Evolutionary Computation for Real-World Problems

Mohammad Reza Bonyadi and Zbigniew Michalewicz

Abstract In this paper we discuss three topics that are present in the area of real-world optimization, but are often neglected in academic research in evolutionary computation community. First, problems that are a combination of several interacting sub-problems (so-called multi-component problems) are common in many real-world applications and they deserve better attention of research community. Second, research on optimisation algorithms that focus the search on the edges of feasible regions of the search space is important as high quality solutions usually are the boundary points between feasible and infeasible parts of the search space in many real-world problems. Third, finding bottlenecks and best possible investment in real-world processes are important topics that are also of interest in real-world optimization. In this chapter we discuss application opportunities for evolutionary computation methods in these three areas.

1 Introduction

The Evolutionary Computation (EC) community over the last 30 years has spent a lot of effort to design optimization methods (specifically Evolutionary Algorithms, EAs) that are well-suited for hard problems—problems where other methods usually

M.R. Bonyadi (✉) · Z. Michalewicz
Optimisation and Logistics, The University of Adelaide, Adelaide, Australia
e-mail: mrbonyadi@cs.adelaide.edu.au

Z. Michalewicz
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
e-mail: zbyszek@cs.adelaide.edu.au
url: http://cs.adelaide.edu.au/∼optlog/

Z. Michalewicz
Polish-Japanese Institute of Information Technology, Warsaw, Poland

Z. Michalewicz
Chief of Science, Complexica, Adelaide, Australia

S. Matwin and J. Mielniczuk (eds.), *Challenges in Computational Statistics and Data Mining*, Studies in Computational Intelligence 605,
DOI 10.1007/978-3-319-18781-5_1

fail [36]. As most real-world problems[1] are very hard and complex, with nonlinearities and discontinuities, complex constraints and business rules, possibly conflicting objectives, noise and uncertainty, it seems there is a great opportunity for EAs to be used in this area.

Some researchers investigated features of real-world problems that served as reasons for difficulties of EAs when applied to particular problems. For example, in [53] the authors identified several such reasons, including premature convergence, ruggedness, causality, deceptiveness, neutrality, epistasis, and robustness, that make optimization problems hard to solve. It seems that these reasons are either related to the landscape of the problem (such as ruggedness and deceptiveness) or the optimizer itself (like premature convergence and robustness) and they are not focusing on the nature of the problem. In [38], a few main reasons behind the hardness of real-world problems were discussed; that included: the size of the problem, presence of noise, multi-objectivity, and presence of constraints. Apart from these studies on features related to the real-world optimization, there have been EC conferences (e.g. GECCO, IEEE CEC, PPSN) during the past three decades that have had special sessions on "real-world applications". The aim of these sessions was to investigate the potentials of EC methods in solving real-world optimization problems.

Consequently, most of the features discussed in the previous paragraph have been captured in optimization benchmark problems (many of these benchmark problems can be found in OR-library[2]). As an example, the size of benchmark problems has been increased during the last decades and new benchmarks with larger problems have appeared: knapsack problems (KP) with 2,500 items or traveling salesman problems (TSP) with more than 10,000 cities, to name a few. Noisy environments have been already defined [3, 22, 43] in the field of optimization, in both continuous and combinatorial optimization domain (mainly from the operations research field), see [3] for a brief review on robust optimization. Noise has been considered for both constraints and objective functions of optimization problems and some studies have been conducted on the performance of evolutionary optimization algorithms with existence of noise; for example, stochastic TSP or stochastic vehicle routing problem (VRP). We refer the reader to [22] for performance evaluation of evolutionary algorithms when the objective function is noisy. Recently, some challenges to deal with continuous space optimization problems with noisy constraints were discussed and some benchmarks were designed [43]. Presence of constraints has been also captured in benchmark problems where one can generate different problems with different constraints, for example Constrained VRP, (CVRP). Thus, the expectation is, after capturing all of these pitfalls and addressing them (at least some of them), EC optimization methods should be effective in solving real-world problems.

However, after over 30 years of research, tens of thousands of papers written on Evolutionary Algorithms, dedicated conferences (e.g. GECCO, IEEE CEC, PPSN),

[1] By real-world problems we mean problems which are found in some business/industry on daily (regular) basis. See [36] for a discussion on different interpretations of the term "real-world problems".

[2] Available at: http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

dedicated journals (e.g. Evolutionary Computation Journal, IEEE Transactions on Evolutionary Computation), special sessions and special tracks on most AI-related conferences, special sessions on real-world applications, etc., still it is not that easy to find EC-based applications in real-world, especially in real-world supply chain industries.

There are several reasons for this mismatch between the efforts of hundreds of researchers who have been making substantial contribution to the field of Evolutionary Computation over many years and the number of real-world applications which are based on concepts of Evolutionary Algorithms—these are discussed in detail in [37]. In this paper we summarize our recent efforts (over the last two years) to close the gap between research activities and practice; these efforts include three research directions:

- Studying multi-component problems [7]
- Investigating boundaries between feasible and infeasible parts of the search space [5]
- Examining bottlenecks [11].

The paper is based on our four earlier papers [5, 7, 9, 11] and is organized as follows. We start with presenting two real-world problems (Sect. 2) so the connection between presented research directions and real-world problems is apparent. Sections 3–5 summarize our current research on studying multi-component problems, investigating boundaries between feasible and infeasible parts of the search space, and examining bottlenecks, respectively. Section 6 concludes the paper.

2 Example Supply Chains

In this section we explain two real-world problems in the field of supply chain management. We refer to these two examples further in the paper.

Transportation of water tank The first example relates to optimization of the transportation of water tanks [21]. An Australian company produces water tanks with different sizes based on some *orders* coming from its customers. The number of customers per month is approximately 10,000; these customers are in different locations, called *stations*. Each customer orders a water tank with specific characteristics (including size) and expects to receive it within a period of time (usually within 1 month). These water tanks are carried to the stations for delivery by a fleet of trucks that is operated by the water tank company. These trucks have different characteristics and some of them are equipped with trailers. The company proceeds in the following way. A subset of orders is selected and assigned to a truck and the delivery is scheduled in a limited period of time. Because the tanks are empty and of different sizes they might be packed inside each other in order to maximize trucks load in a trip. A bundled tank must be unbundled at special sites, called *bases*, before the tank delivery to stations. Note that there might exist several bases close to the

stations where the tanks are going to be delivered and selecting different bases affects the best overall achievable solution. When the tanks are unbundled at a base, only some of them fit in the truck as they require more space. The truck is loaded with a subset of these tanks and deliver them to their corresponding stations for delivery. The remaining tanks are kept in the base until the truck gets back and loads them again to continue the delivery process.

The aim of the optimizer is to divide all tanks ordered by customers into subsets that are bundled and loaded in trucks (possibly with trailers) for delivery. Also, the optimizer needs to determine an exact routing for bases and stations for unbundling and delivery activities. The objective is to maximize the profit of the delivery at the end of the time period. This total profit is proportional to the ratio between the total prices of delivered tanks to the total distance that the truck travels.

Each of the mentioned procedures in the tank delivery problem (subset selection, base selection, and delivery routing, and bundling) is just one component of the problem and finding a solution for each component in isolation does not lead us to the optimal solution of the whole problem. As an example, if the subset selection of the orders is solved optimally (the best subset of tanks is selected in a way that the price of the tanks for delivery is maximized), there is no guarantee that there exist a feasible bundling such that this subset fits in a truck. Also, by selecting tanks without considering the location of stations and bases, the best achievable solutions can still have a low quality, e.g. there might be a station that needs a very expensive tank but it is very far from the base, which actually makes delivery very costly. On the other hand, it is impossible to select the best routing for stations before selecting tanks without selection of tanks, the best solution (lowest possible tour distance) is to deliver nothing. Thus, solving each sub-problem in isolation does not necessarily lead us to the overall optimal solution.

Note also that in this particular case there are many additional considerations that must be taken into account for any successful application. These include scheduling of drivers (who often have different qualifications), fatigue factors and labor laws, traffic patterns on the roads, feasibility of trucks for particular segments of roads, and maintenance schedule of the trucks.

Mine to port operation The second example relates to optimizing supply-chain operations of a mining company: from mines to ports [31, 32]. Usually in mine to port operations, the mining company is supposed to satisfy customer orders to provide predefined amounts of products (the raw material is dig up in mines) by a particular due date (the product must be ready for loading in a particular port). A port contains a huge area, called *stockyard*, several places to berth the ships, called *berths*, and a waiting area for the ships. The stockyard contains some *stockpiles* that are single-product storage units with some capacity (mixing of products in stockpiles is not allowed). Ships arrive in ports (time of arrival is often approximate, due to weather conditions) to take specified products and transport them to the customers. The ships wait in the waiting area until the port manager assigns them to a particular berth. Ships apply a cost penalty, called *demurrage*, for each time unit while it is waiting to be berthed since its arrival. There are a few *ship loaders* that are assigned to each

berthed ship to load it with demanded products. The ship loaders take products from appropriate stockpiles and load them to the ships. Note that, different ships have different product demands that can be found in more than one stockpile, so that scheduling different ship loaders and selecting different stockpiles result in different amount of time to fulfill the ships demand. The goal of the mine owner is to provide sufficient amounts of each product type to the stockyard. However, it is also in the interest of the mine owner to minimize costs associated with early (or late) delivery, where these are estimated with respect to the (scheduled) arrival of the ship. Because mines are usually far from ports, the mining company has a number of trains that are used to transport products from a mine to the port. To operate trains, there is a rail network that is (usually) rented by the mining company so that trains can travel between mines and ports. The owner of the rail network sets some constraints for the operation of trains for each mining company, e.g. the number of passing trains per day through each junction (called *clusters*) in the network is a constant (set by the rail network owner) for each mine company.

There is a number of *train dumpers* that are scheduled to unload the products from the trains (when they arrive at port) and put them in the stockpiles. The mine company schedules trains and loads them at mine sites with appropriate material and sends them to the port while respecting all constraints (the *train scheduling* procedure). Also, scheduling train dumpers to unload the trains and put the unloaded products in appropriate stockpiles (the *unload scheduling* procedure), scheduling the ships to berth (this called *berthing* procedure), and scheduling the ship loaders to take products from appropriate stockpiles and load the ships (the *loader scheduling* procedure) are the other tasks for the mine company. The aim is to schedule the ships and fill them with the required products (ship demands) so that the total demurrage applied by all ships is minimized in a given time horizon.

Again, each of the aforementioned procedures (train scheduling, unload scheduling, berthing, and loader scheduling) is one component of the problem. Of course each of these components is a hard problem to solve by its own. Apart from the complication in each component, solving each component in isolation does not lead us to an overall solution for the whole problem. As an example, scheduling trains to optimality (bringing as much product as possible from mine to port) might result in insufficient available capacity in the stockyard or even lack of adequate products for the ships that arrive unexpectedly early. That is to say, ship arrival times have uncertainty associated with them (e.g. due to seasonal variation in weather conditions), but costs are independent of this uncertainty. Also, the best plan for dumping products from trains and storing them in the stockyard might result in a low quality plan for the ship loaders and result in too much movement to load a ship.

Note that, in the real-world case, there were some other considerations in the problem such as seasonal factor (the factor of constriction of the coal), hatch plan of ships (each product should be loaded in different parts of the ship to keep the balance of the vessel), availability of the drivers of the ship loaders, switching times between changing the loading product, dynamic sized stockpiles, etc.

Both problems illustrate the main issues discussed in the remaining sections of this document, as (1) they consist of several inter-connected components, (2) their

boundaries between feasible and infeasible areas of the search space deserve careful examination, and (3) in both problems, the concept of bottleneck is applicable.

3 Multi-component Problems

There are thousands of research papers addressing traveling salesman problems, job shop and other scheduling problems, transportation problems, inventory problems, stock cutting problems, packing problems, various logistic problems, to name but a few. While most of these problems are NP-hard and clearly deserve research efforts, it is not exactly what the real-world community needs. Let us explain.

Most companies run complex operations and they need solutions for problems of high complexity with several components (i.e. multi-component problems; recall examples presented in Sect. 2). In fact, real-world problems usually involve several smaller sub-problems (several components) that interact with each other and companies are after a solution for the whole problem that takes all components into account rather than only focusing on one of the components. For example, the issue of scheduling production lines (e.g. maximizing the efficiency or minimizing the cost) has direct relationships with inventory costs, stock-safety levels, replenishments strategies, transportation costs, delivery-in-full-on-time (DIFOT) to customers, etc., so it should not be considered in isolation. Moreover, optimizing one component of the operation may have negative impact on upstream and/or downstream activities. These days businesses usually need "global solutions" for their operations, not component solutions. This was recognized over 30 years ago by Operations Research (OR) community; in [1] there is a clear statement: *Problems require holistic treatment. They cannot be treated effectively by decomposing them analytically into separate problems to which optimal solutions are sought.* However, there are very few research efforts which aim in that direction mainly due to the lack of appropriate benchmarks or test cases availability. It is also much harder to work with a company on such global level as the delivery of successful software solution usually involves many other (apart from optimization) skills, from understanding the companys internal processes to complex software engineering issues.

Recently a new benchmark problem called the traveling thief problem (TTP) was introduced [7] as an attempt to provide an abstraction of multi-component problems with dependency among components. The main idea behind TTP was to combine two problems and generate a new problem which contains two components. The TSP and KP were combined because both of these problems were investigated for many years in the field of optimization (including mathematics, operations research, and computer science). TTP was defined as a thief who is going to steal m items from n cities and the distance of the cities ($d(i, j)$ the distance between cities i and j), the profit of each item (p_i), and the weight of the items (w_i) are given. The thief is carrying a limited-capacity knapsack (maximum capacity W) to collect the stolen items. The problem is asked for the best plan for the thief to visit all cities exactly once (traveling salesman problem, TSP) and pick the items (knapsack problem, KP) from

these cities in a way that its total benefit is maximized. To make the two sub-problems dependent, it was assumed that the speed of the thief is affected by the current weight of the knapsack (W_c) so that the more item the thief picks, the slower he can run. A function $v : \mathbb{R} \rightarrow \mathbb{R}$ is given which maps the current weight of the knapsack to the speed of thief. Clearly, $v(0)$ is the maximum speed of the thief (empty knapsack) and $v(W)$ is the minimum speed of the thief (full knapsack). Also, it was assumed that the thief should pay some of the profit by the time he completes the tour (e.g. rent of the knapsack, r). The total amount that should be paid is a function of the tour time. The total profit of the thief is then calculated by

$$B = P - r \times T$$

where B is the total benefit, P is the aggregation of the profits of the picked items, and T is the total tour time.

Generating a solution for KP or TSP in TTP is possible without being aware of the current solution for the other component. In addition, each solution for TSP impacts the best quality that can be achieved in the KP component because of the impact on the pay back that is a function of travel time. Moreover, each solution for the KP component impacts the tour time for TSP as different items impact the speed of travel differently due to the variability of weights of items. Some test problems were generated for TTP and some simple heuristic methods have been also applied to the problem [44].

Note that for a given instance of TSP and KP different values of r and functions f result in different instances of TTPs that might be harder or easier to solve. As an example, for small values of r (relative to P), the value of $r \times T$ has a small contribution to the value of B. In an extreme case, when $r = 0$, the contribution of $r \times T$ is zero, which means that the best solution for a given TTP is equivalent to the best solution of the KP component, hence, there is no need to solve the TSP component at all. Also, by increasing the value of r (relative to P), the contribution of $r \times T$ becomes larger. In fact, if the value of r is very large then the impact of P on B becomes negligible, which means that the optimum solution of the TTP is very close to the optimum solution of the given TSP (see Fig. 1).

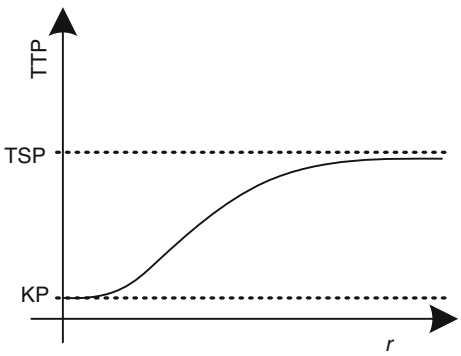

Fig. 1 Impact of the rent rate r on the TTP. For $r = 0$, the TTP solution is equivalent to the solution of KP, while for larger r the TTP solutions become closer to the solutions of TSP

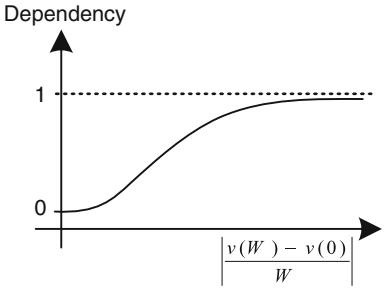

Fig. 2 How dependency between components is affected by speed (function v). When v does not drop significantly for different weights of picked items ($\left|\frac{v(W)-v(0)}{W}\right|$ is small), the two problems can be decomposed and solved separately. The value Dependency = 1 represents the two components are dependent while Dependency = 0 shows that two components are not dependent

The same analysis can be done for the function v. In fact, for a given TSP and KP different function v can result in different instances of TTPs that, as before, might be harder or easier. Let us assume that v is a decreasing function, i.e. picking items with positive weight causes drop or no change in the value of v. For a given list of items and cities, if picking an item does not affect the speed of the travel (i.e. $\left|\frac{v(W)-v(0)}{W}\right|$ is zero) significantly then the optimal solution of the TTP is the composition of the optimal solution of KP and TSP when they are solved separately. The reason is that, with this setting ($\left|\frac{v(W)-v(0)}{W}\right|$ is zero), picking more items does not change the time of the travel. As the value of $\left|\frac{v(W)-v(0)}{W}\right|$ grows, the TSP and KP become more dependent (picking items have more significant impact on the travel time); see Fig. 2.

As the value of $\left|\frac{v(W)-v(0)}{W}\right|$ grows, the speed of the travel drops more significantly by picking more items that in fact reduces the value of B significantly. In an extreme case, if $\left|\frac{v(W)-v(0)}{W}\right|$ is infinitely large then it would be better not to pick any item (the solution for KP is to pick no item) and only solve the TSP part as efficiently as possible. This has been also discussed in [10].

Recently, we generated some test instances for TTP and made them available [44] so that other researchers can also work along this path. The instance set contains 9,720 problems with different number of cities and items. The specification of the tour was taken from existing TSP problems in OR-Library. Also, we proposed three algorithms to solve those instances: one heuristic, one random search with local improvement, and one simple evolutionary algorithm. Results indicated that the evolutionary algorithm outperforms other methods to solve these instances. These test sets were also used in a competition in CEC2014 where participants were asked to come up with their algorithms to solve the instances. Two popular approaches emerged: combining different solvers for each sub-problem and creating one system for the overall problem.

Problems that require the combination of solvers for different sub-problems, one can find different approaches in the literature. First, in bi-level-optimization (and in the more general multi-level-optimization), one component is considered the dominant one (with a particular solver associated to it), and every now and then the other component(s) are solved to near-optimality or at least to the best extent possible by other solvers. In its relaxed form, let us call it "round-robin optimization", the optimization focus (read: CPU time) is passed around between the different solvers for the subcomponents. For example, this approach is taken in [27], where two heuristics are applied alternatingly to a supply-chain problem, where the components are (1) a dynamic lot sizing problem and (2) a pickup and delivery problem with time windows. However, in neither set-up did the optimization on the involved components commence in parallel by the solvers.

A possible approach to multi-component problems with presence of dependencies is based on the cooperative coevolution: a type of multi-population Evolutionary Algorithm [45]. Coevolution is a simultaneous evolution of several genetically isolated subpopulations of individuals that exist in a common ecosystem. Each subpopulation is called species and mate only within its species. In EC, coevolution can be of three types: competitive, cooperative, and symbiosis. In competitive coevolution, multiple species coevolve separately in such a way that fitness of individual from one species is assigned based on how good it competes against individuals from the other species. One of the early examples of competitive coevolution is the work by Hillis [20], where he applied a competitive predator-prey model to the evolution of sorting networks. Rosin and Belew [47] used the competitive model of coevolution to solve number of game learning problems including Tic-Tac-Toe, Nim and small version of Go. Cooperative coevolution uses divide and conquer strategy: all parts of the problem evolve separately; fitness of individual of particular species is assigned based on the degree of collaboration with individuals of other species. It seems that cooperative coevolution is a natural fit for multi-component problems with presence of dependencies. Individuals in each subpopulation may correspond to potential solutions for particular component, with its own evaluation function, whereas the global evaluation function would include dependencies between components. Symbiosis is another coevolutionary process that is based on living together of organisms of different species. Although this type appears to represent a more effective mechanism for automatic hierarchical models [19], it has not been studied in detail in the EC literature.

Additionally, feature-based analysis might be helpful to provide new insights and help in the design of better algorithms for multi-component problems. Analyzing statistical feature of classical combinatorial optimization problems and their relation to problem difficulty has gained an increasing attention in recent years [52]. Classical algorithms for the TSP and their success depending on features of the given input have been studied in [34, 41, 51] and similar analysis can be carried out for the knapsack problem. Furthermore, there are different problem classes of the knapsack problem which differ in their hardness for popular algorithms [33]. Understanding the features of the underlying sub-problems and how the features of interactions in a multi-component problem determine the success of different algorithms is an

interesting topic for future research which would guide the development and selection of good algorithms for multi-component problems.

In the field of machine learning, the idea of using multiple algorithms to solve a problem in a better way has been used for decades. For example, ensemble methods—such as boosting, bagging, and stacking—use multiple learning algorithms to search the hypothesis space in different ways. In the end, the predictive performance of the combined hypotheses is typically better than the performances achieved by the constituent approaches.

Interestingly, transferring this idea into the optimization domain is not straightforward. While we have a large number of optimizers at our disposal, they are typically not general-purpose optimizers, but very specific and highly optimized for a particular class of problems, e.g., for the knapsack problem or the travelling salesperson problem.

4 Boundaries Between Feasible and Infeasible Parts of the Search Space

A constrained optimization problem (COP) is formulated as follows:

$$\text{find } x \in \mathcal{F} \subseteq \mathrm{S} \subseteq R^D \text{ such that } \begin{cases} f(x) \leq f(y) & \text{for all } y \in \mathcal{F} \quad \text{(a)} \\ g_i(\mathrm{x}) \leq 0 & \text{for } i = 1 \text{ to } q \quad \text{(b)} \\ h_i(x) = 0 & \text{for } i = q+1 \text{ to } m \quad \text{(c)} \end{cases} \tag{1}$$

where f, g_i, and h_i are real-valued functions on the search space S, q is the number of inequalities, and $m - q$ is the number of equalities. The set of all feasible points which satisfy constraints (b) and (c) are denoted by $\mathcal{F}$ [39]. The equality constraints are usually replaced by $|h_i(x)| - \sigma \leq 0$ where σ is a small value (normally set to 10^{-4}) [6]. Thus, a COP is formulated as

$$\text{find } x \in \mathcal{F} \subseteq \mathrm{S} \subseteq R^D \text{ such that } \begin{cases} f(x) \leq f(y) & \text{for all } y \in \mathcal{F} \quad \text{(a)} \\ g_i(\mathrm{x}) \leq 0 & \text{for } i = 1 \text{ to } m \quad \text{(b)} \end{cases} \tag{2}$$

where $g_i(x) = |h_i(x)| - \sigma$ for all $i \in \{q+1, \ldots, m\}$. Hereafter, the term COP refers to this formulation.

The constraint $g_i(x)$ is called *active* at the point x if the value of $g_i(x)$ is zero. Also, if $g_i(x) < 0$ then $g_i(x)$ is called *inactive* at x. Obviously, if x is feasible and at least one of the constraints is active at x, then x is on the boundary of the feasible and infeasible areas of the search space.

In many real-world COPs it is highly probable that some constraints are active at optimum points [49], i.e. some optimum points are on the edge of feasibility. The reason is that constraints in real-world problems often represent some limitations of

resources. Clearly, it is beneficial to make use of some resources as much as possible, which means constraints are active at quality solutions. Presence of active constraints at the optimum points causes difficulty for many optimization algorithms to locate optimal solution [50]. Thus, it might be beneficial if the algorithm is able to focus the search on the edge of feasibility for quality solutions.

So it is assumed that there exists at least one active constraint at the optimum solution of COPs. We proposed [5] a new function, called Subset Constraints Boundary Narrower (SCBN), that enabled the search methods to focus on the boundary of feasibility with an adjustable thickness rather than the whole search space. SCBN is actually a function (with a parameter ε for thickness) that, for a point x, its value is smaller than zero if and only if x is feasible and the value of *at least one* of the constraints in *a given subset* of all constraint of the COP at the point x is within a predefined boundary with a specific thickness. By using SCBN in any COP, the feasible area of the COP is limited to the boundary of feasible area defined by SCBN, so that the search algorithms can only focus on the boundary. Some other extensions of SCBN are proposed that are useful in different situations. SCBN and its extensions are used in a particle swarm optimization (PSO) algorithm with a simple constraint handling method to assess if they are performing properly in narrowing the search on the boundaries.

A COP can be rewritten by combining all inequality constraints to form only one inequality constraint. In fact, any COP can be formulated as follows:

$$\text{find } x \in \mathcal{F} \subseteq \mathrm{S} \subseteq R^D \text{ such that } \begin{cases} f(x) \leq f(y) \text{ for all } y \in \mathcal{F} & \text{(a)} \\ M(\mathrm{x}) \leq 0 & \text{(b)} \end{cases} \tag{3}$$

where $M(x)$ is a function that combines all constraints $g_i(x)$ into one function. The function $M(x)$ can be defined in many different ways. The surfaces that are defined by different instances of $M(x)$ might be different. The inequality 3(b) should capture the feasible area of the search space. However, by using problem specific knowledge, one can also define $M(x)$ in a way that the area that is captured by $M(x) \leq 0$ only refers to a sub-space of the whole feasible area where high quality solutions might be found. In this case, the search algorithm can focus only on the captured area which is smaller than the whole feasible area and make the search more effective. A frequently-used [29, 48] instance of $M(x)$ is a function $K(x)$

$$K(x) = \sum_{i=1}^{m} \max\{g_i(x), 0\} \tag{4}$$

Clearly, the value of $K(x)$ is non-negative. $K(x)$ is zero if and only if x is feasible. Also, if $K(x) > 0$, the value of $K(x)$ represents the maximum violation value (called the *constraint violation* value).

As in many real-world COPs, there is at least one active constraint near the global best solution of COPs [49], some researchers developed operators to enable search

methods to focus the search on the edges of feasibility. GENOCOP (GEnetic algorithm for Numerical Optimization for Constrained Optimization) [35] was probably the first genetic algorithm variant that applied boundary search operators for dealing with COPs. Indeed, GENOCOP had three mutations and three crossovers operators and one of these mutation operators was a boundary mutation which could generate a random point on the boundary of the feasible area. Experiments showed that the presence of this operator caused significant improvement in GENOCOP for finding optimum for problems which their optimum solution is on the boundary of feasible and infeasible area [35].

A specific COP was investigated in [40] and a specific crossover operator, called *geometric crossover*, was proposed to deal with that COP. The COP was defined as follows:

$$\begin{aligned} f(x) &= \left| \frac{\sum_{i=1}^{D} cos^4(x_i) - 2\prod_{i=1}^{D} cos^2(x_i)}{\sqrt{\sum_{i=1}^{D} i x_i^2}} \right| \\ g_1(x) &= 0.75 - \prod_{i=1}^{D} x_i \leq 0 \\ g_2(x) &= \sum_{i=1}^{D} x_i - 0.75D \leq 0 \end{aligned} \tag{5}$$

where $0 \leq x_i \leq 10$ for all i. Earlier experiments [23] shown that the value of the first constraint ($g_1(x)$) is very close to zero at the best known feasible solution for this COP. The geometric crossover was designed as $x_{new,j} = \sqrt{x_{1,i} x_{2,j}}$, where $x_{i,j}$ is the value of the jth dimension of the ith parent, and $x_{new,j}$ is the value of the jth dimension of the new individual. By using this crossover, if $g_1(\mathbf{x}_1) = g_1(\mathbf{x}_2) = 0$, then $g_1(\mathbf{x}_{new}) = 0$ (the crossover is *closed* under $g_1(x)$). It was shown that an evolutionary algorithm that uses this crossover is much more effective than an evolutionary algorithm which uses other crossover operators in dealing with this COP. In addition, another crossover operator was also designed [40], called *sphere crossover*, that was closed under the constraint $g(x) = \sum_{i=1}^{D} x_i^2 - 1$. In the sphere crossover, the value of the new offspring was generated by $x_{new,j} = \sqrt{\alpha x_{1,j}^2 + (1-\alpha) x_{2,j}^2}$, where $x_{i,j}$ is the value of the jth dimension of the ith parent, and both parents $\mathbf{x}_1$ and $\mathbf{x}_2$ are on $g(x)$. This operator could be used if $g(x)$ is the constraint in a COP and it is active on the optimal solution.

In [50] several different crossover operators closed under $g(x) = \sum_{i=1}^{D} x_i^2 - 1$ were discussed. These crossovers operators included repair, sphere (explained above), *curve*, and *plane* operators. In the repair operator, each generated solution was normalized and then moved to the surface of $g(x)$. In this case, any crossover and mutation could be used to generate offspring; however, the resulting offspring is moved (repaired) to the surface of $g(x)$. The curve operator was designed in a way that it could generate points on the *geodesic curves*, curves with minimum length on

a surface, on $g(x)$. The plane operator was based on the selection of a plane which contains both parents and crosses the surface of $g(x)$. Any point on this intersection is actually on the surface of the $g(x)$ as well. These operators were incorporated into several optimization methods such as GA and Evolutionary Strategy (ES) and the results of applying these methods to two COPs were compared.

A variant of evolutionary algorithm for optimization of a water distribution system was proposed [54]. The main argument was that the method should be able to make use of information on the edge between infeasible and feasible area to be effective in solving the water distribution system problem. The proposed approach was based on an adapting penalty factor in order to guide the search towards the boundary of the feasible search space. The penalty factor was changed according to the percentage of the feasibility of the individuals in the population in such a way that there are always some infeasible solutions in the population. In this case, crossover can make use of these infeasible and feasible individuals to generate solutions on the boundary of feasible region.

In [28] a boundary search operator was adopted from [35] and added to an ant colony optimization (ACO) method. The boundary search was based on the fact that the line segment that connects two points x and y, where one of these points are infeasible and the other one is feasible, crosses the boundary of feasibility. A binary search can be used to search along this line segment to find a point on the boundary of feasibility. Thus, any pair of points (x, y), where one of them is infeasible and the other is feasible, represents a point on the boundary of feasibility. These points were moved by an ACO during the run. Experiments showed that the algorithm is effective in locating optimal solutions that are on the boundary of feasibility.

In [5] we generalized the definition of edges of feasible and infeasible space by introducing thickness of the edges. We also introduced a formulation that, for any given COP, it could generate another COP that the feasible area of the latter corresponds to the edges of feasibility of the former COP. Assume that for a given COP, it is known that *at least one* of the constraints in the set $\{g_{i\in\Omega}(x)\}$ is active at the optimum solution and the remaining constraints are satisfied at x, where $\Omega \subseteq \{1, 2, \ldots, m\}$. We defined $H_{\Omega,\varepsilon}(x)$ as follows:

$$H_{\Omega,\varepsilon}(x) = \max\left\{\left|\max_{i\in\Omega}\{g_i(x)\} + \varepsilon\right| - \varepsilon,\ \max_{i\notin\Omega}\{g_i(x)\}\right\} \tag{6}$$

where ε is a positive value. Obviously, $H_{\Omega,\varepsilon}(x) \leq 0$ if and only if at least one of the constraints in the subset Ω is active and the others are satisfied. The reason is that, the component $\left|\max_{i\in\Omega}\{g_i(x)\} + \varepsilon\right| - \varepsilon$ is negative if x is feasible and at least one of $g_{i\in\Omega}(x)$ is active. Also, the component $\max_{i\notin\Omega}\{g_i(x)\}$ ensures that the rest of constraints are satisfied. Note that active constraints are considered to have a value between 0 and -2ε, i.e., the value of 2ε represents the thickness of the edges. This formulation can restrict the feasible search space to only the edges so that optimization algorithms are enforced to search the edges. Also, it enabled the user to

provide a list of active constraints so that expert knowledge can help the optimizer to converge faster to better solutions.

Clearly methodologies that focuses the search on the edges of feasible area are beneficial for optimization in real-world. As an example, in the mining problem described in Sect. 2, it is very likely that using all of the trucks, trains, shiploaders, and train dumpers to the highest capacity is beneficial for increasing throughput. Thus, at least one of these constraints (resources) is active, which means that searching the edges of feasible areas of the search space very likely leads us to high quality solutions.

5 Bottlenecks

Usually real-world optimization problems contain constraints in their formulation. The definition of constraints in management sciences is anything that limits a system from achieving higher performance versus its goal [17]. In the previous section we provided general formulation of a COP. As discussed in the previous section, it is believed that the optimal solution of most real-world optimization problems is found on the edge of a feasible area of the search space of the problem [49]. This belief is not limited to computer science, but it is also found in operational research (linear programming, LP) [12] and management sciences (theory of constraints, TOC) [30, 46] articles. The reason behind this belief is that, in real-world optimization problems, constraints usually represent limitations of availability of resources. As it is usually beneficial to utilize the resources as much as possible to achieve a high-quality solution (in terms of the objective value, f), it is expected that the optimal solution is a point where a subset of these resources is used as much as possible, i.e., $g_i\ (x^*) = 0$ for some i and a particular high-quality x^* in the general formulation of COPs [5]. Thus, the best feasible point is usually located where the value of these constraints achieves their maximum values (0 in the general formulation). The constraints that are active at the optimum solution can be thought of as *bottlenecks* that constrain the achievement of a better objective value [13, 30].

Decision makers in industries usually use some tools, known as decision support systems (DSS) [24], as a guidance for their decisions in different areas of their systems. Probably the most important areas that decision makers need guidance from DSS are: (1) optimizing schedules of resources to gain more benefit (accomplished by an optimizer in DSS), (2) identifying bottlenecks (accomplished by analyzing constraints in DSS), and (3) determining the best ways for future investments to improve their profits (accomplished by an analysis for removing bottlenecks,[3] known as what-if analysis in DSS). Such support tools are more readily available than one

[3]The term removing a bottleneck refers to the investment in the resources related to that bottleneck to prevent those resources from constraining the problem solver to achieve better objective values.

might initially think: for example, the widespread desktop application Microsoft Excel provides these via an add-in.[4]

Identification of bottlenecks and the best way of investment is at least as valuable as the optimization in many real-world problems from an industrial point of view because [18]: *An hour lost at a bottleneck is an hour lost for the entire system. An hour saved at a non-bottleneck is a mirage.* Industries are not only after finding the best schedules of the resources in their systems (optimizing the objective function), but they are also after understanding the tradeoffs between various possible investments and potential benefits.

During the past 30 years, evolutionary computation methodologies have provided appropriate tools as optimizers for decision makers to optimize their schedules. However, the last two areas (identifying bottlenecks and removing them) that are needed in DSSs seem to have remained untouched by EC methodologies while it has been an active research area in management and operations research.

There have been some earlier studies on identifying and removing bottlenecks [14, 16, 25, 30]. These studies, however, have assumed only linear constraints and they have related bottlenecks only to one specific property of resources (usually the availability of resources). Further, they have not provided appropriate tools to guide decision makers in finding the best ways of investments in their system so that their profits are maximized by removing the bottlenecks. In our recent work [11], we investigated the most frequently used bottleneck removing analysis (so-called average shadow prices) and identified its limitations. We argued that the root of these limitations can be found in the interpretation of constraints and the definition of bottlenecks. We proposed a more comprehensive definition for bottlenecks that not only leads us to design a more comprehensive model for determining the best investment in the system, but also addresses all mentioned limitations. Because the new model was multi-objective and might lead to the formulation of non-linear objective functions/constraints, evolutionary algorithms have a good potential to be successful on this proposed model. In fact, by applying multi-objective evolutionary algorithms to the proposed model, the solutions found represent points that optimize the objective function and the way of investment with different budgets at the same time.

Let us start with providing some background information on linear programming, the concept of shadow price, and bottlenecks in general. A Linear Programming (LP) problem is a special case of COP, where $f(x)$ and $g_i(x)$ are linear functions:

$$\text{find } x \text{ such that } z = \max c^T x \text{ subject to } Ax \leq b^T \tag{7}$$

where A is a $m \times d$ dimensional matrix known as *coefficients matrix*, m is the number of constraints, d is the number of dimensions, c is a d-dimensional vector, b is a m-dimensional vector known as Right Hand Side (RHS), $x \in \mathbb{R}^d$, and $x \geq 0$.

[4] http://tinyurl.com/msexceldss, last accessed 29th March 2014.

The shadow price (SP) for the ith constraint of this problem is the value of z when b_i is increased by one unit. This in fact refers to the best achievable solution if the RHS of the ith constraint was larger, i.e., there were more available resources of the type i [26].

The concept of SP in Integer Linear Programming (ILP) is different from the one in LP [13]. The definition for ILP is similar to the definition of LP, except that $x \in \mathbb{Z}^d$. In ILP, the concept of Average Shadow Price (ASP) was introduced [25]. Let us define the *perturbation function* $z_i(w)$ as follows:

$$\text{find } x \text{ such that } z_i(w) = \max c^T x \text{ subject to } a_i x \leq b_i + w \; a_k x \leq b_k \; \forall k \neq i \quad (8)$$

where a_i is the ith row of the matrix A and $x \geq 0$. Then, the ASP for the ith constraint is defined by $ASP_i = \sup_{w>0} \left\{ \frac{(z_i(w) - z_i(0))}{w} \right\}$. ASP_i represents that if adding one unit of the resource i costs p and $p < ASP_i$, then it is beneficial (the total profit is increased) to buy w units of this resource. This information is very valuable for the decision maker as it is helpful for removing bottlenecks. Although the value of ASP_i refers to "buying" new resources, it is possible to similarly define a selling shadow price [25].

Several extensions of this ASP definition exist. For example, a set of resources is considered in [15] rather than only one resource at a time. There, it was also shown that ASP can be used in mixed integer LP (MILP) problems.

Now, let us take a step back from the definition of ASP in the context of ILP, and let us see how it fits into a bigger picture of resources and bottlenecks. As we mentioned earlier, constraints usually model availability of resources and limit the optimizers to achieve the best possible solution which maximizes (minimizes) the objective function [26, 30, 46]. Although finding the best solution with the current resources is valuable for decision makers, it is also valuable to explore opportunities to improve solutions by adding more resources (e.g., purchasing new equipment) [25]. In fact, industries are seeking the most efficient way of investment (removing the bottlenecks) so that their profit is improved the most.

Let us assume that the decision maker has the option of providing some additional resource of type i at a price p. It is clearly valuable if the problem solver can determine if adding a unit of this resource can be beneficial in terms of improving the best achievable objective value. It is not necessarily the case that adding a new resource of the type i improves the best achievable objective value. As an example, consider there are some trucks that load products into some trains for transportation. It might be the case that adding a new train does not provide any opportunity for gaining extra benefit because the current number of trucks is too low and they cannot fill the trains in time. In this case, we can say that the number of trucks is a bottleneck. Although it is easy to define bottleneck intuitively, it is not trivial to define this term in general.

There are a few different definitions for bottlenecks. These definitions are categorized into five groups in [13]: (i) capacity based definitions, (ii) critical path based definitions, (iii) structure based definitions, (iv) algorithm based definitions, and (v) system performance based definitions. It was claimed that none of these definitions

was comprehensive and some examples were provided to support this claim. Also, a new definition was proposed which was claimed to be the most comprehensive definition for a bottleneck: "a set of constraints with positive average shadow price" [13]. In fact, the average shadow price in a linear and integer linear program can be considered as a measure for bottlenecks in a system [30].

Although ASP can be useful in determining the bottlenecks in a system, it has some limitations when it comes to removing bottlenecks. In this section, we discuss some limitations of removing bottlenecks based on ASP.

Obviously, the concept of ASP has been only defined for LP and MILP, but not for problems with non-linear objective functions and constraints. Thus, using the concept of ASP prevents us from identifying and removing bottlenecks in a non-linear system.

Let us consider the following simple problem[5] (the problem is extremely simple and it has been only given as an example to clarify limitations of the previous definitions): in a mine operation, there are 19 trucks and two trains. Trucks are used to fill trains with some products and trains are used to transport products to a destination. The rate of the operation for each truck is 100 tonnes/h (tph) and the capacity of each train is 2,000 tonnes. What is the maximum tonnage that can be loaded to the trains in 1 h? The ILP model for this problem is given by:

$$\begin{gathered} \text{find } x \text{ and } y \text{ s.t. } z = \max\{2000y\} \text{ subject to} \\ g_1 : 2000y - 100x \leq 0,\ g_2 : x \leq 19,\ g_3 : y \leq 2 \end{gathered} \tag{9}$$

where $x \geq 0$ is the number of trucks and $y \geq 0$ is the number of loaded trains (y can be a floating point value which refers to partially loaded trains). The constraint g_1 limits the amount of products loaded by the trucks into the trains (trucks cannot overload the trains). The solution is obviously $y = 0.95$ and $x = 0.19$ with objective value 1,900. We also calculated the value of ASP for all three constraints:

- ASP for g_1 is 1: by adding one unit to the first constraint ($2000y - 100x \leq 0$ becomes $2000y - 100x \leq 1$) the objective value increases by 1,
- ASP for g_2 is 100: by adding 1 unit to the second constraint ($x \leq 19$ becomes $x \leq 20$) the objective value increases by 100,
- ASP for g_3 is 0: by adding 1 unit to the second constraint ($y \leq 2$ becomes $y \leq 3$) the objective value does not increase.

Accordingly, the first and second constraints are bottlenecks as their corresponding ASPs are positive. Thus, it would be beneficial if investments are concentrated on adding one unit to the first or second constraint to improve the objective value.

[5] We have made several such industry-inspired stories and benchmarks available: http://cs.adelaide.edu.au/~optlog/research/bottleneck-stories.htm.

Adding one unit to the first constraint is meaningless from the practical point of view. In fact, adding one unit to RHS of the constraint g_1 means that the amount of products that is loaded into the trains can exceed the trains' capacities by one ton, which is not justifiable. In the above example, there is another option for the decision maker to achieve a better solution: if it is possible to improve the operation rate of the trucks to 101 tph, the best achievable solution is improved to 1,919 tons. Thus, it is clear that the bottleneck might be a specification of a resource (the operation rate of trucks in our example) that is expressed by a value in the coefficients matrix and not necessarily RHS.

Thus, it is clear that ASP only gives information about the impact of changing RHS in a constraint, while the bottleneck might be a value in the coefficient matrix. The commonly used ASP, which only gives information about the impact of changing RHS in a constraint, cannot identify such bottlenecks. Figure 3 illustrates this limitation.

The value of ASP represents only the effects of changing the value of RHS of the constraints (Fig. 3, left) on the objective value while it does not give any information about the effects the values in the coefficients matrix might have on the objective value (constraint g_1 in Fig. 3, right). However, as we are show in our example, it is possible to change the values in the coefficient matrix to make investments in order to remove bottlenecks.

The value of ASP does not provide any information about the best strategy of selecting bottlenecks to remove. In fact, it only provides information about the benefit of elevating the RHS in each constraint and does not say anything about the order of significance of the bottlenecks. It remains the task of the decision maker to compare different scenarios (also known as *what-if* analysis). For example, from a managerial point of view, it is important to answer the following question: is adding one unit to the first constraint (if possible) better than adding one unit to the second constraint (purchase a new truck)? Note that in real-world problems, there might be many

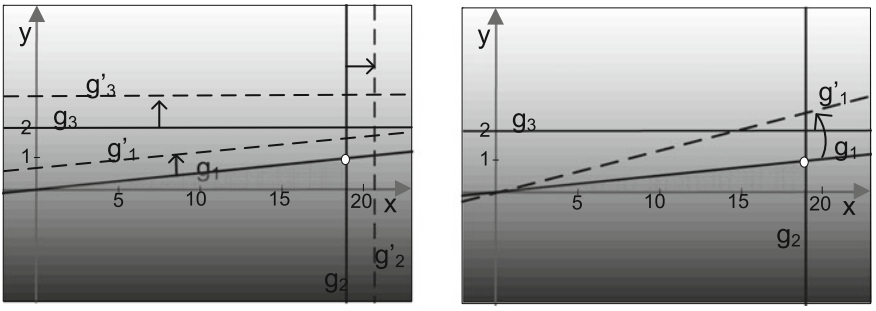

Fig. 3 x and y are number of trucks and number of trains respectively, *gray gradient* indication of objective value (the lighter the better), *shaded area* feasible area, g_1, g_2, g_3 are constraints, the white point is the best feasible point

resources and constraints, and a manual analysis of different scenarios might be prohibitively time consuming. Thus, a smart strategy is needed to find the best set of to-be-removed bottlenecks in order to gain maximum profit with lowest investment. In summary, the limitations of identifying bottlenecks using ASP are:

- **Limitation 1**: ASP is only applicable if objective and constraints are linear.
- **Limitation 2**: ASP does not evaluate changes in the coefficients matrix (the matrix A) and it is only limited to RHS.
- **Limitation 3**: ASP does not provide information about the strategy for investment in resources, and the decision maker has to manually conduct analyses to find the best investment strategy.

In order to resolve the limitations of ASP we proposed a new definition for bottlenecks and a new formulation for investment [11]. We defined bottlenecks as follows: *A bottleneck is a modifiable specification of resources that by changing its value, the best achievable performance of the system is improved*. Note that this definition is a generalization of the definition of bottleneck in [13]: a set of constraints with positive average shadow price is defined as a bottleneck. In fact, the definition in [13] concentrated on RHS only (it is just about the average shadow price) and it considers a bottleneck as a set of constraints. Conversely, our definition is based on any modifiable coefficient in the constraints (from capacity, to rates, or availability) and it introduces each specification of resources as a potential bottleneck.

Also, in order to determine the best possible investment to a system, we defined a Bottleneck COP (BCOP) for any COP as follows:

$$\text{find } x \text{ and } l \text{ s.t. } z = \begin{cases} \max f(x, l) \\ \min B(l) \end{cases} \text{ subject to } g_i(x, l_i) \leq 0 \text{ for all } i \tag{10}$$

where l is a vector (l might contain continuous or discrete values) which contains l_i for all i and $B(l)$ is a function that calculates the cost of modified specifications of resources coded in the vector l. For any COP, we can define a corresponding BCOP and by solving the BCOP, the plan for investment is determined.

The identification of bottlenecks and their removal are important topics in real-world optimization. As it was mentioned earlier, locating bottlenecks and finding the best possible investment is of a great importance in large industries. For example, in the mining process described in Sect. 2 not only the number of trucks, trains, or other resources can constitute a bottleneck, but also the operation rate of any of these resources can also constitute a bottleneck. Given the expenses for removing any of these bottlenecks, one can use the model in Eq. 10 to identify the best way of investment to grow the operations and make the most benefit. This area has remained untouched by the EC community, while there are many opportunities to apply EC-based methodologies to deal with bottlenecks and investments.

6 Discussion and Future Directions

Clearly, all three research directions (multi-component problems, edge of feasibility, and bottlenecks and investment) are relevant for solving real-world problems.

First, as it was mentioned earlier, an optimal solution for each component does not guarantee global optimality, so that a solution that represents the global optimum does not necessarily contain good schedules for each component in isolation [36]. The reason lies on the dependency among components. In fact, because of dependency, even if the best solvers for each component are designed and applied to solve each component in isolation, it is not useful in many real-world cases—the whole problem with dependency should be treated without decomposition of the components. Note that, decomposing problems that are not dependent on each other can be actually valuable as it makes the problem easier to solve. However, this decomposition should be done carefully to keep the problem unchanged. Of course complexity of decomposing multi-component problems is related to the components dependencies. For example, one can define a simple dependency between KP and TSP in a TTP problem that makes the problems decomposable or make them tighten together so that they are not easily decomposable.

Looking at dependencies among components, the lack of abstract problems that reflect this characteristic is obvious in the current benchmarks. In fact, real-world supply chain optimization problems are a combination of many smaller sub-problems dependent on each other in a network while benchmark problems are singular. Because global optimality is in interest in multi-component problems, singular benchmark problems cannot assess quality of methods which are going to be used for multi-component real-world problems with the presence of dependency.

Multi-component problems pose new challenges for the theoretical investigations of evolutionary computation methods. The computational complexity analysis of evolutionary computation is playing a major role in this field [2, 42]. Results have been obtained for many NP-hard combinatorial optimization problems from the areas of covering, cutting, scheduling, and packing. We expect that the computational complexity analysis can provide new rigorous insights into the interactions between different components of multi-component problems. As an example, we consider again the TTP problem. Computational complexity results for the two underlying problems (KP and TSP) have been obtained in recent years. Building on these results, the computational complexity analysis can help to understand when the interactions between KP and TSP make the optimization process harder.

Second, there has been some experimental evidence that showed the importance of searching the boundaries of feasible and infeasible areas in a constraint optimization problem (COP) [40, 49, 50]. This boundary is defined as: the points that are feasible and the value of *at least one* of the constraints is zero for them. In [5] three new instances (called Constraint Boundary Narrower, CBN, Subset CBN, SCBN, and All in a subset CBN, ACBN) for the constraint violation function were proposed which were able to reduce the feasible area to only boundaries of the feasible area. In the SCBN (ACBN), it is possible to select a subset of constraints and limit the boundaries

where *at least one* of these constraints (*all* of these constraints) is (are) active. The thickness of the boundaries was adjustable in the proposed method by a parameter (ϵ). Experiments showed that changing the value of ϵ influences the performance of the algorithm. In fact, a smaller value of ϵ causes limiting the feasible area to narrower boundaries, which makes finding the feasible areas harder. However, although it is harder to find the feasible areas (narrower boundaries), improving the final solutions is easier once the correct boundary was found. Thus, as a potential future work, one can design an adaptive method so that the search begins by exploring the feasible area and later concentrates on the boundaries.

Finally, a new definition for bottlenecks and a new model to guide decision makers to make the most profitable investment on their system should assist in narrowing the gap between what is being considered in academia and industry. Our definition for bottlenecks and model for investment overcomes several of the drawbacks of the model that is based on average shadow prices:

- It can work with non-linear constraints and objectives.
- It offers changes to the coefficient matrix.
- It can provide a guide towards optimal investments.

This more general model can form the basis for more comprehensive analytical tools as well as improved optimization algorithms. In particular for the latter application, we conjecture that nature-inspired approaches are adequate, due to the multi-objective formulation of the problem and its non-linearity.

Bottlenecks are ubiquitous and companies make significant efforts to eliminate them to the best extent possible. To the best of our knowledge, however, there seems to be very little published research on approaches to identify bottlenecks research on optimal investment strategies in the presence of bottlenecks seems to be even non-existent. In the future, we will push this research further, in order to improve decision support systems. If bottlenecks can be identified efficiently, then this information can be easily shown to the decision maker, who can then subsequently use this information in a manual optimization process.

There is also another research direction recently introduced to address real-world optimization problems that is locating disjoint feasible regions in a search space [4, 8].[6] It has been argued that the feasible area in constrained optimization problems might have an irregular shape and might contain many disjoint regions. Thus, it is beneficial if an optimization algorithm can locate these regions as much as possible so that the probability of finding the region that contain the best feasible solution is increased. The problem of locating many disjoint feasible regions can be viewed as niching in multi-modal optimization [4].

[6] we have excluded this topic from this chapter because of the lack of space.

References

1. Ackoff RL (1979) The future of operational research is past. J Oper Res Soc 53(3):93–104. ISSN 0160–5682
2. Auger A, Doerr B (2011) Theory of randomized search heuristics: foundations and recent developments, vol 1. World Scientific. ISBN 9814282669
3. Bertsimas D, Brown DB, Caramanis C (2011) Theory and applications of robust optimization. SIAM Rev 53(3):464–501. ISSN 0036–1445
4. Bonyadi MR, Michalewicz Z (2014) Locating potentially disjoint feasible regions of a search space with a particle swarm optimizer, book section to appear. Springer, New York
5. Bonyadi MR, Michalewicz Z (2014) On the edge of feasibility: a case study of the particle swarm optimizer. In: Congress on evolutionary computation, IEEE, pp 3059–3066
6. Bonyadi MR, Li X, Michalewicz Z (2013) A hybrid particle swarm with velocity mutation for constraint optimization problems. In: Genetic and evolutionary computation conference, ACM, pp 1–8. doi:10.1145/2463372.2463378
7. Bonyadi MR, Michalewicz Z, Barone L (2013) The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In: Congress on evolutionary computation, IEEE
8. Bonyadi MR, Li X, Michalewicz Z (2014) A hybrid particle swarm with a time-adaptive topology for constrained optimization. Swarm Evol Comput 18:22–37. doi:10.1016/j.swevo.2014.06.001
9. Bonyadi MR, Michalewicz Z, Neumann F, Wagner M (2014) Evolutionary computation for multi-component problems: opportunities and future directions. Frontiers in Robotics and AI, Computational Intelligence, under review, 2014
10. Bonyadi MR, Michalewicz Z, Przybyek MR, Wierzbicki A (2014) Socially inspired algorithms for the travelling thief problem. In: Genetic and evolutionary computation conference (GECCO), ACM
11. Bonyadi MR, Michalewicz Z, Wagner M (2014) Beyond the edge of feasibility: analysis of bottlenecks. In: International conference on simulated evolution and learning (SEAL), volume To appear, Springer
12. Charnes A, Cooper WW (1957) Management models and industrial applications of linear programming. Manag Sci 4(1):38–91. ISSN 0025–1909
13. Chatterjee A, Mukherjee S (2006) Unified concept of bottleneck. Report, Indian Institute of Management Ahmedabad, Research and Publication Department
14. Cho S, Kim S (1992) Average shadow prices in mathematical programming. J Optim Theory Appl 74(1):57–74
15. Crema A (1995) Average shadow price in a mixed integer linear programming problem. Eur J Oper Res 85(3):625–635. ISSN 0377–2217
16. Frieze A (1975) Bottleneck linear programming. Oper Res Q 26(4):871–874
17. Goldratt EM (1990) Theory of constraints. North River, Croton-on-Hudson
18. Goldratt EM, Cox J (1993) The goal: a process of ongoing improvement. Gower, Aldershot
19. Heywood MI, Lichodzijewski P (2010) Symbiogenesis as a mechanism for building complex adaptive systems: a review. In: Applications of evolutionary computation, Springer, pp 51–60
20. Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Phys D: Nonlinear Phenom 42(1):228–234. ISSN 0167–2789
21. Jacob Stolk AMZM, Mann I (2013) Combining vehicle routing and packing for optimal delivery schedules of water tanks. OR Insight 26(3):167190. doi:10.1057/ori.2013.1
22. Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. IEEE Trans Evol Comput 9(3):303–317. ISSN 1089–778X
23. Keane A (1994) Genetic algoritm digest. ftp://ftp.cse.msu.edu/pub/GA/gadigest/v8n16.txt
24. Keen PG (1981) Value analysis: justifying decision support systems. MIS Q 5:1–15. ISSN 0276–7783
25. Kim S, Cho S-C (1988) A shadow price in integer programming for management decision. Eur J Oper Res 37(3):328–335. ISSN 0377–2217

26. Koopmans TC (1977) Concepts of optimality and their uses. Am Econ Rev 67:261–274. ISSN 0002–8282
27. Lau HC, Song Y (2002) Combining two heuristics to solve a supply chain optimization problem. Eur Conf Artif Intell 15:581–585
28. Leguizamon G, Coello CAC (2009) Boundary search for constrained numerical optimization problems with an algorithm inspired by the ant colony metaphor. IEEE Trans Evol Comput 13(2):350–368. ISSN 1089–778X
29. Li X, Bonyadi MR, Michalewicz Z, Barone L (2013) Solving a real-world wheat blending problem using a hybrid evolutionary algorithm. In: Congress on evolutionary computation, IEEE, pp 2665–2671. ISBN 1479904538
30. Luebbe R, Finch B (1992) Theory of constraints and linear programming: a comparison. Int J Prod Res 30(6):1471–1478. ISSN 0020–7543
31. Maksud Ibrahimov SSZM, Mohais A (2012) Evolutionary approaches for supply chain optimisation part 1. Int J Intell Comput Cybern 5(4):444–472
32. Maksud Ibrahimov SSZM, Mohais A (2012) Evolutionary approaches for supply chain optimisation part 2. Int J Intell Comput Cybern 5(4):473–499
33. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, Chichester
34. Mersmann O, Bischl B, Trautmann H, Wagner M, Bossek J, Neumann F (2013) A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Ann Math Artif Intell 1–32. ISSN 1012–2443
35. Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs. Springer. ISBN 3540606769
36. Michalewicz Z (2012) Quo vadis, evolutionary computation? Adv Comput Intell 98–121
37. Michalewicz Z (2012) Ubiquity symposium: evolutionary computation and the processes of life: the emperor is naked: evolutionary algorithms for real-world applications. Ubiquity, 2012(November):3
38. Michalewicz Z, Fogel D (2004) How to solve it: modern heuristics. Springer, New York. ISBN 3540224947
39. Michalewicz Z, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4(1):1–32. ISSN 1063–6560
40. Michalewicz Z, Nazhiyath G, Michalewicz M (1996) A note on usefulness of geometrical crossover for numerical optimization problems. In: Fifth annual conference on evolutionary programming, Citeseer, p 305312
41. Nallaperuma S, Wagner M, Neumann F, Bischl B, Mersmann O, Trautmann H (2013) A feature-based comparison of local search and the christofides algorithm for the travelling salesperson problem. In: Proceedings of the twelfth workshop on foundations of genetic algorithms XII, ACM, pp 147–160. ISBN 1450319904
42. Neumann F, Witt C (2012) Bioinspired computation in combinatorial optimization: algorithms and their computational complexity. In: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion, ACM, pp 1035–1058. ISBN 1450311784
43. Nguyen T, Yao X (2012) Continuous dynamic constrained optimisation-the challenges. IEEE Trans Evol Comput 16(6):769–786. ISSN 1089–778X
44. Polyakovskiy S, Bonyadi MR, Wagner M, Michalewicz Z, Neumann F (2014) A comprehensive benchmark set and heuristics for the travelling thief problem. In: Genetic and evolutionary computation conference (GECCO), ACM. ISBN 978-1-4503-2662-9/14/07. doi:10.1145/2576768.2598249
45. Potter M, De Jong K (1994) A cooperative coevolutionary approach to function optimization. In: Parallel problem solving from nature, Springer, Berlin Heidelberg, pp 249–257. doi:10.1007/3-540-58484-6269
46. Rahman S-U (1998) Theory of constraints: a review of the philosophy and its applications. Int J Oper Prod Manage 18(4):336–355. ISSN 0144–3577

47. Rosin CD, Belew RK (1995) Methods for competitive co-evolution: finding opponents worth beating. In: ICGA, pp 373–381
48. Runarsson T, Yao X (2000) Stochastic ranking for constrained evolutionary optimization. IEEE Trans Evol Comput 4(3):284–294. ISSN 1089–778X
49. Schoenauer M, Michalewicz Z (1996) Evolutionary computation at the edge of feasibility. In: Parallel problem solving from nature PPSN IV, pp 245–254
50. Schoenauer M, Michalewicz Z (1997) Boundary operators for constrained parameter optimization problems. In: ICGA, pp 322–32
51. Smith-Miles K, van Hemert J, Lim XY (2010) Understanding TSP difficulty by learning from evolved instances, Springer, pp 266–280. ISBN 3642137997
52. Smith-Miles K, Baatar D, Wreford B, Lewis R (2014) Towards objective measures of algorithm performance across instance space. Comput Oper Res 45:12–24. ISSN 0305–0548
53. Weise T, Zapf M, Chiong R, Nebro A (2009) Why is optimization difficult? Nature-inspired algorithms for optimisation, pp 1–50
54. Wu ZY, Simpson AR (2002) A self-adaptive boundary search genetic algorithm and its application to water distribution systems. J Hydraul Res 40(2):191–203. ISSN 0022–1686