Evolutionary computation for multi-component problems: Opportunities and future directions

Mohammad Reza Bonyadi
Optimisation and logistics group
The University of Adelaide
mrbonyadi@cs.adelaide.edu.au

Zbigniew Michalewicz
Optimisation and logistics group
The University of Adelaide
zbyszek@cs.adelaide.edu.au

Frank Neumann
Optimisation and logistics group
The University of Adelaide
frank.neumann@adelaide.edu.au

Markus Wagner
Optimisation and logistics group
The University of Adelaide
markus.wagner@adelaide.edu.au

1 Motivation

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 fail [1]. 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 [2] 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 [3], 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 (e.g. knapsack problems, KP, with 2,500 items and traveling salesman problems, TSP, with more than 10,000 cities). Noisy environments have been already defined [4-6] in the field of optimization, in both continuous and combinatorial optimization domain (mainly from the operations research field), see [4] 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 (e.g. stochastic TSP, stochastic vehicle routing problem, VRP), see [6] for performance evaluation of evolutionary algorithms when the objective function is noisy. Recently, some challenges in dealing with continuous space optimization problems with noisy constraints were discussed and some benchmarks were designed [5]. Presence of constraints has also been captured in benchmark problems where one can generate different problems with different constraints (e.g. Constrained VRP, CVRP). Thus, the expectation is, after capturing all of these pitfalls and addressing them (or 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), dedicated journals (e.g. Evolutionary Computation

[1] By real-world problems we mean problems which are found in some business/industry on daily (regular) basis. See [1] For details on different interpretations of the term "real-world problems".
[2] Available at: http://people.brunel.ac.uk/~mastjjb/jeb/info.html

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 in industries.

As discussed in [7], 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 reasons include:

a. Experiments focus on single component (also known as single *silo*) benchmark problems
b. Experiments focus on global optima
c. Theory does not support practice
d. General dislike of business issues in research community
e. Limitations of EA-based companies
f. Dominance of Operation Research methods in industry

It seems that the first reason (most of the experimental research efforts have been directed towards single-component problems such as TSP and KP) is of the key importance – and all other reasons (b) – (f) are just consequences of (a). Note that 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). In fact, problems in real-world 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 already over 30 years ago by Operations Research (OR) community; in [8] 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 available. 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 company's internal processes to complex software engineering issues.

Let us illustrate differences between single-component problems and multi-component problems by presenting a puzzle[3]. There is a ball that has been cut into two parts in a special way (see Fig. 1a), and a cube with a hole inside, that has been also cut into two parts in a special way (see figure Fig. 1b). The two parts of the ball can be easily set up together to make a complete ball (see Fig. 1a). Also, the two parts of the cube can be put easily together to shape the cube (see Fig. 1a). The size of the hole inside the cube is slightly larger than the size of the ball, so that, if the ball is inside the hole it can spin freely. However, it is not possible to set up the cube and then put the ball inside the cube as the entry of the hole of the cube is smaller than the size of the ball (see Fig. 1c). Now, the puzzle is stated as follows: set up the cube with the ball inside (see figure Fig. 1d). Setting up the ball separately and the cube separately is easy. However, setting up the cube while the ball is set up inside the cube is extremely hard[4].

[3] The name of this puzzle is the "Cast Marble", created by Hanayama company.
[4] The difficulty level of this puzzle was reported as 4 out of 6 by the Hanayama website, that is equal to the difficulty of Rubik's cube.

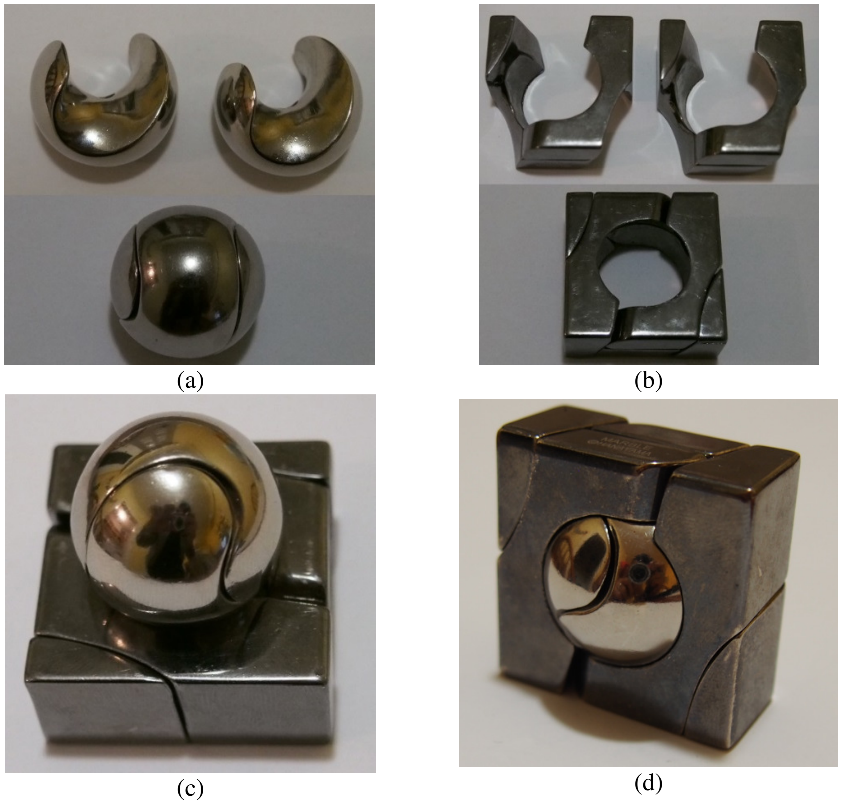


Fig. 1 The Cast Marble puzzle: (a) two pieces of a ball and the ball that is generated by setting up the pieces, (b) two pieces of a cube and the cube that is generated by setting up the pieces, (c) the ball is not fit in the hole of the cube if the cube is set up first, and (d) the solution of the puzzle.

This puzzle nicely represents the difference between single-component and multi-component problems. In fact, solving a single component problem (setting up the ball or the cube separately) might be easy; however, solving the combination of two simple component problems (setting up the ball while it is inside the cube) is potentially extremely harder.

In this paper we explore this issue further – and we organized the paper as follows. In section 2 two real-world examples are explained, in section 3 some important observations about real-world problems are discussed, in section 4 a recently presented benchmark multi-component problem is introduced and discussed, and in section 5 some discussions and directions for future research are provided.

2 Two examples

The first example relates to optimization of the transportation of water tanks [9]. 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 one 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 (it is called *subset selection* procedure). Because the tanks are empty and of different sizes they might be packed inside each other (it is called *bundling* procedure) to maximize truck's 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 (it is called *base selection* procedure) 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 carries 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 (it is called *delivery routing* procedure).

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 and to determine an exact routing for bases and stations for unbundling and delivery activities – to maximize the total “value” of the delivery at the end of the time period. This total value 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 to optimality (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 fist in a truck. Also, by selecting tanks without considering the location of stations and bases, the best achievable solutions might not be a very quality one, 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, maintenance schedule of the trucks.

The second example relates to optimizing supply-chain operations of a mining company: from mines to ports [10, 11]. 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 ship’s demand. This is the task of the mine owner to provide enough products of each type in the stockyard by the time that the ships arrive. 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 (this is called *train scheduling* procedure). Also, scheduling train dumpers to unload the trains and put the unloaded products in appropriate stockpiles (this is called *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 (this is called *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 soon in the time schedule. 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 make them move too much 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.

3 Lessons Learned – Dependencies

Let us take a closer look at the examples presented in section 2. Obviously, both optimization problems contain constraints and noise; both of them might be large in terms of the number of decision variables. However, there is another characteristic present in both problems: each problem is a *combination* of several sub-problems (components/silos). The tank delivery problem is a combination of tank selection, delivery routing, base selection, and bundling. Also, the mine to port problem is a combination of train scheduling, unload scheduling, load scheduling, and berthing. Because these problems are a combination of some components, we call them *multi-component* problems. Of course each component is hard to solve in isolation, however, as it was mentioned, solving each component even to optimality does not necessarily direct the search towards good overall solutions if other components are not considered. In fact, solutions for each sub-problem affect the environment (constraints or achievable quality) of some other sub-problems because of the *dependencies* among components. As an example, in the tank delivery problem, delivery routing (best route to deliver tanks) is affected by the base selection (best choice for the base to unbundle the tanks) as choosing different bases imposes different lengths of travels between the base and stations. Delivery routing is also affected by the tank selection (selecting the tanks for delivery) as the selected subset of tanks determines the stations to visit. Dependency is also found in the mine to port problem. As an example, unload scheduling (scheduling the train dumpers to unload the trains) affects the loader scheduling (scheduling the loaders to take products from stockpiles and load them to the ship) in such a way that loading the products in different stockpiles results in different arrangement of the products which impacts the ship loading activities (it might make the loader schedule to move the ship loaders a lot to load each ship). In short, a solution for each component actually affects the feasibility or the best achievable solution in other components.

Dependency causes appearance of some other features in real-world problems. As an example, the dependency among components in a problem sometimes causes a special *flow of data* among components of that problem. For instance, generating a solution (independent from the quality of the solution) for the unloading schedule is impossible without being aware of the train schedule because assignment of dumpers depends on the arrival time of trains. Also, generating a solution for the delivery routing in the tank delivery problem is impossible without being aware of the selected subset of tanks.

Definition 1: if generating a solution (independent from the quality) for a component A is impossible because of some data that needs to be provided by the component B, we say that A is the data follower of B.

This indeed imposes a special sequence of solution procedure for some components that need to be taken into account by the solver.

Also, dependency among components makes *modeling* (mathematical modeling) of problems more complex. In fact, modeling of existing benchmark problems (such as TSP, multidimensional KP, job shop scheduling, etc) is relatively easy and many different models for them already exist. However, a multi-component problem involves a more complex model, even if it has been composed of well-modeled components. The main reason is that in a multi-component problem a constraint that is in one of the components may influence feasibility of other components because of dependency. Thus, modeling each component in isolation and putting these models together does not express the model for the whole problem.

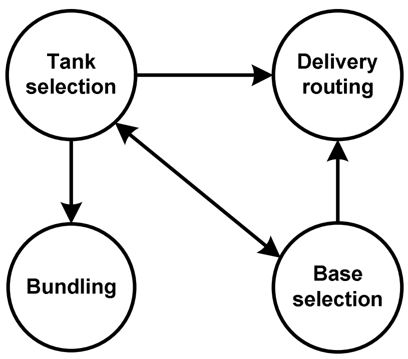
Another effect that is caused by dependency in multi-component problems is the *propagation of noise*. Noise in benchmark problems is normally defined by a stochastic function like a normal or Poisson distribution that is simply added to the objective function or constraints. However, with the presence of dependency noise is propagated from each component to the others. Because dependency between components might follow a complex function, the propagated noise from one component to the others might also become complex (even if the original noise was based on a simple distribution) which causes difficulties in solving the problem. As an example, if the break-down distribution of the dumpers in the mine to port problem is Poisson with some parameters and this break-down has some other distribution for the ship loaders, the effect of these noises on the objective function cannot be treated as a simple Poisson and it is actually hard to even estimate. Note also that investigation of noise over the whole system results in a better risk analysis in the whole operation and a better estimation tolerance in the final benefit.

An additional interesting feature that comes into play because of dependency is the concept of *bottleneck component*. A bottleneck component is a component in the whole system that constraints the best overall achievable solution. In fact, if there is a bottleneck component in the system, adding more resources to other components does not cause improvement of the best achievable objective value. As an example, in the mine to port problem if the bottleneck component is the number of trains then investment in *any* other parts of the system (expanding the stockyard area, having more loaders, more dumpers, or more trucks) has minimal affect (or even no effect) on the best overall achievable solution in the system.

Dependency is the source of some complexities in multi-component problems. Thus, a proper definition for dependency among components is of importance. The concept of dependency among components and effects of the components on each other is similar to the concept of *separability* of variables in continuous space optimization problems [12]. As it was mentioned earlier, dependency stems from the effects of components on each other, i.e., changing the solution of a component affects feasibility or quality of solutions of other components. Accordingly, we suggest the following definition for dependency among components:

Definition 2: We say component B is dependent on component A (notation: A->B) if
(1) A is not data follower of B (see definition 1),
and
(2) changing the solutions of component A can change the best achievable solution for the component B in terms of the overall objective value.

The part (1) of the definition prevents introduction of dependency between two components, A and B, where A needs a flow of data from B. Note that, if A is a data follower of B then B cannot be dependent to A. Assessing the first part of the definition is not a hard task. As an example, generating a solution for the train scheduling in the mine to port problem is possible without being aware of unload schedule. However, it is impossible to generate a solution for the unload schedule without knowing the solution for the train schedule. The part (2) of the definition ensures that the components are not separable. To assess if B is dependent to A, assume that there exist a solution a for the component A and, given A is fixed to a (showing by $A=a$), setting $B=b$ results in the best possible overall objective value. Now, if there exists an a' that for $A=a'$, $B = b' \neq b$ results in the best overall objective value, then A->B (B is dependent to A). This means that, changing the solution for A actually might change the best solution for B. Dependency is shown in a diagram for the example problems (see Fig. 2).



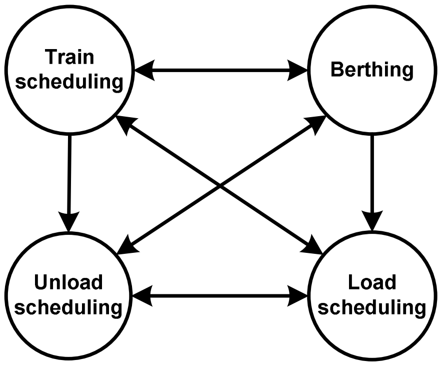


Fig. 2 Diagram of dependency among components of (a) tank delivery problem, and (b) mine to port problem.

Fig. 2 shows the diagram of dependency among components of the mine to port problem and the tank delivery problem. The links in the figure refer to a dependency among different components. As an example, one can fix the solution for the base selection (of course with being aware of the solution for the subset selection) without being aware of the delivery routing solution. Also, by changing the base, the best achievable solution (shortest tours) for delivery routing is changed. Thus, the base selection is linked to delivery routing. Note that generating a solution for delivery routing is impossible without being aware of the location of the base, hence, there is no link from delivery routing to the base selection.

Hypothetically, any dependency can exist between a set of problems. These dependencies can be represented by a digraph, which can potentially form a complex network. In the simplest case, there are some problems with no dependencies. In this case, one can solve each problem separately to optimality and combine them to get the global optimal solution.

4 Traveling Thief Problem (TTP)

Recently a new benchmark problem called traveling thief problem (TTP) was introduced [13] 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 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 $v: \mathscr{R} \rightarrow \mathscr{R}$ 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.

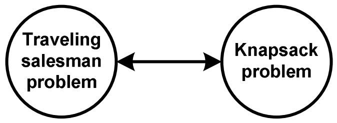


Fig. 3 TTP dependency graph

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 [14].

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 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, that implies that 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.

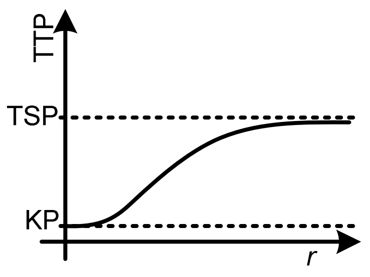


Fig. 4 Impact of the rent rate *r* on the TTP. For *r=0*, the TTP solution is equivalent to the solution of KP, while for larger *r* the TTP solutions become more closer to the solutions of TSP

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).

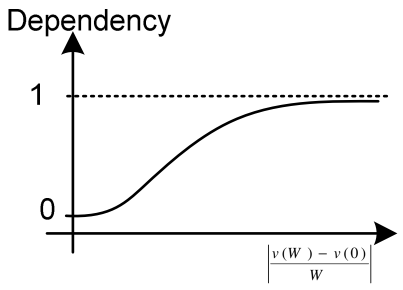


Fig. 5 How dependency between components is affected by speed (function *v*). When *v* does not drop significantly for different weights of picked items ($\left|\frac{v(W)-v(0)}{W}\right|$ is small), the two problems can be decomposed and solved separately.

The value Dependency=1 represents the two components are dependent while Dependency=0 shows that two components are not dependent.

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 to do not pick any item (the solution for KP is to pick no item) and only solve the TSP part as efficient as possible. This has been also discussed in [15].

5 Discussion and future directions

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 [1]. 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. As an example, if in the cast marble puzzle the final objective is to set up the ball on top of the cube then the puzzle was extremely easy (one could set up the ball and the cube separately and put the ball on the cube). In contrast, setting up the ball inside the cube is extremely hard as the ball and the cube need to be set up simultaneously one inside another. As another 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 [16, 17]. 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.

In a similar way, 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 [18]. Classical algorithms for the TSP and their success depending on features of the given input have been studied in [19-21] 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 [22]. 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.

For 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 so often 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 [23], 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 none of both setups, the optimization on the involved components happens 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 [24]. 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 two types: competitive and cooperative. 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 [25], where he applied a competitive predator-prey model to for the evolution of sorting networks. Rosin and Belew [26] 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 specie 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.

It seems multi-component problems provide great opportunity for further research in EC community. Thus, we believe that future research in this direction can potentially close the gap between academic research in EC community and needs for optimization methodologies in industries.

6 References

[1] Z. Michalewicz, "Quo Vadis, Evolutionary Computation?," *Advances in Computational Intelligence,* pp. 98-121, 2012.

[2] T. Weise, M. Zapf, R. Chiong, and A. Nebro, "Why is optimization difficult?," *Nature-Inspired Algorithms for Optimisation,* pp. 1-50, 2009.

[3] Z. Michalewicz and D. B. Fogel, *How to solve it: modern heuristics*: Springer-Verlag New York Inc, 2004.

[4] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM review,* vol. 53, no. 3, pp. 464-501, 2011.

[5] T. Nguyen and X. Yao, "Continuous dynamic constrained optimisation-The challenges," *IEEE Transactions on Evolutionary Computation,* vol. 16, no. 6, pp. 769-786, 2012.

[6] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on Evolutionary Computation,* vol. 9, no. 3, pp. 303-317, 2005.

[7] Z. Michalewicz, "Ubiquity symposium: Evolutionary computation and the processes of life: the emperor is naked: evolutionary algorithms for real-world applications," *Ubiquity,* vol. 2012, no. November, p. 3, 2012.

[8] R. L. Ackoff, "The future of operational research is past," *Journal of the operational research society,* pp. 93-104, 1979.

[9] I. M. Jacob Stolk, Arvind Mohais, Zbigniew Michalewicz, "Combining Vehicle Routing and Packing for Optimal Delivery Schedules of Water Tanks," *OR Insight,* vol. 26, no. 3, pp. 167–190, 2013, doi:10.1057/ori.2013.1.

[10] A. M. Maksud Ibrahimov, Sven Schellenberg, Zbigniew Michalewicz "Evolutionary Approaches for Supply Chain Optimisation – Part 1," *International Journal of Intelligent Computing and Cybernetics,* vol. 5, no. 4, pp. 444-472, 2012.

[11] A. M. Maksud Ibrahimov, Sven Schellenberg, Zbigniew Michalewicz "Evolutionary Approaches for Supply Chain Optimisation – Part 2," *International Journal of Intelligent Computing and Cybernetics,* vol. 5, no. 4, pp. 473 – 499, 2012.

[12] D. Whitley, S. Rana, and R. B. Heckendorn, "Island model genetic algorithms and linearly separable problems," in *Evolutionary computing*, ed: Springer, 1997, pp. 109-125.

[13] M. R. Bonyadi, Z. Michalewicz, and L. Barone, "The travelling thief problem: the first step in the transition from theoretical problems to realistic problems," in *Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico*, 2013,

[14] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann, "A Comprehensive Benchmark Set and Heuristics for the Travelling Thief Problem," in *Genetic and Evolutionary Computation Conference (GECCO)*, Vancouver, BC, Canada, 2014, doi:http://dx.doi.org/10.1145/2576768.2598249

[15] M. R. Bonyadi, Z. Michalewicz, M. R. Przybyłek, and A. Wierzbicki, "Socially Inspired Algorithms for the Travelling Thief Problem," in *Genetic and Evolutionary Computation Conference (GECCO)*, Vancouver, BC, Canada, 2014, doi:http://dx.doi.org/10.1145/2576768.2598367

[16] A. Auger and B. Doerr, *Theory of Randomized Search Heuristics: Foundations and Recent Developments* vol. 1: World Scientific, 2011.

[17] F. Neumann and C. Witt, "Bioinspired computation in combinatorial optimization: algorithms and their computational complexity," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, 2012, pp. 1035-1058,

[18] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, "Towards objective measures of algorithm performance across instance space," *Computers & Operations Research,* vol. 45, pp. 12-24, 2014.

[19] K. Smith-Miles, J. van Hemert, and X. Y. Lim, "Understanding TSP difficulty by learning from evolved instances," in *Learning and intelligent optimization*, ed: Springer, 2010, pp. 266-280.

[20] S. Nallaperuma, M. Wagner, F. Neumann, B. Bischl, O. Mersmann, and H. Trautmann, "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*, 2013, pp. 147-160,

[21] O. Mersmann, B. Bischl, H. Trautmann, M. Wagner, J. Bossek, and F. Neumann, "A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem," *Annals of Mathematics and Artificial Intelligence,* pp. 1-32, 2013.

[22] S. Martello and P. Toth, "Knapsack problems: algorithms and computer implementations," *John Wiley&Sons, Chichester, UK,* 1990.

[23] H. C. Lau and Y. Song, "Combining Two Heuristics to Solve a Supply Chain Optimization Problem," in *European Conference on Artificial Intelligence*, France, 2002, pp. 581-585,

[24] M. Potter and K. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature*, 1994, pp. 249-257, doi:10.1007/3-540-58484-6_269.

[25] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D: Nonlinear Phenomena,* vol. 42, no. 1, pp. 228-234, 1990.

[26] C. D. Rosin and R. K. Belew, "Methods for Competitive Co-Evolution: Finding Opponents Worth Beating," in *ICGA*, 1995, pp. 373-381,