicz, Z. (2016b). Stability analysis of the particle swarm optimization without stagnation assumption. *IEEE Transactions on Evolutionary Computation*, 20(5):814–819. 10.1109/TEVC.2015.2508101

Bonyadi, M., and Michalewicz, Z. (2017a). Impacts of coefficients on movement patterns in the particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 21(3):378–390.

Bonyadi, M., and Michalewicz, Z. (2017b). Particle swarm optimization for single objective continuous space problems: A review. *Evolutionary Computation*, 25(1):1–54. 10.1162/EVCO_r_00180, PubMed: 26953883

Bonyadi, M., Michalewicz, Z., and Barone, L. (2013). Travelling thief problem: The first step in transition from theoretical problems to realistic problems. *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*. 10.1109/CEC.2013.6557681

Bonyadi, M., Michalewicz, Z., Nallaperuma, S., and Neumann, F. (2016). Ahura: A heuristic-based racer for the open racing car simulator. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3):290–304. 10.1109/TCIAIG.2016.2565661

Bonyadi, M., Li, X., and Michalewicz, Z. (2014). A hybrid particle swarm with a time-adaptive topology for constrained optimization. *Swarm and Evolutionary Computation*, 18(1):22–37. 10.1016/j.swevo.2014.06.001

Bonyadi, M., Wagner, M., and Michalewicz, Z. (2014). Beyond the edge of feasibility: Analysis of bottlenecks. *Proceedings of the 10th International Conference on Simulated Evolution and Learning*.

Dasgupta, D., and Michalewicz, Z. (Eds.). (1997). *Evolutionary algorithms in engineering applications*. Springer.

Davis, L. (Ed.). (1987). *Genetic algorithms and simulated annealing*. Pitman.

De Jong, K. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Technical Report UMR0635. College of Engineering, University of Michigan.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141. 10.1109/4235.771166

Ghandar, A., Michalewicz, Z., Schmidt, M., To, T.-D., and Zurbruegg, R. (2009). Computational intelligence for evolving trading rules. *IEEE Transactions on Evolutionary Computation*, 13(1):71–86. 10.1109/TEVC.2008.915992

Ghandar, A., Michalewicz, Z., and Zurbruegg, R. (2009). Return performance volatility and adaptation in an automated technical analysis approach to portfolio management. *Journal of Intelligent Systems in Accounting and Finance Management*, 16(1):127–146. 10.1002/isaf.297

Ghandar, A., Michalewicz, Z., and Zurbruegg, R. (2016). The relationship between model complexity and forecasting performance for computer intelligence optimization in finance. *International Journal of Forecasting*, 32(3):598–613. 10.1016/j.ijforecast.2015.10.003

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Grefenstette, J. (1987). Incorporating problem specific knowledge into genetic algorithms. *Genetic Algorithms and Simulated Annealing*, 42–60.

Holland, J. (1975). *Adaptation in natural and artificial systems*. MIT Press.

Janikow, C., and Michalewicz, Z. (1991). An experimental comparison of binary and floating point representations in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 31–36.

Johnson, R., Melich, M., Michalewicz, Z., and Schmidt, M. (2005). Coevolutionary optimization of fuzzy logic intelligence for strategic decision support. *IEEE Transactions on Evolutionary Computation*, 9(6):682–694. 10.1109/TEVC.2005.856208

Khouja, M., Michalewicz, Z., and Satoskar, S. (2000). A comparison between genetic algorithms and the RAND method for solving the joint replenishment problem. *Production Planning & Control*, 11(6):556–564.

Khouja, M., Michalewicz, Z., and Vijayaragavan, P. (1998). Evolutionary algorithm for economic lot and delivery scheduling problem. *Fundamenta Informaticae*, 35(1–4):113–123. 10.3233/FI-1998-35123407

Knight, P. (2016). *Shoe dog*. Simon & Schuster.

Kozieł, S., and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44.

Li, X., Bonyadi, M., Michalewicz, Z., and Barone, L. (2014). A hybrid evolutionary algorithm for wheat blending problem. *The Scientific World Journal*, special issue on Recent Advances on Bioinspired Computation, Article ID 967254.

Meyer, E., Falkner, N., Sooriamurthi, R., and Michalewicz, Z. (2014). *A guide to teaching puzzle-based learning*. Springer.

Michalewicz, M., and Michalewicz, Z. (2006). *Winning credibility: A guide for building a business from rags to riches*. Hybrid Publishers.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, *Artificial Intelligence Series*.

Michalewicz, Z. (1993). A hierarchy of evolution programs: An experimental study. *Evolutionary Computation*, 1(1):51–76. 10.1162/evco.1993.1.1.51

Michalewicz, Z. (2012a). Quo vadis, evolutionary computation? On a growing gap between theory and practice, pp. 98–121. *Lecture Notes in Computer Science*, Vol. 7311. 10.1007/978-3-642-30687-7_6

Michalewicz, Z. (2012b). The emperor is naked: Evolutionary algorithms for real-world applications. *ACM Ubiquity*, 1–13.

Michalewicz, Z., Deb, K., Schmidt, M., and Stidsen, T. (2000). Test-case generator for nonlinear continuous parameter optimization techniques. *IEEE Transactions on Evolutionary Computation*, 4(3):197–215. 10.1109/4235.873232

Michalewicz, Z., and Fogel, D. (2000). *How to solve it: Modern heuristics*. Springer.

Michalewicz, Z., and Janikow, C. (1991). Handling constraints in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*.

Michalewicz, Z., and Janikow, C. (1996). GENOCOP: A genetic algorithm for numerical optimization problems with linear constraints. *Communications of the ACM*, p. 118.

Michalewicz, Z., and Michalewicz, M. (2008). *Puzzle-based learning: An introduction to critical thinking, mathematics, and problem solving.* Hybrid Publishers.

Michalewicz, Z., and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32. 10.1162/evco.1996.4.1.1

Michalewicz, Z., Arantes, L., and Michalewicz, M. (2021). *The rise of artificial intelligence: Real-world applications for revenue and margin growth*. Hybrid Publishers.

Michalewicz, Z., Dasgupta, D., Le Riche, R. G., and Schoenauer, M. (1996). Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, 30(4):851–870.

Michalewicz, Z., Schmidt, M., Michalewicz, M., and Chiriac, C. (2005). Case study: An intelligent decision support system. *IEEE Intelligent Systems*, 20(4):44–49. 10.1109/MIS.2005.64

Michalewicz, Z., Schmidt, M., Michalewicz, M., and Chiriac, C. (2007). *Adaptive business intelligence*. Springer.

Michalewicz, Z., Vignaux, G. A., and Hobbs, M. (1991). A nonstandard genetic algorithm for the nonlinear transportation problem. *ORSA Journal on Computing*, 3(4):307–316. 10.1287/ijoc.3.4.307

Pólya, G. (1945). *How to solve it*. Princeton University Press.

Przybyłek, M., Wierzbicki, A., and Michalewicz, Z. (2018). Decomposition algorithms for a multi-hard problem. *Evolutionary Computation*, 26(3):507–533.

Schoenauer, M., and Michalewicz, Z. (1996). Evolutionary computation at the edge of feasibility. *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pp. 245–254. Lecture Notes in Computer Science, Vol. 1141.

Schoenauer, M., and Michalewicz, Z. (1997). Boundary operators for constrained parameter optimization problems. *Proceedings of the 7th International Conference on Genetic Algorithms*, 320–329.

Schoenauer, M., and Michalewicz, Z. (1998). Sphere operators and their applicability for constrained parameter optimization problems. *Proceedings of the 7th Annual Conference on Evolutionary Programming*, pp. 241–250. Lecture Notes in Computer Science, Vol. 1447.

Śmierzchalski, R., and Michalewicz, Z. (2000). Modeling of ship trajectory in collision situations by an evolutionary algorithm. *IEEE Transactions on Evolutionary Computation*, 4(3):227–241.

Tolstoy, L. (1877). *Anna Karenina*. Simon & Schuster.

Ullman, J. (2009). Advising students for success. *Communications of the ACM*, 52(3):34–37. 10.1145/1467247.1467260

Ursem, R., Krink, T., Jensen, M., and Michalewicz, Z. (2002). Analysis and modeling of control tasks in dynamic systems. *IEEE Transactions on Evolutionary Computation*, 6(4):378–389. 10.1109/TEVC.2002.802871

Vergara, F., Khouja, M., and Michalewicz, Z. (2002). An evolutionary algorithm for optimizing material flow in supply chains. *Computers & Industrial Engineering*, 43:407–421.

Vignaux, G., and Michalewicz, Z. (1991). A genetic algorithm for the linear transportation problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(2):445–452. 10.1109/21.87092

Wagner, N., Khouja, M., Michalewicz, Z., and McGregor, R. R. (2008). Forecasting economic time series with the DyFor genetic program model. *Journal of Applied Financial Economics*, 18(5):357–378. 10.1080/09603100600949200

Weise, T., Chiong, R., Lässig, J., Tang, K., Tsutsui, S., Chen, W., Michalewicz, Z., and Yao, X. (2014). Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 9(3):40–52. 10.1109/MCI.2014.2326101

Wieczorek, J., Gol, O., and Michalewicz, Z. (1998). An evolutionary algorithm for the optimal design of induction motors. *IEEE Transactions on Magnetics*, 34(6):3882–3887. 10.1109/20.728298

Wirth, N. (1976). *Algorithms + Data Structures = Programs*. Prentice-Hall.

Xiao, J., Michalewicz, Z., Zhang, L., and Trojanowski, K. (1997). Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, 1(1):18–28. 10.1109/4235.585889

Zhang, R., Chiong, R., Michalewicz, Z., and Chang, P.-C. (2016). Sustainable scheduling of manufacturing and transportation systems. *European Journal of Operations Research*, 248(3):74–743.