Optimization Methods and Software

ISSN: 1055-6788 (Print) 1029-4937 (Online) Journal homepage: http://www.tandfonline.com/loi/goms20

Parallel algorithm portfolios with performance forecasting

D. Souravlias, I. S. Kotsireas, P. M. Pardalos & K. E. Parsopoulos

To cite this article: D. Souravlias, I. S. Kotsireas, P. M. Pardalos & K. E. Parsopoulos (2018): Parallel algorithm portfolios with performance forecasting, Optimization Methods and Software, DOI: 10.1080/10556788.2018.1484123

To link to this article: https://doi.org/10.1080/10556788.2018.1484123

Published online: 21 Jun 2018.

Submit your article to this journal

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
http://www.tandfonline.com/action/journalInformation?journalCode=goms20

OPTIMIZATION METHODS & SOFTWARE
https://doi.org/10.1080/10556788.2018.1484123

Taylor & Francis
Taylor & Francis Group

Check for updates

Parallel algorithm portfolios with performance forecasting

D. Souravlias[a], I. S. Kotsireas[b], P. M. Pardalos[c,d] and K. E. Parsopoulos[e]

[a]Department of Logistics Management, Helmut Schmidt University, Hamburg, Germany; [b]Department of Physics and Computer Science, Wilfrid Laurier University, Waterloo, ON, Canada; [c]Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA; [d]National Research University Higher School of Economics, Laboratory of Algorithms and Technologies for Network Analysis, Nizhny Novgorod Russia; [e]Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

ABSTRACT

We propose a novel algorithm portfolio model that incorporates time series forecasting techniques to predict online the performance of its constituent algorithms. The predictions are used to allocate computational resources to the algorithms, accordingly. The proposed model is demonstrated on parallel algorithm portfolios consisting of three popular metaheuristics, namely tabu search, variable neighbourhood search, and multistart local search. Moving average and exponential smoothing techniques are employed for forecasting purposes. A challenging combinatorial problem, namely the detection of circulant weighing matrices, is selected as the testbed for the analysis of the proposed approach. Experimental evidence and statistical analysis provide insight on the performance of the proposed algorithms and reveal the benefits of using forecasting techniques for resource allocation in algorithm portfolios.

ARTICLE HISTORY
Received 7 June 2017
Accepted 30 May 2018

KEYWORDS
Algorithm portfolios; metaheuristics; resource allocation; performance forecasting; parallel algorithms; combinatorial optimization

AMS SUBJECT CLASSIFICATIONS
05B20; 37M10; 68T20; 68W10; 68W20

1. Introduction

A central problem in computational optimization lies in the selection of the most appropriate algorithm and parameter setting for solving a given problem. The so-called algorithm selection problem has received the attention of the research community for many decades [33,35] due to the observed performance fluctuations of optimization algorithms when applied on different problems or different instances of a specific problem type. Strong theoretical results such as the No Free Lunch Theorem [42] suggest that no universal algorithm exists that can tackle all problems equally well. Nevertheless, experimental evidence has shown that specific algorithmic instances can be particularly effective and efficient in specific optimization problems [34] or instances of a specific problem type [2]. The ability to identify the most appropriate algorithm eventually determines the boundary between success and failure when challenging optimization problems are confronted.

Algorithm selection methods usually operate offline [2]. Given a collection of relevant problems (e.g. different instances of the travelling salesman problem) and a collection of different algorithms or different parameterizations of a specific algorithm, selection

CONTACT K. E. Parsopoulos kostasp@cse.uoi.gr

© 2018 Informa UK Limited, trading as Taylor & Francis Group

methods identify the most promising solver based on preliminary experimentation on a representative (training) subset of the problems. Then, the selected approach is used to solve the rest of the problems. Despite the success of such methods on collections of similar problems, their direct applicability on previously unseen problem types without the preliminary experimentation phase is questionable. Also, the offline methods neglect possible performance fluctuations of the algorithm during its run. While an algorithm may prove to be very efficient in early stages of the optimization procedure, it may exhibit declining performance after a critical number of iterations. At that point, dynamically switching to a different algorithm can be highly beneficial.

Algorithm portfolios [1,20,22,34,37,41,43] were introduced as general frameworks for building algorithmic schemes that alleviate deficiencies originating in the selection of a single algorithm for tackling a specific optimization problem. Instead of struggling for the selection of a single algorithm, which includes the risk of a poor choice, portfolios admit a set of possible solvers for the given problem and try to optimally allocate the available computational resources among them. The constituent algorithms of the portfolio are executed either serially and interchangeably on a single processing unit [43] or in parallel when many processors are available [1,37]. In the serial case, a fraction of the available computational budget is consumed by one algorithm before proceeding to the next algorithm in a round-robin manner. In the parallel case, a number of the available processing units is devoted to each algorithm. In both cases, the algorithms assume no interaction among them and the prescribed allocated fractions of the computational budget are determined through a preprocessing phase based on preliminary experimentation [20,22].

The use of many algorithms instead of one can relieve the practitioner from the error-prone selection of a single solver. However, the assignment of prespecified fractions of the computational resources to each algorithm may be inefficient since it neglects the online dynamics of the algorithms. In such cases, online techniques for the dynamic allocation of resources during the portfolio's run can be beneficial, especially in computationally demanding problems.

Metaheuristics have been frequently used in an algorithm portfolio framework, in order to alleviate the algorithm selection problem and exploit the diverse dynamics of different solvers. Recently, such schemes were proposed for solving difficult combinatorial optimization problems in parallel computing environments [26,38]. The underlying resource allocation scheme was proved to have significant impact on the portfolio's performance on these problems. Sophisticated allocation schemes were shown to achieve high-performance standards in terms of solution quality and time efficiency of the portfolios against their constituent algorithms individually [38]. Also, the domination of trajectory-based approaches, especially tabu search, has been verified against other algorithms.

The present work proposes a novel parallel algorithm portfolio model that incorporates time series forecasting techniques. Forecasting is periodically applied during the run of the portfolio to predict online the performance of the constituent algorithms based on their achievements. Thus, the portfolio can make informative decisions on the allocation of the available computational resources (processing units). We demonstrate the idea on a heterogeneous algorithm portfolio consisting of three popular metaheuristics, namely tabu search, variable neighbourhood search, and multistart local search. These solvers have been widely used in combinatorial optimization, and they are combined with three essential

forecasting models, namely the moving average, simple exponential smoothing, and linear exponential smoothing.

Combinatorial matrices are involved in diverse applications such as quantum information processing, coding theory, and statistical experimentation [4,27,40]. Significant amount of theoretical research has been devoted to the investigation of existence of finite or infinite classes of circulant weighing matrices [5,6,11,12,16]. Whenever theoretical analysis has been fruitless [24], computational methods have been used by transforming the problem to an equivalent permutation optimization task. Metaheuristics have proved to be effective solvers in such cases. Motivated by the difficulty of this problem type, we decided to evaluate the proposed algorithm portfolio model on a testbed of circulant weighing matrices detection problems of variable dimension. The performance of the model was studied and statistically analysed with and without the use of performance forecasting, verifying the benefits of the proposed approach.

The rest of the paper is structured as follows: Section 2 offers background information on the employed algorithms and test problems. The proposed algorithm portfolios are described in Section 3 and the experimental analysis is reported and discussed in Section 4. The paper closes with conclusions in Section 5.

2. Background information

In the following paragraphs, we provide necessary background information. This includes very brief descriptions of the general algorithm portfolio model, its constituent algorithms, the considered time series forecasting models, and the basic formulation of the circulant weighing matrices problems that served as our testbed for experimentation. Detailed descriptions of the presented algorithms and concepts can be found in the provided references.

2.1. Algorithm portfolios

Algorithm portfolios [22] are defined as collections of algorithms that operate either interchangeably or concurrently on a given problem by sharing the available computational resources. They are characterized as *homogeneous* if they consist of instances of the same algorithm or *heterogeneous* if they comprise different algorithms. If only a single processing unit is available, the portfolio's constituent algorithms are alternately executed according to a time-sharing schedule. Otherwise, they are executed in parallel with each algorithm typically occupying a number of the available processing unit [20,26]. Standard algorithm portfolios assume no interaction among their constituent algorithms, although recent studies with interactive algorithms suggest that performance benefits are possible through information exchange [37,38]. Extensive experimentation has shown that the use of algorithm portfolios with appropriate resource allocation to their constituent algorithms can offer significant performance benefits against the application of an individual algorithm equipped with all the available resources [20,22,37,38].

2.2. Tabu search

Tabu search [17] is a widely used local search algorithm, originally proposed for solving discrete optimization problems. It is based on two essential components, namely

neighbourhood search and a short-term memory called the *tabu list*. The algorithm is initialized on a randomly selected position in the search space. At each iteration, the current position is replaced by the best position in its neighbourhood (excluding itself). This property equips tabu search with local search and hill-climbing capability, which is necessary for alleviating local minimizers. In order to avoid cyclic moves, the l most recent visited positions are stored in the tabu list and they are prohibited for a number of iterations. Also, the algorithm is restarted on a new random position whenever a prescribed number of iterations, t_{rest}, with no improvement of the overall best solution is exceeded. This promotes the exploration of different regions of the search space by deploying new trajectories. The reader is referred to [15,18,19] for a detailed presentation of the algorithm.

2.3. Variable neighbourhood search

Variable neighbourhood search [31] is a popular metaheuristic in combinatorial optimization. The main idea is the use of various neighbourhood structures and the systematic change among them either for approximating local minima or producing new initial points. Thus, whenever the search stagnates within one neighbourhood, the next neighbourhood is selected allowing the algorithm to alleviate local minimizers. It is not necessary to search each neighbourhood exhaustively. Instead, a prespecified number of trial points is generated before proceeding to the next neighbourhood. If the whole procedure does not provide any improvement, then the algorithm is initialized to a new random point selected in one of the available neighbourhoods. The neighbourhoods may come in various forms depending on the problem at hand. For instance, in permutation optimization problems they are defined through different permutation search operators [44]. A comprehensive introduction on this algorithm can be found in [21].

2.4. Multistart local search

This is the simplest of the employed algorithms. It consists of pure local search starting from different initial points, which are randomly generated in the search space. Specifically, a random initial point is specified and local search is initiated until the nearest local minimum is reached. Naturally, locality depends on the employed neighbourhood structure. Then, a new random initial point is taken and the procedure is repeated until the available computational budget is exceeded. During this procedure, the best detected solution is retained. Further details on this approach can be found in [28].

2.5. Time series forecasting models

Time series forecasting [8,9] is concerned with the prediction of a model based on its historical observations. The main struggle lies in the detection of statistical patterns in the available data. Among the plethora of available models for time series forecasting, *exponential smoothing* [13,14] is distinguished as a simple and straightforward approach. For the needs of our proposed model, we considered the following forecasting schemes.

(1) *Simple moving average* [23,30]: This is the simplest approach for data smoothing. If f_t denotes the observed quantity at time moment t, then the predicted value at $t+1$ is the

average of the k most recent observations,

$$\hat{f}_{t+1} = \frac{1}{k} \sum_{i=0}^{k-1} f_{t-i}. \tag{1}$$

In this model, each one of the k observations has equal weight. Thus, increasing k results in declining impact of the most recent observations, producing smoother forecast series. The case $k = 1$ is the well-known *random walk* model, which has the property of following the exact path of the predicted variable but with 1-period lag.

(2) *Simple exponential smoothing* [29,30,36]: This is also known as the exponentially weighted moving average model. Its main difference from the previous one lies in the assumption of gradually decreasing weights for older observations. Thus, the next forecasted value is given as

$$\hat{f}_{t+1} = \alpha f_t + (1 - \alpha) \hat{f}_t, \tag{2}$$

with $\alpha \in [0, 1]$ being the smoothing constant. Smaller values of α produce smoother forecast series, while higher values assume that each observation introduces significant changes in the level of the series.

(3) *Linear exponential smoothing* [30]: This model takes into consideration both the level and trend of the time series. If l_t and r_t denote the local estimates of level and trend, respectively, then their values are computed as

$$\text{Trend}: \quad l_t = \alpha f_t + (1 - \alpha)(l_{t-1} + r_{t-1}), \tag{3}$$

$$\text{Level}: \quad r_t = \beta (l_t - l_{t-1}) + (1 - \beta) r_{t-1}, \tag{4}$$

where $\alpha, \beta \in [0, 1]$ are weighing factors. Eventually, the forecast for k steps ahead is computed as

$$\hat{f}_{t+k} = l_t + k r_t. \tag{5}$$

Higher values of β assume rapid changes in the trend of the data, while smaller values match slower changes.

The forecasted variable can be considered as the aggregation of pure signal and noise, where the signal can be predicted while the noise introduces distortion to its values. The forecasting model shall be capable of capturing and extrapolating the underlying signal in time. Naturally, this is far from an easy task for complex signals.

2.6. Circulant weighing matrices

Circulant weighing matrices [5] are combinatorial matrices with special properties. A square $n \times n$ matrix $W = [w_{ij}]$ with entries $w_{ij} \in \{-1, 0, 1\}$, $i, j \in I \triangleq \{1, 2, \ldots, n\}$, is called a *weighing matrix of order n and weight* γ, if there exists a positive integer $\gamma < n$, such that $W W^T = \gamma \mathcal{I}_n$, where $\mathcal{I}_n$ is the identity matrix of dimension n, and W^T denotes the transpose matrix of W. A *circulant weighing matrix*, denoted as $\text{CW}(n, k^2)$, is a weighing matrix with the property that each row, excluding the first one, is a right cyclic shift

of its preceding row. Thus, the matrix can be completely defined solely by one of its rows. Note that the weight of a circulant weighing matrix is always a perfect square.

Various methodologies have been proposed for the systematic detection of circulant weighing matrices of various orders and weights [3,7,12,39]. In addition to theoretical algebraic methods, computational optimization algorithms have been successfully applied. In these cases, the original problem is modelled as a permutation optimization problem where each global minimizer corresponds to a matrix of the desirable type, i.e. it defines a row of the matrix (the first row, without loss of generality). The objective function of the corresponding optimization problem is based on the *periodic autocorrelation function* [24]. More specifically, let

$$\mathbb{T}^n = \{\mathbf{x} = (x_1, x_2, \ldots, x_n) : \ x_i \in \{-1, 0, +1\} \text{ for all } i \in I\},$$

be the set of all ternary sequences of length n, and $\mathbf{x} \in \mathbb{T}^n$ be the first row that defines a $\mathrm{CW}(n, k^2)$ matrix. Then, its periodic autocorrelation function values are defined as

$$\mathrm{PAF}_{\mathbf{x}}(s) = \sum_{i=1}^{n} x_i \, x_{i+s}, \quad s = 0, 1, \ldots, n-1, \tag{6}$$

where $i+s$ is taken modulo n when $i+s > n$. Note that the symmetry property $\mathrm{PAF}_{\mathbf{x}}(s) = \mathrm{PAF}_{\mathbf{x}}(n-s)$ allows to use only the first $[n/2]$ terms in Equation (6). The defining property of circulant weighing matrices is then equivalent to the system

$$\mathrm{PAF}_{\mathbf{x}}(s) = 0, \quad s = 0, 1, \ldots, n-1.$$

Thus, the admissible sequences that define $\mathrm{CW}(n, k^2)$ matrices are the global minimizers of the combinatorial optimization problem,

$$\min_{\mathbf{x} \in \mathbb{T}^n} \ f(\mathbf{x}) = \sum_{s=1}^{[n/2]} \left| \mathrm{PAF}_{\mathbf{x}}(s) \right|. \tag{7}$$

Moreover, it is proved that each admissible sequence has exactly k^2 non-zero (i.e. $+1$ or -1) components, $k(k+1)/2$ components equal to $+1$, and $k(k-1)/2$ components equal to -1. Taking into consideration the fixed number of appearances of 0, $+1$, and -1 in the sequence, the problem becomes a permutation optimization problem. Various approaches including metaheuristics have been extensively used for solving such problems [24–26,38]. Experimental evidence has shown that the difficulty level of the problem increases with the length n of the sequence and the weight k^2 of the matrix.

2.7. Neighbourhood search operators

The presented metaheuristics have a common prerequisite for their application, namely a proper neighbourhood structure (or more than one in the case of variable neighbourhood search). For this reason, we considered four state-of-the-art neighbourhood search operators for permutation optimization [44]. Specifically, let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, be a sequence

that defines a $\mathrm{CW}(n, k^2)$ matrix (i.e. $\mathbf{x}$ is its first row), and let $i < j$ be two component indices. Then, the following neighbourhood search operators can be used:

(O_1) *Interchange Operator*: swaps x_i and x_j.
(O_2) *Relocate Operator*: relocates x_i immediately after x_j.
(O_3) *Or-Opt Operator*: relocates x_i and x_{i+1} between x_j and x_{j+1}.
(O_4) *2-Opt Operator*: interchanges x_{i+1} and x_j and reverses the order of all components between them.

The neighbourhood of $\mathbf{x}$ for each operator consists of all vectors that can be produced by a simple application of the corresponding operator on $\mathbf{x}$. Note that these operators except O_1 shift or reverse whole subsequences of $\mathbf{x}$, with O_4 having the strongest expected impact.

Another issue of interest is whether the algorithm will conduct exhaustive search in each neighbourhood or directly select the first point of improvement. The first case is the well-known *neighbourhood-best* approach, which conducts the deepest descent and guarantees detection of a local minimizer if it lies in the neighbourhood of the current point. The second one is the *first-best* approach, which moves rapidly from one point to another sparing function evaluations, although at the risk of temporarily overshooting the minimizer. These two approaches are rather complementary and they are frequently considered together in relevant applications.

In our portfolio implementation, the standard tabu search algorithm with the O_1 operator was considered, while the variable neighbourhood search utilized all operators. Multistart local search used all but O_4 because less impactful approaches seem to be more suitable for such simplistic local search procedures. Moreover, both the neighbourhood-best and the first-best search were considered. Further details on the employed algorithms are postponed until Section 4.

3. Proposed algorithm portfolio model

A metaheuristic optimization algorithm can be considered as a stochastic system that evolves through time. Its state signal during a run can be defined as the function value of the best solution it achieved up to the current time moment or iteration. This signal can be tracked and extrapolated in order to predict the algorithm's performance. Random performance fluctuations due to stochasticity or dynamic adaptation (e.g. through parameter tuning) introduce noise to the observed signal. The noise can be detrimental for the quality of the predictions if it is based solely on simple observations or explicit if-then-else rules.

The main idea in our approach lies in monitoring the performance signal of each constituent algorithm of the portfolio in order to predict its forthcoming performance. Then, the predictions are used to proportionally allocate the available computational resources, which are the available processing units (CPUs) in the high-performance computation environments of our interest.

Putting it formally, let $P = \langle A_1, A_2, \ldots, A_N \rangle$ denote an algorithm portfolio consisting of N algorithms. Let $C \geqslant N$ denote the number of the available CPUs. These can be either physical cores or processing threads. Also, let $e_{\max}$ denote the available computational budget in terms of function evaluations. This means that the portfolio is terminated when the total number of function evaluations spent by all its constituent algorithms exceeds $e_{\max}$.

Both C and $e_{\max}$ are external parameters dictated by the computation environment and user-related restrictions.

In our model, the computational budget is assigned to the portfolio in *batches*. Specifically, the user defines a desirable number of batches $b_{\max}$. At each batch b, a fixed number of function evaluations,

$$e_b = \frac{e_{\max}}{b_{\max}}, \quad b = 1, 2, \ldots, b_{\max}, \tag{8}$$

is allocated to the portfolio and shall be completely consumed before the next batch assignment. These function evaluations are equally shared among the C CPUs. Thus, the computational budget allocated to each CPU per batch is fixed and equal to

$$e_c = \frac{e_b}{C} = \frac{e_{\max}}{C\, b_{\max}}, \quad c = 1, 2, \ldots, C. \tag{9}$$

At each batch, each CPU is occupied strictly by one of the available algorithms of the portfolio. Thus, the main decision issue is the allocation of the N algorithms to the C CPUs, i.e. the number of copies of each algorithm in the present batch. Let c_b^i denote the number of CPUs running algorithm A_i during batch b. Each one of these processing units executes the specific algorithm independently of the rest. Obviously, it shall hold that

$$C = \sum_{i=1}^{N} c_b^i, \tag{10}$$

for all $b = 1, 2, \ldots, b_{\max}$. At the end of batch b, the best performance achieved by each algorithm is recorded. This consists of the best solution value achieved by all instances of the algorithm from the start of the run. Using this information, forecasting methods are used to predict the performance of each algorithm in the forthcoming $b{+}1$ batch.

According to the predicted performance, the new number of CPUs that will be occupied by each algorithm in the next batch is determined. Specifically, let f_b^i denote the overall best solution value detected by all instances of algorithm A_i at the end of batch b. We assume that the objective value of the optimization problem at hand is strictly nonnegative, as in the permutation problems of our interest. Also, let $\hat{f}_b^i$ be the corresponding predicted best solution value of algorithm A_i (regardless of the employed forecasting model). The N sets,

$$H_b^i = \left\{ f_1^i, \hat{f}_2^i, f_2^i, \hat{f}_3^i, f_3^i, \ldots, \hat{f}_b^i, f_b^i \right\}, \quad i \in \{1, 2, \ldots, N\},$$

contain all the available (actual and forecasted) performance data achieved by each algorithm A_i up to batch b. Additional information may be also included in these sets depending on the selected forecasting method (e.g. trend values shall be additionally stored for the linear exponential smoothing model). This information is used to predict the performance of each algorithm A_i in the next batch $b{+}1$. Specifically, let

$$\hat{f}_{b+1}^i = \texttt{forecast}\left(H_b^i\right), \quad i \in \{1, 2, \ldots, N\},$$

be the forecasted solution values of the algorithms for the next batch. Then, the fraction of the available CPUs that will be occupied by algorithm A_i in batch $b{+}1$ is given by the

normalized inverse of its predicted performance value,

$$\eta_{b+1}^i = \frac{1/\hat{f}_{b+1}^i}{\sum_{j=1}^{N} 1/\hat{f}_{b+1}^j}, \quad i \in \{1, 2, \ldots, N\}. \tag{11}$$

Using these fractions, the actual number of CPUs that host each algorithm in the next batch is determined through a resource allocation plan

$$c_{b+1}^i = \texttt{allocate}\left(\eta_{b+1}^i, C\right), \quad i \in \{1, 2, \ldots, N\}, \tag{12}$$

taking care that Equation (10) holds. For example, a simple allocation procedure may directly set $c_{b+1}^i = [\eta_{b+1}^i\, C]$, where [.] stands for rounding to the nearest integer. However, correction may be needed since the outcome of the rounded quantities is not guaranteed to sum up to C. In our implementation, we considered a different scheme that uses the floor values

$$c_{b+1}^i = \left\lfloor \eta_{b+1}^i\, C \right\rfloor, \quad i \in \{1, 2, \ldots, N\},$$

to make a first-round assignment of algorithms to CPUs. This approach may leave spare processing units, which are subsequently assigned to the algorithms in a second allocation round. Specifically, the algorithm with the best (highest) η_{b+1}^i gets the first spare CPU, the

Algorithm 1 Algorithm Portfolio with Performance Forecasting

INPUT: $A_1, \ldots, A_N$ (algorithms); C (available processing units); $b_{\max}$ (number of batches); $e_{\max}$ (computational budget)
set e_c according to Equation (9) for all $c = 1, 2, \ldots, C$
$b \leftarrow 0$
`/* processing units equally shared in 1st batch */`
$\eta_{b+1}^i \leftarrow 1/N$ for all $i = 1, \ldots, N$
$c_{b+1}^i \leftarrow \texttt{allocate}\left(\eta_{b+1}^i, C\right)$ for all $i = 1, \ldots, N$
$H_b^i \leftarrow \emptyset$ for all $i = 1, \ldots, N$
`/* loop on the number of batches */`
for $(b = 1 \ldots b_{\max})$ **do**
    `/* parallel execution of the algorithm portfolio */`
    `execute_algorithm` $\left(A_i, c_b^i, e_c\right)$ for all $i = 1, \ldots, N$
    for $(i = 1 \ldots N)$ **do**
        `update_best` $\left(f_b^i\right)$
        $H_b^i \leftarrow H_{b-1}^i \cup \{f_b^i\}$
        $\hat{f}_{b+1}^i \leftarrow \texttt{forecast}\left(H_b^i\right)$
        $H_b^i \leftarrow H_b^i \cup \{\hat{f}_{b+1}^i\}$
    end for
    compute η_{b+1}^i and p_{b+1}^i according to Equations (11) and (12) for all $i = 1, \ldots, N$
end for
report overall best solution

second best algorithm gets the second spare CPU, and we continue in the same manner until all spare CPUs have been occupied.

The proposed procedure is given in the pseudocode of Algorithm 1 and depicted in the flowchart of Figure 1. In a parallel master-slave environment, better work division is attained when the master node is devoted to book-keeping, forecasting, and resource allocation procedures, while the rest of the CPUs (slave nodes) are devoted to the execution of the portfolio's constituent algorithms. In this framework, Algorithm 1 can be executed on the master node except step 11, which can utilize all slave nodes.

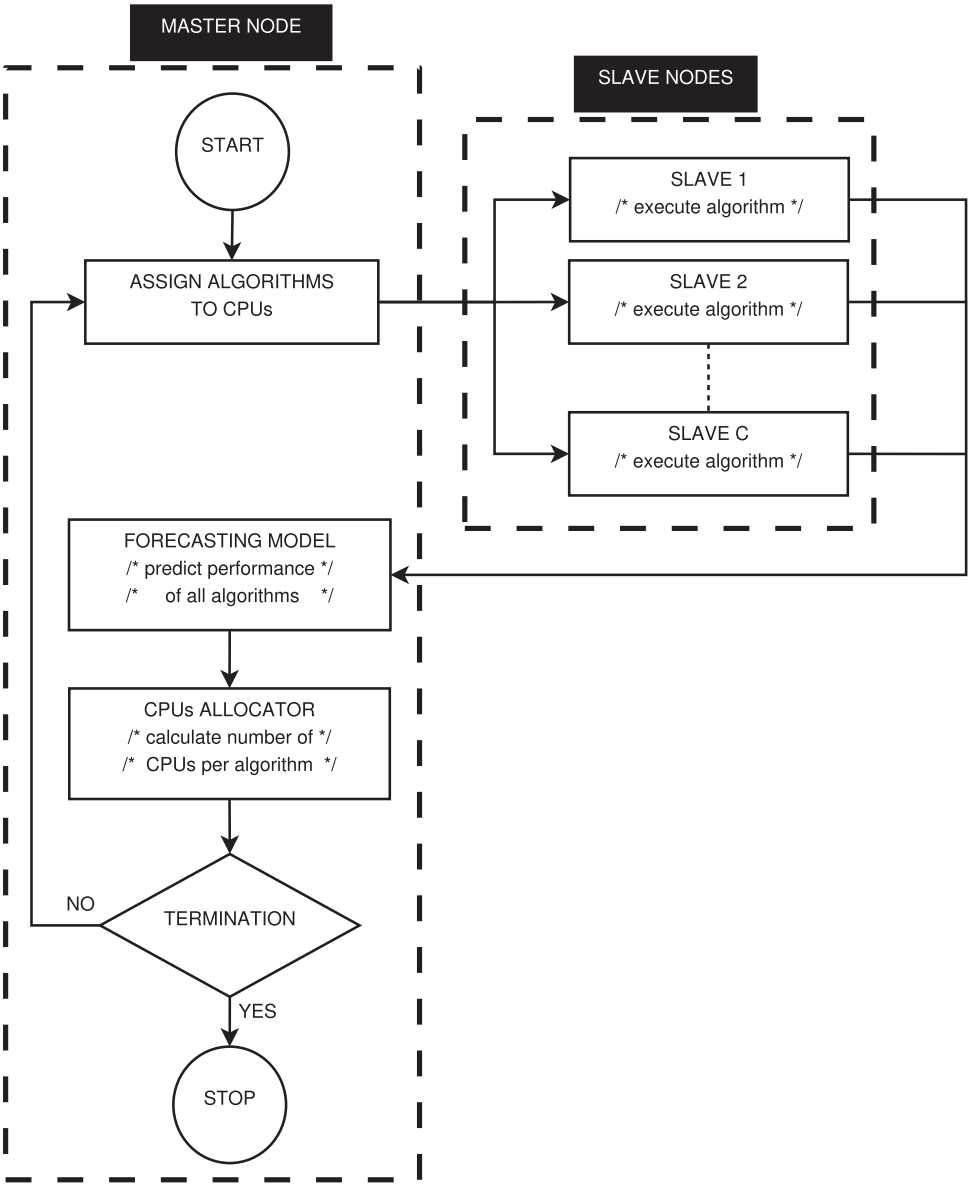

Figure 1. Flowchart of the proposed algorithm portfolio model.

Another issue of interest is the minimum number of CPUs occupied by each algorithm. Specifically, it is widely perceived that the efficiency of algorithm portfolios stems from the inclusion of diverse algorithms. Complementarity of the constituent algorithms has been widely recognized as performance booster of the portfolio since weaknesses of one algorithm can be addressed by another [32]. Thus, the `allocate()` function in Equation (12) shall guarantee that c_{b+1}^{i} does not vanish for any algorithm. In order to avoid this potential deficiency, we recommend at each batch to initially assign each one of the N algorithms to one CPU regardless of its predicted performance and, then, allocate the rest $C-N$ CPUs proportionally to the algorithms according to the procedure described above. This way, it is guaranteed that $c_{b}^{i} \geqslant 1$, for all batches $b = 1, 2, \ldots, b_{\max}$.

It shall be noted that the proposed algorithm portfolio model does not exceed the user-defined computational budget $e_{\max}$. It rather allocates it more efficiently by favouring the best-performing algorithms, which occupy more CPUs at each batch. Thus, in the long-run, dominant algorithms are provided with higher number of function evaluations than the rest. Nevertheless, the total computational cost of the portfolio remains fixed to its predefined value. This is a significant property for algorithms executed in parallel environments where execution time shall be predetermined under the penalty of deferred execution or additional charges if exceeded.

4. Experimental analysis

We validated the proposed approach through simulation experiments. The complete experimental configuration is summarized in Table 1. For the demonstration of the proposed approach, we considered portfolios consisting of seven algorithms, namely two variants of tabu search, two variants of variable neighbourhood search, and three variants of multistart local search, under different configurations. Specifically, tabu search assumed tabu list size $l = n/2$ in both variants, although the one utilized the neighbourhood-best and the other one utilized the first-best search approach (both with the O_1 neighbourhood operator). Similarly, one variant of variable neighbourhood search was based on

Table 1. Experimental configuration.

Description	Notation	Value (s)
Test problems	CW(n, k^2)	CW(33, 25), CW(42, 16), CW(48, 36), CW(52, 36) CW(57, 49), CW(62, 16), CW(70, 16), CW(78, 9) CW(84, 16), CW(96, 9), CW(112, 16), CW(130, 9)
Tabu search (TS)	A_1	Tabu list size $l = n/2$, neighbourhood-best
	A_2	Tabu list size $l = n/2$, first-best
Variable neighbourhood search (VNS)	A_3	Neighborhood-best
	A_4	First-best
Multistart local search (MLS)	A_5	Neighborhood operator O_1, neighbourhood-best
	A_6	Neighborhood operator O_2, neighbourhood-best
	A_7	Neighborhood operator O_3, neighbourhood-best
Algorithm portfolios	NoF	No forecasting, equally shared resources (plain portfolio)
	SMA	Simple moving average, $k = 1$
	SES	Simple exponential smoothing, $\alpha = 0.3$
	LES	Linear exponential smoothing, $\alpha = 0.3$, $\beta = 0.8$
Number of batches	$b_{\max}$	100 (1st stage), 50, 100, 150 (2nd stage)
Number of processing units	C	36 (1 master node and 35 slave nodes)
Number of function evaluations	$e_{\max}$	10^{10}
Number of experiments per case	ν	100

neighbourhood-best and the other one on first-best. Note that this algorithm uses all four neighbourhood operators, concurrently. Contrary to this, multistart local search assumes a single operator and its only search power stems from the neighbourhood search. For this reason, we considered three variants using the O_1, O_2, and O_3 neighbourhood operator, respectively, but all under the neighbourhood-best search scheme in order to enhance the effectiveness of local neighbourhood search.

All seven algorithms were incorporated into four heterogeneous algorithm portfolio schemes. Specifically, we considered a plain portfolio without performance forecasting, henceforth denoted as NoF, as the baseline for the assessment of our proposed approach. Its constituent algorithms are concurrently executed without any information exchange, equally sharing the available resources. We also considered three algorithm portfolios with performance forecasting, namely one for each forecasting technique. The portfolio using the simple moving average technique is henceforth denoted as SMA, the one with the simple exponential smoothing is denoted as SES, and the one with the linear exponential smoothing as LES. The shape of the forecasted performance curves, which are expected to rapidly change at the beginning and gradually decline as the execution of the algorithm proceeds, was considered as the guideline for the selection of the forecasting parameters. Thus, SMA assumed the random walk approach with predictions being made using only one previous observation, while $\alpha = 0.3$ was used for SES in Eqution (2), and $\alpha = 0.3$, $\beta = 0.8$ were used for LES in Equations (3)–(5).

The simulations were conducted on an heterogeneous Beowulf cluster consisting of sixth and seventh generation Intel® i7 processors. Each portfolio occupied $C = 36$ CPUs, namely 1 master node and 35 slave nodes. Initially, each one of the seven algorithms of the portfolio occupied five CPUs. In the NoF portfolio, this allocation was retained throughout the whole experiment, while in the rest of the portfolios the assignments were changed at each batch according to the forecasted performance, according to the procedures described in Section 3.

Each portfolio was assessed on the 12 CW(n, k^2) problems reported in Table 1, consisting of sequences of length $n = 33$–130 and weights $k^2 = 9$–49. All test problems are reported as solvable in [5] and they constitute hard permutation optimization tasks since the corresponding search spaces are huge. For each test problem, $\nu = 100$ independent experiments were conducted. The total computational budget per experiment was equal to $e_{\max} = 10^{10}$ function evaluations and the algorithms were restarted to a new random position after 10^3 evaluations with no improvement of the best solution. Finally, the computational budget was allocated in $b_{\max} = 100$ batches to the algorithms according to our presentation in Section 3. In a second stage of experiments, the best-performing portfolio was analysed also for $b_{\max} = 50$ and 150 batches to reveal the effect of this parameter on its performance.

Figure 2 illustrates the number of function evaluations per problem and portfolio. The boxplots illustrate the sample of the required function evaluations in 100 experiments per test problem. The main body of the box contains 50% of the observed values (25th–75th percentile), the horizontal middle line is the median of the sample, and notches define the corresponding confidence intervals for the median. Outliers are denoted as crosses outside of the sample's whiskers. The exact average values and standard deviation of the observed performances are reported in Table A1 of Appendix. According to the experimental data, the performance of the portfolios seems to be problem-dependent. Thus, a reasonable

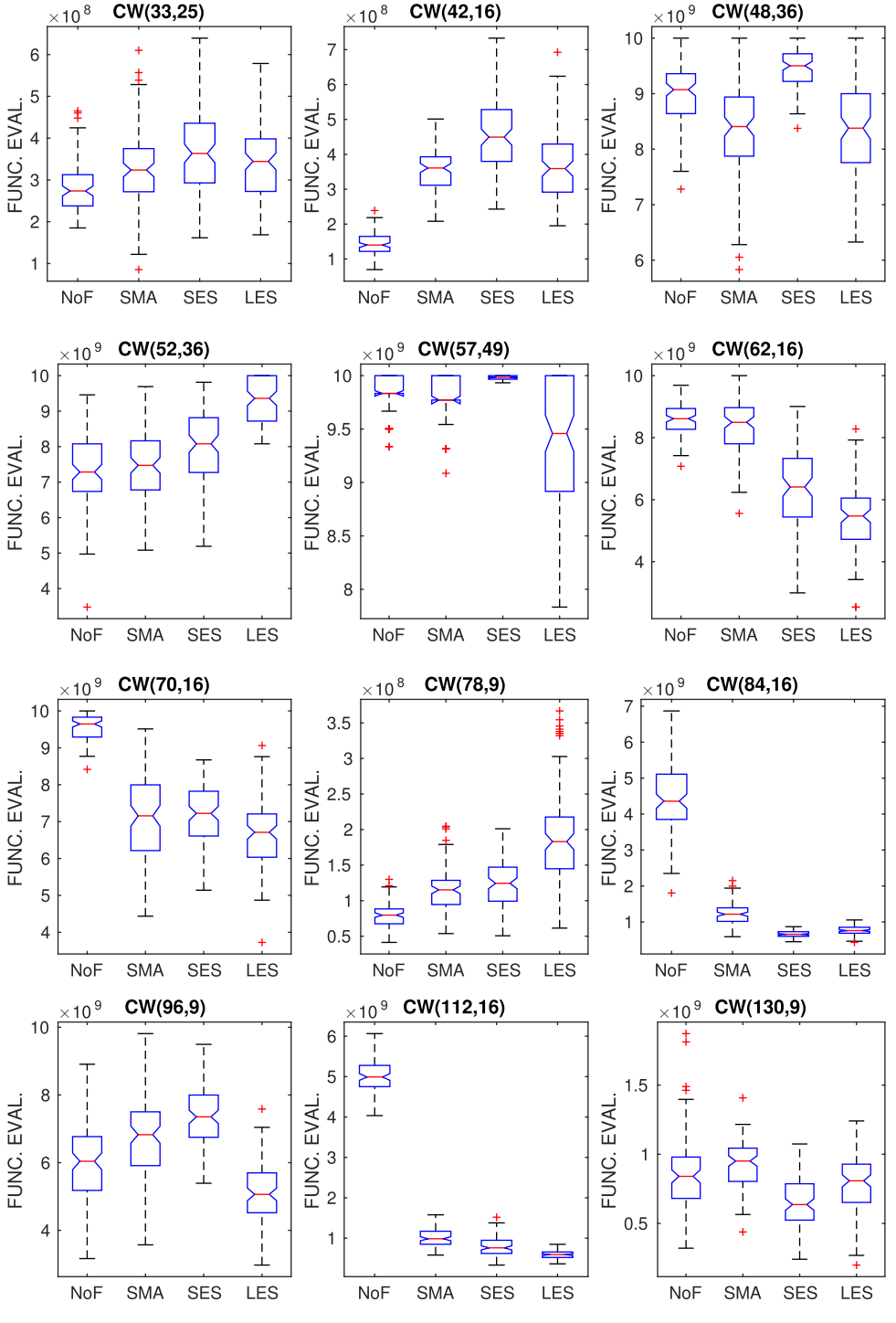

Figure 2. Number of the required function evaluations per problem and algorithm portfolio in 100 experiments.

validation of the portfolios can be considered only by averaging their performance over all test problems. Moreover, there are clear overlaps in the confidence intervals depicted in Figure 2, which may render some of the observed differences statistically insignificant.

For these reasons, we conducted Wilcoxon rank-sum tests for each pair of portfolios and test problem. A favourable comparison for portfolio A, i.e. smaller average number of function evaluations and statistically significant difference from a competitor portfolio B, was counted as a win for A and loss for B. Otherwise, a tie was counted for both of them. Obviously, for each one of the four considered portfolios, three comparisons were conducted per test problem. Thus, the maximum number of wins, losses, or ties for each portfolio, summed over all 12 test problems, was 36. Table 2 reports these values. As we can see, the more sophisticated the forecasting approach was, the best was the performance of the algorithm. The simplistic SMA approach where predictions were based only on the previous observed value was only slightly better than the plain NoF (no forecasting) portfolio. In fact, it achieved only one additional win and a few additional ties. However, the SES and LES portfolios doubled the number of wins, with LES exhibiting clear superiority.

A ranking scheme was considered for the portfolios, primarily based on their number of wins (in descending order), secondarily on the number of losses (in ascending order), and eventually according to the mean number of function evaluations (in ascending order), for each test problem. The ranks achieved by the portfolios averaged over all test problems are illustrated in Figure 3. Note that smaller ranks denote better approaches. The evidence verifies and firmly supports the previous findings, with LES occupying the first position, followed by SES. This clearly suggests that the use of both level and trend information of

Table 2. Number of wins, losses, and ties per portfolio summed over all test problems.

	Wins	Losses	Ties
NoF	9	25	2
SMA	10	19	7
SES	20	13	3
LES	25	7	4

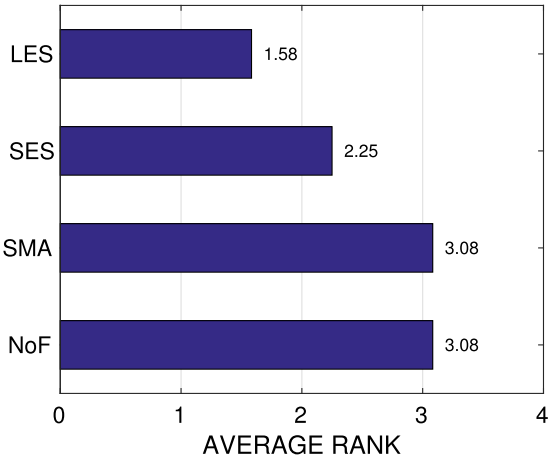

Figure 3. Average rank of each portfolio over all test problems (smaller values are better).

the performance signal is highly important for effective performance forecasting in the portfolios.

Another issue of interest was the constituent algorithm that eventually detected the global minimizer for each portfolio and test problem. In the upper part of Figure 4, the total number of times where each algorithm found the best solution per portfolio over all experiments is illustrated. In the lower part of the same figure, we can see the percentage of cases where each algorithm was the best in the portfolio (i.e. it was the one that detected the global minimizer). The algorithms are denoted as $A_1, \ldots, A_7$, according to the notation in Table 1. This data offers some useful insight. As we observe, there is an indisputable dominance of tabu search with neighbourhood-best search (algorithm A_1) against the rest of the algorithms in all portfolios. This verifies previous results on the CW(n, k^2) problems, which suggested that tabu search is probably the most efficient algorithm for this problem type [10,26,38]. The pie chart reveals that in more than half of the experiments the specific algorithm was the one that attained the best solution. Interestingly, the same tabu search but with first-best neighbourhood search (algorithm A_2) failed in attaining optimal

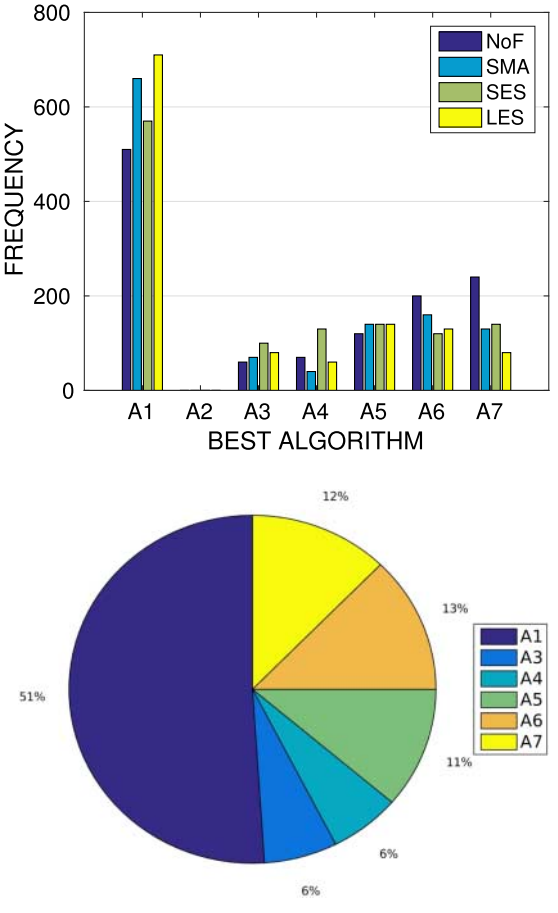

Figure 4. Up: Number of times where each algorithm found the best solution per portfolio over all experiments. Down: Rounded percentage of cases where each algorithm was the best in the portfolio.

solutions in all cases. This indicates that it is highly probable to overshoot the optimal solution if the neighbourhood is only partially probed with the specific neighbourhood search operator (i.e. the interchange O_1 operator).

In contrast to tabu search, the variable neighbourhood search (algorithms A_3 and A_4) seems to be only mildly affected from the use of either neighbourhood-best or first-best search in the neighbourhoods. Indeed, the pie chart in Figure 4 shows that they attained the best solution for the same fraction of experiments. This is a direct consequence of the use of all four neighbourhood operators, which counterbalances the possible overshooting of the solution. Thus, in the neighbourhood-best case the algorithm converges slower than tabu search because the computational budget is shared among operators $O_1, \ldots, O_4$. In the first-best case, the problem of overshooting the solution is evidently counterbalanced by the use of the alternative operators in the same neighbourhood. Hence, first-best becomes more efficient in variable neighbourhood search than in different algorithms. Eventually, their overall performance is almost identical. However, it shall be noted that some performance fluctuations are still observed for the two algorithms from one portfolio to another, as we can see in the upper part of Figure 4.

Finally, the multistart local search (algorithms A_5, A_6, and A_7), which all adopted the neighbourhood-best search with their corresponding operator (i.e. O_1, O_2, and O_3), attained global minimizers almost twice as frequently as the variable neighbourhood search approaches. However, although A_5 adopted the same neighbourhood operator as the tabu search algorithm, it exhibited significantly inferior performance. This can be attributed to the lack of tabu list in the multistart local search, and connotes that cyclic moves are highly probable under the interchange operator in the specific problem type.

Table 3. ANOVA table for comparisons between the algorithm.

Source	SS	df	MS	*F*	Prob> *F*
Columns	4.229776681894434e+19	2	2.114888340947217e+19	1.4908	0.22534
Error	5.102946250301801e+22	3597	1.418667292271838e+19		
Total	5.107176026983696e+22	3599			

Note: SS: sum of squares, df: degrees of freedom, MS: mean squares, F: F ratio.

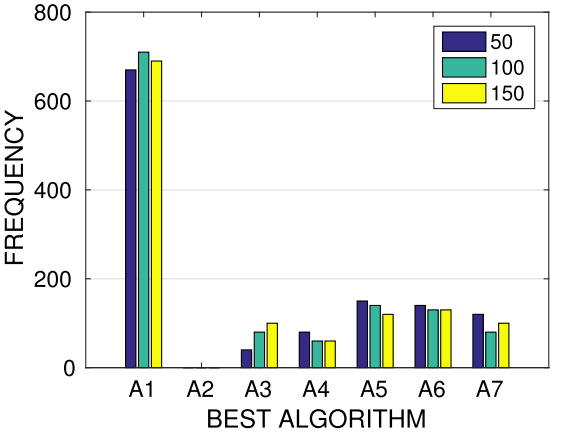

Figure 5. Number of times where each algorithm found the best solution for the LES portfolio with different number of batches $b_{max} = 50$, 100, and 150.

So far we showed that LES is the best-performing portfolio. We shall now concentrate our attention on it. As we have seen, the main parameter of the proposed forecasting-based portfolio model is the number of batches $b_{\max}$. This is the only user-defined parameter that is irrelevant to the externally available resources. The experimental results presented so far were based on $b_{\max} = 100$. In order to investigate the robustness of the algorithm, we repeated the experiments for the best-performing LES portfolio also for $b_{\max} = 50$ and 150. Then, the results for all three cases were compared through ANOVA and the relevant output is reported in Table 3. The obtained p-value, $p = .22534$, implies the lack of significant differences between the three different batch settings. Thus, additionally to its efficiency, LES is characterized also by robustness. Finally, Figure 5 displays the effect of the number of batches on the number of times where each algorithm found the best solution. The similarity of the results with that of Figure 4 comes as a consequence of the robustness of LES with respect to this parameter.

5. Conclusions

We proposed a new forecasting-based algorithm portfolio model. Time series forecasting is employed to predict the performance of the portfolio's constituent algorithms. Then, computational resources (processing units) are allocated to the algorithms, accordingly.

The challenging problem of detecting circulant weighing matrices was selected as our testbed for the empirical analysis of the proposed approach. Twelve problems of various sequence lengths and weights were used for this purpose. Moving average and exponential smoothing techniques were used with heterogeneous parallel algorithm portfolios based on three state-of-the-art algorithms, namely tabu search, variable neighbourhood search, and multistart local search.

The experimental evidence and statistical analysis suggest that the proposed portfolios are very competitive to the standard ones, especially when sophisticated forecasting models are considered. Moreover, the distinguished portfolios exhibited remarkable robustness to the only user-defined parameter that is irrelevant to the externally available resources.

Further work will consider large-scale application of the proposed approaches on the open circulant weighing matrices problems. Also, the online adaptation of the employed forecasting model and parameters will significantly enhance its performance and broaden its applicability.

Acknowledgements

The authors would like to thank the editor and the anonymous reviewers for their valuable comments and suggestions.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research was conducted at the National Research University Higher School of Economics and supported by RSF grant 14-41-00039. The authors would like to thank the Shared Hierarchical

Academic Research Computing Network (SHARCNET) and the WestGrid HPC consortium for supporting their research. Panos M. Pardalos is partially supported by the Laboratory of Algorithms and Technologies for Network Analysis.

References

[1] R. Akay, A. Basturk, A. Kalinli, and X. Yao, *Parallel population-based algorithm portfolios: an empirical study*, Neurocomputing 247 (2017), pp. 115–125.
[2] A. Almakhlafi and J. Knowles, *Systematic construction of algorithm portfolios for a maintenance scheduling problem*, IEEE Congress on Evolutionary Computation (CEC), Cancun, Mexico, 2013, pp. 245–252.
[3] M. H. Ang, K.T. Arasu, S. Lun Ma. and Y. Strassler, *Study of proper circulant weighing matrices with weigh 9*, Discrete. Math. 308 (2008), pp. 2802–2809.
[4] K.T. Arasu and T.A. Gulliver, *Self-dual codes over Fp and weighing matrices*, IEEE Trans. Inf. Theory 47 (2001), pp. 2051–2055.
[5] K.T. Arasu and A.J. Gutman, *Circulant weighing matrices*, Cryptogr. Commun. 2 (2010), pp. 155–171.
[6] K.T. Arasu, J.F. Dillon, D. Jungnickel, and A. Pott, *The solution of the Waterloo problem*, J. Comb. Theory A 71 (1995), pp. 316–331.
[7] K.T. Arasu, K.H. Leung, S.L. Ma, A. Nabavi, and D.K. Ray-Chaudhuri, *Determination of all possible orders of weight 16 circulant weighing matrices*, Finite Fields Appl. 12 (2006), pp. 498–538.
[8] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung, *Time Series Analysis: Forecasting and Control*, John Wiley & Sons, Hoboken, 2015.
[9] C. Chatfield, *Time-Series Forecasting*, CRC Press, Boca Raton, 2000.
[10] M. Chiarandini, I.S. Kotsireas, C. Koukouvinos, and L. Paquete, *Heuristic algorithms for hadamard matrices with two circulant cores*, Theor. Comput. Sci. 407 (2008), pp. 274–277.
[11] P. Eades, *On the existence of orthogonal designs*, Ph.D. diss., Australian National University, 1997.
[12] P. Eades and R.M. Hain, *On circulant weighing matrices*, Ars Combinatoria 2 (1976), pp. 265–284.
[13] E.S. Gardner, *Exponential smoothing: the state of the art*, J. Forecast. 4 (1985), pp. 1–28.
[14] E.S. Gardner, *Exponential smoothing: the state of the art–part II*, Int. J. Forecast. 22 (2006), pp. 637–666.
[15] M. Gendreau and J.Y. Potvin, *Tabu search*, in *Handbook of Metaheuristics*, M. Gendreau and J.Y. Potvin, eds., Springer, New York, 2010, pp. 41–59.
[16] A.V. Geramita and J. Sebery, *Orthogonical Designs: Quadratic Forms and Hadamard Matrices*, Lecture Notes in Pure Appl. Mathematics Vol. 45, Marcel Dekker, Inc., New York, 1979.
[17] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Comput. Oper. Res. 13 (1986), pp. 533–549.
[18] F. Glover, *Tabu search – part I*, ORSA J. Comput. 1 (1989), pp. 190–206.
[19] F. Glover, *Tabu search – part II*, ORSA J. Comput. 2 (1990), pp. 4–32.
[20] C.P. Gomes and B. Selman, *Algorithm portfolio design: theory vs. practice*, Thirteenth Conference on Uncertainty in Artificial Intelligence, Providence, RI, USA, 1997, pp. 190–197.
[21] P. Hansen, N. Mladenovic, J. Brimberg and J.A. Moreno Pérez, Variable neighborhood search, in *Handbook of Metaheuristics*, M. Gendreau and J.Y. Potvin, eds., Springer, New York, 2010, pp. 61–86.
[22] B.A. Huberman, R.M. Lukose and T. Hogg, *An economics approach to hard computational problems*, Science 27 (1997), pp. 51–53.
[23] R.J. Hyndman, Moving averages, in *International Encyclopedia of Statistical Science*, M. Lovric, ed., Springer, Berlin, 2011, pp. 866–869.
[24] I.S. Kotsireas, Algorithms and metaheuristics for combinatorial matrices, in *Handbook of Combinatorial Optimization*, P.M. Pardalos, D.Z. Du, and R.L. Graham, eds., Springer, New York, 2013, pp. 283–309.

[25] I.S. Kotsireas, C. Koukouvinos, P.M. Pardalos, and D.E. Simos, *Competent genetic algorithms for weighing matrices*, J. Comb. Optim. 24 (2012), pp. 508–525.
[26] I.S. Kotsireas, P.M. Pardalos, K.E. Parsopoulos, and D. Souravlias, On the solution of circulant weighing matrices problems using algorithm portfolios on multi-core processors, in *Experimental Algorithms*, A.V. Goldberg and A.S. Kulikov, eds., Springer, Cham, 2016, pp. 184–200.
[27] C. Koukouvinos and J. Seberry, *Weighing matrices and their applications*, J. Stat. Plan. Inference 62 (1997), pp. 91–101.
[28] H.R. Lourenço, O.C. Martin, and T. Stützle, Iterated local search: framework and applications, in *Handbook of Metaheuristics*, M. Gendreau and J.Y. Potvin, eds., Springer, New York, 2010, pp. 363–397.
[29] J.M. Lucas and M.S. Saccucci, *Exponentially weighted moving average control schemes: properties and enhancements*, Technometrics 32 (1990), pp. 1–12.
[30] S. Makridakis, S.C. Wheelwright, and R.J. Hyndman, *Forecasting Methods and Applications*, John Wiley & Sons, 2008.
[31] N. Mladenovic and P. Hansen, *Variable neighborhood search*, Comput. Oper. Res. 24 (1997), pp. 1097–1100.
[32] M.A. Muñoz and M. Kirley, *ICARUS: identification of complementary algorithms by uncovered sets*, IEEE Congress on Evolutionary Computation (CEC), Vancouver, Canada, 2016, pp. 2427–2432.
[33] M.A. Muñoz, M. Kirley, and S.K. Halgamuge, The algorithm selection problem on the continuous optimization domain, in *Computational Intelligence in Intelligent Data Analysis*, C. Moewes and A. Nürnberger, eds., Springer, Heidelberg, 2013, pp. 75–89.
[34] F. Peng, K. Tang, G. Chen, and X. Yao, *Population-based algorithm portfolios for numerical optimization*, IEEE Trans. Evol. Comput. 14 (2010), pp. 782–800.
[35] J.R. Rice, *The algorithm selection problem*, Adv. Comput. 15 (1976), pp. 65–118.
[36] S.W. Roberts, *Control chart tests based on geometric moving averages*, Technometrics 1 (1959), pp. 239–250.
[37] D. Souravlias, K.E. Parsopoulos and E. Alba, Parallel algorithm portfolio with market trading-based time allocation, in *Operations Research Proceedings 2014*, M. Lübbecke, A. Koster, P. Letmathe, R. Madlener, B. Peis, and G. Walther, eds., Springer, 2014, pp. 567–574.
[38] D. Souravlias, K.E. Parsopoulos, and I.S. Kotsireas, *Circulant weighing matrices: a demanding challenge for parallel optimization metaheuristics*, Optim. Lett. 10 (2016), pp. 1303–1314.
[39] Y. Strassler, *The classification of circulant weighing matrices of weight 9.*, Ph.D. diss., Bar-Ilan University, 1997.
[40] W. van Dam, *Quantum algorithms for weighing matrices and quadratic residues*, Algorithmica 34 (2002), pp. 413–428.
[41] J.A. Vrugt, B.A. Robinson, and J.M. Hyman, *Self-adaptive multimethod search for global optimization in real-parameter spaces*, IEEE Trans. Evol. Comput. 13 (2009), pp. 243–259.
[42] D.H. Wolpert and W.G. Macready, *No free lunch theorem for optimization*, IEEE Trans. Evol. Comput. 1 (1997), pp. 67–82.
[43] S. Yuen, C. Chow and X. Zhang, *Which algorithm should I choose at any point of the search: an evolutionary portfolio approach*, 15th Annual Conference on Genetic and Evolutionary Computation (GECCO), New York, NY, USA, 2013, pp. 567–574.
[44] G.I. Zobolas, C.D. Tarantilis, and G. Ioannou, *Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm*, Comput. Oper. Res. 36 (2009), pp. 1249–1267.

Appendix

Table A1 reports the mean and standard deviation of the required number of function evaluations per portfolio and test problem in 100 experiments.

Table A1. Mean and standard deviations of the required number of function evaluations per portfolio and test problem in 100 experiments.

Problem		NoF	SMA	SES	LES
CW(33, 25)	Mean	265,720,142.52	331,835,339.83	373,491,176.26	333,194,618.75
	St.d.	176,831,228.50	326,614,780.47	307,813,895.15	271,396,704.86
CW(42, 16)	Mean	148,021,251.04	355,494,012.95	465,529,464.80	366,757,922.58
	St.d.	87,457,702.84	197,920,978.26	337,829,868.56	283,411,559.00
CW(48, 36)	Mean	8,966,094,259.03	8,450,373,663.21	9,522,177,083.09	8,390,176,940.68
	St.d.	1,628,783,111.09	2,534,472,164.43	941,549,683.09	2,623,454,572.03
CW(52, 36)	Mean	7,015,380,106.43	7,595,344,052.22	7,830,003,623.60	9,410,924,269.19
	St.d.	3,365,230,663.40	3,019,641,613.58	3,106,607,178.10	1,541,947,163.22
CW(57, 49)	Mean	9,845,380,201.73	9,789,828,627.24	9,983,252,680.50	9,431,244,647.27
	St.d.	347,688,544.57	554,744,648.62	41,628,397.33	1,362,108,847.45
CW(62, 16)	Mean	8,552,745,227.36	8,158,027,372.19	6,086,237,122.05	5,169,643,847.51
	St.d.	1,717,529,879.00	2,502,034,113.35	3,920,314,169.99	3,209,578,302.15
CW(70, 16)	Mean	9,535,375,684.45	7,017,255,759.94	7,203,123,536.62	6,464,505,353.24
	St.d.	1,085,967,189.42	3,967,903,304.55	2,345,619,436.51	3,235,329,077.92
CW(78, 9)	Mean	78,308,049.92	118,582,116.69	115,282,916.32	182,774,996.53
	St.d.	50,711,965.22	88,824,810.13	101,032,082.98	180,481,149.01
CW(84, 16)	Mean	4,429,933,493.29	1,235,246,337.81	654,610,865.80	822,054,389.19
	St.d.	2,642,499,036.43	946,674,438.16	295,086,758.98	411,906,987.14
CW(96, 9)	Mean	5,899,585,917.93	6,820,004,192.95	7,494,804,612.15	5,024,175,677.24
	St.d.	3,467,027,798.10	3,615,213,497.65	2,769,214,372.77	3,043,546,751.84
CW(112, 16)	Mean	5,084,800,143.27	995,875,152.49	754,864,358.78	575,018,440.46
	St.d.	1,255,391,314.54	633,616,476.51	755,194,567.19	304,936,921.11
CW(130, 9)	Mean	881,997,544.89	900,147,336.62	649,903,262.95	795,164,721.09
	St.d.	775,450,398.34	487,501,928.09	629,421,473.56	714,179,077.04