On the Design of Metaheuristics-Based Algorithm Portfolios

Dimitris Souravlias and Konstantinos E. Parsopoulos

Abstract Metaheuristic optimization has been long established as a promising alternative to classical optimization approaches. However, the selection of a specific metaheuristic algorithm for solving a given problem constitutes an impactful decision. This can be attributed to possible performance fluctuations of the metaheuristic during its application either on a single problem or on different instances of a specific problem type. Algorithm portfolios offer an alternative where, instead of using a single solver, a number of different solvers or variants of one solver are concurrently or interchangeably used to tackle the problem at hand by sharing the available computational resources. The design of algorithm portfolios requires a number of decisions from the practitioner's side. The present chapter exposes the essential open problems related to the design of algorithm portfolios, namely the selection of constituent algorithms, resource allocation schemes, interaction among the algorithms, and parallelism issues. Recent research trends relevant to these issues are presented, offering motivation for further elaboration.

Keywords Algorithm portfolios · Metaheuristics · Global optimization · Design of algorithms

1 Introduction

A central issue in applied optimization is the selection of a suitable solver for a given problem. Metaheuristics have proved to be very useful under various conditions when the approximation of (sub-) optimal solutions is desirable. Despite the large

D. Souravlias (✉)
Logistics Management Department, Helmut-Schmidt University, Hamburg, Germany
e-mail: dsouravl@hsu-hh.de

K. E. Parsopoulos
Department of Computer Science & Engineering, University of Ioannina, Ioannina, Greece
e-mail: kostasp@cse.uoi.gr

P. M. Pardalos, A. Migdalas (eds.), *Open Problems in Optimization and Data Analysis*, Springer Optimization and Its Applications 141,
https://doi.org/10.1007/978-3-319-99142-9_14

number of metaheuristics in the relevant literature, choosing one for the problem at hand is a crucial decision that is often based on the practitioner's experience and knowledge on the specific problem type.

Strong theoretical and experimental evidence suggest that there is no universal optimization algorithm capable of tackling all problems equally well [17]. Thus, relevant research has been focused on matching efficient algorithms with specific problem types. However, even different parameterizations of an algorithm may exhibit highly varying performance on a given problem. In this context, the idea of concurrently or interchangeably using a number of different algorithms or variants of a single algorithm was cultivated in order to reduce the risk of a wrong decision. Inspiration behind this idea was drawn from the financial markets, where selecting a portfolio of stocks instead of investing the whole capital on a single stock reduces the risk of the investment.

Algorithm portfolios (APs) are algorithmic schemes that harness a number of algorithms into a unified framework [5, 7]. They were initially introduced two decades ago and, since then, they have gained increasing popularity in various scientific fields such as inventory routing [11], lot sizing [12], facility location [4], and combinatorics [13]. The term "algorithm portfolio" was first proposed in [7] and dominated over previously used terms such as the *ensembles of algorithms* [6]. The algorithms employed in an algorithm portfolio are called the *constituent algorithms*, and they can be of different types ranging from evolutionary algorithms [9] and randomized search methods [5] to SAT solvers [18].

In the present chapter, we focus on algorithm portfolios consisting of metaheuristics, including both population-based and local-based algorithms. Population-based algorithms exploit a population of search points that iteratively probe the search space. On the other hand, local-based (or trajectory-based) methods operate on a single search point that is iteratively improved through various local search strategies. Thus, population-based algorithms possess essential exploration capabilities, whereas local-based algorithms are well known for their exploitation capacities. It is widely perceived that the trade-off between exploration and exploitation determines the quality of a metaheuristic approach. Therefore, a framework that encompasses both exploration-oriented and exploitation-oriented methods is expected to be more beneficial than using a single approach.

The design of algorithm portfolios involves a number of issues that shall be addressed by the practitioner. Specifically, the following decisions shall be made:

1. How many and which algorithms shall be included?
2. How are the computational resources allocated to the constituent algorithms?
3. Shall the algorithms interact among them and how?
4. How parallelism affects the operation of the algorithm portfolio?

In the following sections, we discuss these issues and review relevant research developments. The rest of the chapter is structured as follows: Section 2 provides a general description of algorithm portfolios. Various approaches on the selection of constituent algorithms are presented in Section 3.1, while resource allocation

schemes are discussed in Section 3.2. Section 3.3 is devoted to the interaction among algorithms, and Section 3.4 discusses the implications of parallelism. Finally, Section 4 concludes the chapter.

2 Algorithm Portfolios

The need for the use of metaheuristics stems from the inability of classical optimization algorithms to efficiently tackle optimization problems of various types either due to special characteristics of the objective function or due to excessive running time [3]. Such problems usually lack nice mathematical properties and frequently involve, among others, discontinuous, noisy, and computationally heavy objective functions, as well as multiple local and global minimizers.

Without loss of generality, we consider the unconstrained minimization problem,

$$\min_{x \in D} f(x)$$

where D stands for the corresponding search space. The type of D (e.g., continuous, discrete, etc.) is irrelevant since it affects only the type of relevant solvers that will be used. Let also

$$S = \{S_1, S_2, \ldots, S_k\},$$

be a set of available solvers suitable for the problem at hand. The set S may consist of different algorithms or different instances (parameterizations) of a single algorithm or both.

The design of an algorithm portfolio primarily requires the determination of its constituent algorithms [2]. Thus, an algorithm selection process takes places to distinguish n out of the k available solvers. The selection is typically based on performance profiles of the algorithms on the specific problem or similar ones. The outcome is a portfolio AP defined as

$$AP = \{S_{i_1}, S_{i_2}, \ldots, S_{i_n}\} \subseteq S.$$

For simplicity reasons, we will henceforth use the indices $1, 2, \ldots, n$, instead of $i_1, i_2, \ldots, i_n$, for the constituent algorithms.

Moreover, let $T_{\max}$ be the available computational budget. This may refer to the total execution time or the number of function evaluations that is available to the algorithm portfolio. Then, each solver S_i of AP is assigned a fraction T_i of this budget, taking care of that

$$\sum_{i=1}^{n} T_i = T_{\max}. \tag{1}$$

The set of the allocated budgets will be henceforth denoted as $T = \{T_1, T_2, \ldots, T_n\}$. The allocated budgets can be determined either offline, i.e., prior to the application of the algorithm portfolio, or online during its run. In both cases, performance data (historical or current, respectively) are used along with predefined performance measures.

During the run of the algorithm portfolio, the constituent algorithms may be either isolated [14] or interactive [12]. In the latter case, the algorithms exchange information and the user shall determine all the relevant details, including the type and size of the exchanged information, as well as the communication frequency or the conditions for triggering such a communication. Typically, the communication takes place between pairs of algorithms, and the same type of information is exchanged among them. For example, such information can be the best solution detected so far by each algorithm.

The constituent algorithms of an algorithm portfolio may sequentially and interchangeably run on a single processor. In this case, each constituent algorithm S_i is run in turn on the processor consuming a prespecified fraction of its assigned computational budget T_i before the next algorithm occupies the processor. Thus, the total budget assigned to each algorithm is not consumed in one turn but rather in multiple turns (called also *episodes* or *batches*).

Nevertheless, the use of multiple algorithms in an algorithm portfolio makes parallelism very appealing. In this case, a number of CPUs are available and devoted to AP. Let $U = \{U_1, U_2, \ldots, U_m\}$ be the set of available processing units. A common master–slave parallelism model usually employs a master-node, which is responsible for the coordination of the portfolio's run as well as the necessary book-keeping, and a number of slave-nodes that host the constituent algorithms. In the ideal case, the number of slave-nodes is equal to the number of constituent algorithms, i.e., $m = n + 1$. If this is not possible, CPUs host more than one of the constituent algorithms.

Our descriptions above have brought forward a number of issues and open problems that influence the application of algorithm portfolios. The aptness of the underlying choices determines the efficiency and effectiveness of the portfolio. A number of research works have probed these issues and derived useful conclusions and suggestions for various problem types. The rest of the present chapter offers an outline of the most important issues and suggestions.

3 Open Problems in Designing Algorithm Portfolios

In the following paragraphs, we present essential open problems in the design of algorithm portfolios. Specifically, we consider the following problems:

1. Selection of the constituent algorithms.
2. Allocation of the computational budget.
3. Interaction among the constituent algorithms.
4. Parallel vs sequential application of the algorithm portfolio.

Our main goal is to expose the corresponding problems and discuss some of the state-of-the-art remedies proposed in the relevant literature. This information can be used as motivation and starting point for further elaboration.

3.1 Selection of Constituent Algorithms

The selection of the constituent algorithms in an algorithm portfolio is highly related to the *algorithm selection problem* originally described by Rice [10]. This refers to the selection and application of the best-performing algorithm for a given optimization problem. Most commonly, algorithm selection is addressed for specific problem types or sets of problems. Given a predefined set of algorithms S, a set of problem instances I, and a cost function

$$\mathscr{F} : S \times I \rightarrow \mathbb{R},$$

the widely known *per instance algorithm selection problem* refers to the detection of a mapping $\mathscr{G} : I \rightarrow S$, such that the cost function $\mathscr{F}$ is minimized over all the considered problem instances, i.e.,

$$\min_{\mathscr{G}} \sum_{i \in I} \mathscr{F}\left(\mathscr{G}(i), i\right). \tag{2}$$

In the context of algorithm portfolios, this problem is referred as the *constituent algorithms selection problem* [2, 15]. Specifically, given a predefined set of algorithms S, a set of problem instances I, and a cost function $\mathscr{F}$, the goal is to select a set of algorithms $AP \subseteq S$, such that the cost function is minimized over all problem instances. This can be considered as a relaxed version of the problem defined in Equation (2) where, instead of mapping problem instances to algorithms, the main challenge is to design the best-performing algorithm portfolio over all instances, consisting of the selected algorithms $S_i \in S$.

The constituent algorithms selection problem has been primarily addressed by using offline selection methods. Such methods are applied prior to the execution of the algorithm portfolio. They are based on *a priori* knowledge on the average performance of each candidate algorithm over all problems at hand. If there is no adequate information on the algorithms, a preprocessing phase is usually needed to identify their relevant properties. However, this can be a time-consuming and laborious task in cases of hard real-life problems.

In [2] a general selection scheme was proposed to tackle the constituent algorithms selection problem. This scheme admits as input the number of all available algorithms and the targeted problems, and generates the set of constituent algorithms based on various selectors. The number of selected algorithms is either specified by the user or determined by the used selector. The evaluation of the selectors in [2] was conducted on a set of instances of the maintenance scheduling problem using random data generated by the same distribution.

Comparisons among different selectors revealed the superiority of the *selector delete distance* (SDI) and the *selector racing* (SRC) policies [2]. The SDI technique applies all the available algorithms on a declining list of problem instances. First, the order of the available instances is randomized and then each problem instance is individually solved by each algorithm in turn. The algorithm that solves the problem instance (i.e., the acquired objective function value is below a user-defined threshold) is selected as a constituent algorithm, while the problem instance is removed from the list. The process ends when all instances are solved by an algorithm. On the other hand, the SRC method uses the F-Race method to select algorithms. Specifically, each problem instance is randomly selected and tackled by all algorithms in turn. If the instance is solved once, it is not considered again. Algorithms that do not perform adequately well are discarded, based on statistical comparisons of their performance with the rest. This procedure is terminated when n constituent algorithms are selected.

A portfolio of population-based algorithms was considered in [15], consisting of evolutionary algorithms for numerical optimization problems. The portfolio is equipped with an online mechanism used to automatically select its constituent algorithms. The selection mechanism is allocated a portion of the available computational budget, and relies on the *estimated performance matrix* method applied on each candidate evolutionary algorithm. Specifically, each algorithm is applied k_1 times for each one of the available k_2 problem instances. Hence, for each algorithm S_i, a $k_1 \times k_2$ performance matrix PM_i is formed, where each component is the solution's function value detected in a single run of the algorithm on the specific problem instance. The matrices are eventually used to select the constituent algorithms of the portfolio, while the remaining computational budget is utilized to solve the problem at hand.

The role of the synergy of the constituent algorithms in the success of an algorithm portfolio has been recognized through theoretical and experimental evidence [9]. The use of portfolios instead of individual algorithms aims at alleviating possible weaknesses of one algorithm by using another one with complementary performance. For example, let $S_1, S_2 \in S$ be two algorithms, and I_1, I_2, I_3, I_4 be four test problems. Suppose that S_1 performs better than S_2 on instances I_1 and I_2, whereas S_2 is more successful than S_1 on instances I_3 and I_4. Including both algorithms into a portfolio seems to be a reasonable choice, as they complementarily perform on the considered instances. Thus, complementarity of the constituent algorithms in a portfolio arises as an important issue in the algorithm selection phase.

Moreover, complementarity is essential even during the run of the portfolio. For example, the use of both exploration-oriented (global search) algorithms that are very useful at the early stages of the optimization procedure and exploitation-oriented (local search) algorithms that conduct more refined search around the best detected solutions at the final stages of the optimization procedure can be highly beneficial. A combination of these two can prevent from the undesirable premature convergence problem. Recent experimental evidence on portfolios consisting of both population-based and local search algorithms justified the above claims on challenging manufacturing problems [12].

The identification of complementary algorithms is not a trivial task. To this end, a new technique for designing portfolios with algorithms that exhibit complementarity properties was introduced in [8]. This technique models the constituent algorithm selection problem as an election task, where each problem "votes" for some of the available algorithms. An algorithm S_i is considered to be better than algorithm S_j for a particular problem if it achieves better performance in terms of the necessary number of function evaluations for attaining a specific objective value threshold. The method consists of two phases. In the first (initialization) phase, a subset of the available algorithms is generated. This subset contains all solvers that achieve the best performance for at least one of the problems at hand. In the second phase, an iterative procedure is initiated, where each problem defines a partial ordering of the available algorithms based on the proposed performance measure. The algorithms that are preferred by the majority of the problems are then added to the portfolio.

Summarizing, the constituent algorithms selection problem in algorithm portfolios is an open problem directly connected to the well-known algorithm selection problem. Although its relevance and significance for the portfolio's performance is supported by strong experimental evidence, it is probably the less studied problem in relevant works. This can be partially attributed to the existence of abundant data for various optimization problem types, which may narrow the selection to specific algorithm types. This way the user can make more informative decisions on the considered candidates. Nevertheless, integrated solutions that would offer ad hoc selection options per problem or automate the whole selection procedure, considering also variable numbers of algorithms in the portfolio, would add significant merit to the general concept of algorithm portfolios.

3.2 Allocation of Computation Budget

The allocation of computational budget to the constituent algorithms is perhaps the most studied topic in algorithm portfolios' literature. The allocation problem can be summarized in two essential questions:

1. What fraction of the total computational budget shall be assigned to each constituent algorithm?
2. How is this budget allocated to the algorithms during the run?

Following the notation of Section 2, budget allocation requires the determination of the set $T = \{T_1, T_2, \ldots, T_n\}$, where T_i stands for the budget (function evaluations or running time) allocated to the constituent algorithm S_i, $i = 1, 2, \ldots, n$, such that Equation (1) holds. The goal is to distribute the available computational resources among the n constituent algorithms such that the expected objective function value, denoted as E_f, is minimized, i.e.,

$$\min_{T} E_f, \quad \text{subject to} \quad \sum_{i=1}^{n} T_i = T_{\max}, \quad \text{and} \quad T_i \geqslant 0, \ \forall i. \tag{3}$$

The expected objective function value is averaged over a number of independent experiments for each problem instance under consideration.

Naturally, the assignment of computational budget to algorithms has significant performance impact. Assigning computational resources prior to the execution of the algorithm portfolio may be insufficient since it neglects the online dynamic of each algorithm. For this reason, online techniques for the dynamic allocation of resources during the run have been recently proposed [4, 12, 19].

In [4] the proposed algorithm portfolio is composed of both population-based and local search metaheuristics, and it is applied on the dynamic maximal covering location problem. The portfolio is equipped with a learning mechanism that distributes the available credit (namely computational budget) among the constituent algorithms according to their performance. The credit assigned to each algorithm is based on a weight function that takes into account the credit of each algorithm in the previous iteration, a reward, and a penalization score. The reward score is added to the current weight if the algorithm improves the best so far solution. The penalization score is subtracted from the current weight and occurs when the algorithm results in a solution that is inferior than the overall best. Eventually, the weight is used to define a selection probability that is fed to a roulette wheel procedure.

The procedure above is incorporated into the main algorithm selection scheme and determines the algorithm that will be executed in the next iteration. In the specific study, the term "iteration" is interpreted differently for each type of constituent algorithm. Thus, in local search methods, iteration is defined as one application of the employed neighborhood operator and the computation of the corresponding objective function. In population-based metaheuristics, an iteration refers to the sequential application of the underlying evolutionary operators such as crossover, mutation, and selection.

A different population-based portfolio was introduced in [19]. The algorithms of that portfolio are individually executed without any exchange of information among them. Also, they are executed interchangeably, i.e., they automatically switch from one algorithm to another as a function of the available computational resources. During the search procedure, the performance of each algorithm is predicted by using its past objective function values. Specifically, a future time point is determined and the performance curve of each algorithm is extrapolated to that point. Extrapolation is conducted by applying a linear regression model to the performance curve of each algorithm between iterations $i - l$ and i. As l can admit different values, a number of linear models (straight lines) are the outcome of this procedure.

In turn, each linear model results in a corresponding predicted value, and all these values are used to create a bootstrap probability distribution. The final predicted value is sampled from the bootstrap probability distribution. Eventually, the algorithm that achieves the highest predicted function value is executed in the next iteration. The term iteration corresponds to the application of the algorithm's operators and the computation of the corresponding objective values, under the assumption that all algorithms have identical population sizes. An advantage of this allocation scheme is the lack of many parameters, which disburdens the practitioner from the tedious task of their tuning.

Another allocation scheme was proposed in [12], based on stock trading models similar to the ones that motivated the development of algorithm portfolios. The proposed parallel algorithm portfolio embeds a market trading-based budget allocation mechanism. This is used to dynamically distribute the available computational budget, rewarding the best-performing algorithms of the portfolio with additional execution time. The core idea behind the allocation mechanism assumes constituent algorithms-investors that invest on elite solutions, using execution time as the trading currency. Both population-based and local search metaheuristics are incorporated into the studied portfolio in [12]. Moreover, the constituent algorithms are concurrently executed by using the master–slave parallelism model, where slave-nodes host the constituent algorithms and the master-node coordinates the procedure. According to the proposed scheme in [12], all algorithms are initially allocated equal computational budgets, which is the running time in this case. Then, a fraction of the assigned time (investment time) is used by each algorithm to buy solutions from other algorithms, while its remaining time is devoted to the algorithm's own execution. During this procedure, the master-node retains an archive of one elite solution per algorithm. For each stored solution, a price is computed taking into consideration the objective values of all archived solutions in the elite set. Obviously, high-quality solutions are also the most costly ones.

In the case that an algorithm cannot improve its own best solution for a specific time period, it decides to bid for a better solution among the archived ones. To do so, the buyer-algorithm contacts the master-node to initiate a profitable bargain. Among the available archived solutions, the buyer-algorithm chooses to buy the one that maximizes a specific quality measure, called the *return on investment* index [12]. In simple words, among the affordable archived solutions (determined by its available investment time and the solution prices) the buyer-algorithm chooses to buy the solution that maximizes the improvement over its own best solution.

Then, the seller-algorithm, i.e., the one that discovered the traded solution, receives from the buyer-algorithm the amount of computational budget (running time) specified by the traded solution's price. Overall, the proposed allocation mechanism enables the best-performing algorithms to sell solutions more frequently, thus gaining additional execution time during their run. Note that the total execution time that was allocated initially to the algorithm portfolio remains unchanged throughout the optimization procedure. It is simply reallocated to the algorithms according to the aforementioned dynamic scheme.

Summarizing, the allocation of computational budget to the constituent algorithms of an algorithm portfolio is the main key for the portfolio's efficiency. The mechanism shall be capable of identifying either offline, based on historical performance data or preprocessing, or online, based on current feedback of the algorithms during their execution, the most promising algorithms and award them additional fractions of the total budget. Online approaches fit their budget allocation to the specific conditions during the portfolio's execution. Thus, they can be highly reactive to the course of the optimization procedure.

On the other hand, their outcome is hardly reusable in similar problems contrasting the offline methods. Given the variety of optimization problems and the diversity

of algorithm characteristics, the possibility of developing universal optimal budget allocation schemes seems questionable. Nevertheless, it is the authors' belief that enhanced ad hoc procedures can be developed by thoroughly analyzing (offline or online) the performance profiles of the constituent algorithms. To this end, machine learning and time series analysis can be valuable.

3.3 Interaction of Constituent Algorithms

In early algorithm portfolio models, the constituent algorithms were considered to be isolated [11, 14, 19], i.e., there was no form of interaction among them. In other works, some form of interaction existed [9, 12, 16]. Interaction comes in various forms. For instance, in [9] the proposed portfolios comprise different population-based algorithms. The portfolio holds separate subpopulations, each one assigned its own computational budget. Also, each subpopulation adopts one of the constituent algorithms. Interaction takes place through the exchange of individuals among the subpopulations.

This process is the well-known *migration* property, and it is applied following a simple scheme. In particular, two parameters are used, namely the *migration interval* and the *migration size*, which are defined prior to the portfolio's run. The migration interval determines the number of iterations between two consecutive migrations, while the migration size stands for the number of migrating individuals.

In [16], a multi-method framework that accommodates various evolutionary algorithms was proposed for real-valued optimization problems. According to this approach, the constituent algorithms are concurrently executed and share a joint population of search points. In the beginning, an initial population is randomly and uniformly created. Then, instead of using a single algorithm, all the constituent algorithms contribute offspring to the population of the next generation. The number of offspring per algorithm is determined through a self-adaptive learning strategy.

This strategy follows a two-step procedure. In the first step, the contribution of each algorithm to the overall improvement of the objective function value is computed. In the second step, this value is used to determine the fraction of offspring generated by each algorithm in the next iteration. Obviously, the algorithms that achieve higher improvements are offered the chance to be more productive than the rest.

Besides the use of joint populations or subpopulations, other types of information can be exchanged among the constituent algorithms. For example, in [12] the proposed portfolios include both population-based and local search algorithms, which are executed in parallel based on a master–slave model. Then, whenever an algorithm fails to improve its findings for a specific amount of time, it receives the best solution achieved by a different algorithm.

Overall, the interaction of constituent algorithms can be beneficial for an algorithm portfolio especially when population-based algorithms are used. Migration of individuals between populations can offer the necessary boost to less efficient

algorithms or may lead to improved solutions if it is used as initial condition for a local search algorithm. Moreover, information exchange promotes interaction and cooperation between the algorithms, also promoting performance correlations. Nevertheless, it is a concept that remains to be studied individually as well as in combination with the budget allocation approaches.

3.4 The Role of Parallelism

The constituent algorithms of an algorithm portfolio can be executed either interchangeably on a single CPU or concurrently on many CPUs. Three essential parallelism models are used in parallel portfolios. Probably the most popular one is the *master–slave* model, which has been widely used [12, 14]. In this model, the slave-nodes usually host the constituent algorithms, while the master-node is devoted to coordination and book-keeping services.

Figure 1 depicts a parallel heterogeneous and a parallel homogeneous portfolio based on the master–slave model. The heterogeneous portfolio consists of two variable neighborhood search (VNS) metaheuristics and two iterated local search (ILS) algorithms. The two instances in each pair of metaheuristics assume the same parameter configuration between them. The homogeneous portfolio includes four copies of the VNS algorithm with the same configuration. Note that each metaheuristic runs on a single slave-node, which sends (e.g., periodically) its best solution to the master-node during the optimization procedure.

A different approach is the *fine-grained* parallelism model. This is used mostly for population-based metaheuristics when massive parallel computers are available [1]. According to this model, the algorithm consists of a single population and each individual that belongs to at least one neighborhood is confined to communicate with the members of its own neighborhood. Overlaps among different neighborhoods may also exist, promoting interactions among all individuals of the population. Obviously, the neighborhood topology and the number of individuals that comprise the neighborhood affect the performance of this model. Typically, a number of individuals of the population are assigned to a single processing unit,

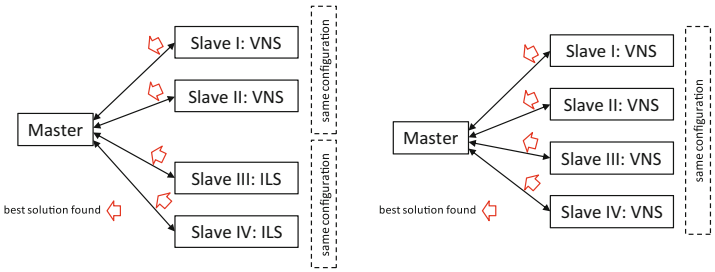

Fig. 1 Heterogeneous (left) and homogeneous (right) parallel algorithm portfolios

while the ideal case for time-consuming problems would allow only one individual per CPU. However, in ordinary problems such distributed computation may impose additional communication delays and should be avoided.

Another model is the so-called *coarse-grained* parallelism model, which is also very common in population-based algorithms [1]. In this model, a population consists of individual subpopulations, each one running on a single CPU. The following parameters control the performance of such models:

1. Model topology: it specifies the communication links among subpopulations.
2. Migration period: it defines the frequency of migrations.
3. Migration size: it determines the number of migrating individuals.
4. Migration type: it distinguishes between synchronous and asynchronous migration.

Synchronous migration takes place at prespecified time points (e.g., periodically), whereas asynchronous migration dictates that subpopulations communicate when specific events occur.

It becomes obvious that the design of parallel algorithm portfolios raises a number of issues. Special attention is required in the selection of the suitable parallelism model and its parameters tuning [1]. Despite that, parallelism shall be promoted against sequential approaches in order to take full advantage of modern computer systems and increase the portfolios' time-efficiency [1].

An equally important goal refers to effectiveness in terms of solution quality. Even a simplistic portfolio that comprises copies of the same algorithm can in some cases outperform its sequential counterparts, under the assumption that both models receive equal computational resources [14]. This is attributed to the fact that more than one threads concurrently explore different parts of the search space, hence the probability of detecting better solutions is significantly increased.

Overall, parallelism can offer obvious advantages to algorithm portfolios in terms of time-efficiency. Also, the parallelism model can influence the synchronization and information flow in interactive portfolios. Thus, it is an open problem of interest that shall be carefully considered in combination with the previously exposed open problems in order to guarantee the development of more efficient approaches.

4 Conclusions

Metaheuristics-based algorithm portfolios have gained ongoing popularity as promising alternatives for tackling hard optimization problems. So far, their effectiveness and efficiency have been identified on several challenging problems spanning diverse research areas. The present chapter provided information on recent research trends and the most important issues in the design of efficient algorithm portfolios.

In the relevant literature, four essential open problems can be distinguished, namely the selection of constituent algorithms, resource allocation schemes, inter-

action among the algorithms, and parallelism against sequential implementations. Each problem individually but, mostly, their interplay in the design of an algorithm portfolio can draw the borderline between a top-performing scheme and a failure.

Research developments of the past decade offer the necessary motivation for further discussion and elaboration on algorithm portfolios. It is the authors' belief that the accessibility to powerful parallel machines in the forthcoming years will further expand research on algorithm portfolios. Latest developments on machine learning and data analysis cultivate the ground toward this goal, placing algorithm portfolios in a salient position among the available algorithmic artillery for global optimization.

References

1. Akay, R., Basturk, A., Kalinli, A., Yao, X.: Parallel population-based algorithm portfolios. Neurocomputing **247**, 115–125 (2017)
2. Almakhlafi, A., Knowles, J.: Systematic construction of algorithm portfolios for a maintenance scheduling problem. In: IEEE Congress on Evolutionary Computation, Cancun, Mexico, pp. 245–252 (2013)
3. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Inf. Sci. **237**, 82–117 (2013)
4. Calderín, J.F., Masegosa, A.D., Pelta, D.A.: An algorithm portfolio for the dynamic maximal covering location problem. Memetic Comput. **9**(2), 141–151 (2017)
5. Gomes, C.P., Selman, B.: Algorithm portfolios. Artif. Intell. **126**(1), 43–62 (2001)
6. Hart, E., Sim, K.: On constructing ensembles for combinatorial optimisation. Evol. Comput. **26**(1), 67–87 (2018)
7. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science **275**(5296), 51–54 (1997)
8. Muñoz, M.A., Kirley, M.: Icarus: identification of complementary algorithms by uncovered sets. In: IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, pp. 2427–2432 (2016)
9. Peng, F., Tang, K., Chen, G., Yao, X.: Population-based algorithm portfolios for numerical optimization. IEEE Trans. Evol. Comput. **14**(5), 782–800 (2010)
10. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
11. Shukla, N., Dashora, Y., Tiwari, M., Chan, F., Wong, T.: Introducing algorithm portfolios to a class of vehicle routing and scheduling problem. In: 2nd International Conference on Operations and Supply Chain Management (OSCM), Bangkok, Thailand, pp. 1015–1026 (2007)
    In: Proceedings OSCM 2007, pp. 1015–1026 (2007)
12. Souravlias, D., Parsopoulos, K.E., Alba, E.: Parallel algorithm portfolio with market trading-based time allocation. In: Lübbecke, M., Koster, A., Letmathe, P., Madlener, R., Peis, B., Walther, G. (eds.) Operations Research Proceedings 2014, pp. 567–574. Springer, Berlin (2016)
13. Souravlias, D., Parsopoulos, K.E., Kotsireas, I.S.: Circulant weighing matrices: a demanding challenge for parallel optimization metaheuristics. Optim. Lett. **10**(6), 1303–1314 (2016)
14. Souravlias, D., Parsopoulos, K.E., Meletiou, G.C.: Designing Bijective S-boxes using algorithm portfolios with limited time budgets. Appl. Soft Comput. **59**, 475–486 (2017)
15. Tang, K., Peng, F., Chen, G., Yao, X.: Population-based algorithm portfolios with automated constituent algorithms selection. Inf. Sci. **279**, 94–104 (2014)

16. Vrugt, J.A., Robinson, B.A., Hyman, J.M.: Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans. Evol. Comput. **13**(2), 243–259 (2009)
17. Wolpert, D.H., Macready, W.G.: No free lunch theorem for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
18. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla-07: the design and analysis of an algorithm portfolio for SAT. In: International Conference on Principles and Practice of Constraint Programming, Providence, RI, USA, pp. 712–727 (2007)
19. Yuen, S.Y., Chow, C.K., Zhang, X., Lou, Y.: Which algorithm should I choose: An evolutionary algorithm portfolio approach. Appl. Soft Comput. **40**, 654 – 673 (2016)