On the Solution of Circulant Weighing Matrices Problems Using Algorithm Portfolios on Multi-core Processors

Ilias S. Kotsireas[1], Panos M. Pardalos[2], Konstantinos E. Parsopoulos[3(✉)], and Dimitris Souravlias[3]

[1] Department of Physics and Computer Science, Wilfrid Laurier University, Waterloo, ON, Canada
ikotsire@wlu.ca
[2] Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL, USA
pardalos@ufl.edu
[3] Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece
{kostasp,dsouravl}@cse.uoi.gr

Abstract. Research on the existence of specific classes of combinatorial matrices such as the Circulant Weighing Matrices (CWMs) lies in the core of diverse theoretical and computational efforts. Modern metaheuristics have proved to be valuable tools for solving such problems. Recently, parallel Algorithm Portfolios (APs) composed of established search algorithms and sophisticated resource allocation procedures offered significant improvements in terms of time efficiency and solution quality. The present work aims at shedding further light on the latent quality of parallel APs on solving CWM problems. For this purpose, new AP configurations are considered along with specialized procedures that can enhance their performance. Experimental evaluation is conducted on a computationally restrictive, yet widely accessible, multi-core processor computational environment. Statistical analysis is used to reveal performance trends and extract useful conclusions.

Keywords: Algorithm Portfolios · Circulant Weighing Matrices · Computational optimization · Multi-core processors

1 Introduction

Combinatorial matrices are involved in various significant applications ranging from statistical experimentation to coding theory and quantum information processing [3,8,23]. Special types of combinatorial matrices have been extensively investigated. Circulant Weighing Matrices (CWMs) constitute an important class in this framework. The existence of finite or infinite classes of CWMs has been the core subject in several theoretical works [2,4,9,10,12].

A.V. Goldberg and A.S. Kulikov (Eds.): SEA 2016, LNCS 9685, pp. 184–200, 2016.
DOI: 10.1007/978-3-319-38851-9_13

Metaheuristics have proved to be very useful in cases where theoretical approaches have not provided adequate insight. The application of metaheuristics requires the transformation of the CWM existence problem to a combinatorial optimization task [6,7,18,19,21,22]. Recently, prevailing metaheuristics have been used in the Algorithm Portfolios (APs) framework [16,17] for solving CWM problems in parallel computational environments [28]. Sophisticated resource allocation schemes based on market trading procedures were used in those approaches, achieving high standards of performance. The provided results suggested that APs can remarkably enhance the time performance and quality of solution of their constituent algorithms in CWM problems [28]. Also, they verified the domination of trajectory-based approaches against population-based stochastic algorithms.

The present work aims at extending the previous studies by offering further insight regarding the performance of interactive and non-interactive parallel APs. Based on previous findings, the established Tabu Search (TS) algorithm and the previously unused Iterated Local Search (ILS) approach compose the considered APs. Additionally, a sequence-comparison scheme that prevents TS from revisiting classes of equivalent sequences is introduced.

The experimental evaluation of the APs is conducted on a low-specification parallel hardware, i.e., a common multi-core processor, in contrast to the abundant grid-computing environment of previous studies [28]. The overall performance of the APs is investigated in terms of time efficiency and solution quality on two representative CWM problems. Additionally, the impact of the number of concurrently running algorithms on the overall performance is investigated. Diverse homogeneous and heterogeneous APs with various parameter configurations are also considered.

The rest of the paper is structured as follows: Sect. 2 formulates the CWM problem as a combinatorial optimization task. The employed individual algorithms and APs are described in Sect. 3. Experimental analysis is reported in Sect. 4, and the paper concludes in Sect. 5.

2 Circulant Weighing Matrices

Circulant Weighing Matrices (CWMs) [4] refer to a special type of combinatorial matrices. A square $n \times n$ matrix W defined as,

$$W = (w_{ij}), \qquad w_{ij} \in \{-1, 0, 1\}, \qquad i, j = 1, \ldots, n,$$

is a CWM of *order* n and *weight* k^2, denoted as $CW(n, k^2)$, if it satisfies the condition,

$$W\, W^{\top} = k\, I_n,$$

where I_n is the identity matrix of size n, and $W^{\top}$ is the transpose of W. Thus, a CWM is primarily a *weighing matrix*. Additionally, each row of a CWM, except the first one, is obtained through a right cyclic shift of its preceding row. Hence, the complete matrix can be fully defined by its first row. A significant amount

of research has been devoted to theoretical and experimental investigations on the existence of CWMs of various orders and weights [1,5,10,29].

Metaheuristics have been employed in cases where theoretical efforts have been fruitless. In these cases, the problem is solved as a permutation optimization one, aiming at the detection of the first row of the considered CWM type. The underlying objective function is based on the concept of *Periodic Autocorrelation Function* (PAF) [20]. The defining row of a CWM is a ternary sequence,

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \{-1, 0, +1\}^n,$$

of length n, and its PAF values are defined as,

$$PAF_{\boldsymbol{x}}(s) = \sum_{i=1}^{n} x_i \, x_{i+s}, \qquad s = 1, 2, \ldots, \left\lceil \frac{n}{2} \right\rceil . \tag{1}$$

CWMs with zero PAF values have special research interest [19,20,28]. In addition, it has been proved that admissible sequences have exactly k^2 non-zero components, with $k(k+1)/2$ components being equal to $+1$ and $k(k-1)/2$ components assuming the -1 value.

Let $S_{(n,k)}$ be the search space that contains all admissible ternary sequences that define CWMs of order n and weight k^2. Then, the objective function of the corresponding combinatorial optimization problem is defined as,

$$\min_{\boldsymbol{x} \in S_{(n,k)}} f(\boldsymbol{x}) = \sum_{s=1}^{\lceil \frac{n}{2} \rceil} |PAF_{\boldsymbol{x}}(s)| = \sum_{s=1}^{\lceil \frac{n}{2} \rceil} \left| \sum_{i=1}^{n} x_i \, x_{i+s} \right|, \tag{2}$$

where $i+s$ is taken modulo n when $i+s > n$. Obviously, the global minimizer of this optimization problem is a sequence with zero PAF values for all s. Experimental evidence has shown that the difficulty of a $CW(n, k^2)$ problem increases with the order n (length of sequence) and, particularly, with the weight k^2.

3 Employed Algorithms

In the following paragraphs, we briefly describe the employed individual algorithms as well as the considered APs. For presentation purposes, we assume that the considered optimization problem is given in the general form,

$$\min_{\boldsymbol{x} \in S} \; f(\boldsymbol{x}),$$

where S in the corresponding search space.

3.1 Iterated Local Search

Iterated Local Search (ILS) defines a simple and straightforward framework for probing complex search spaces. Its main requirement is the use of a suitable

Table 1. Pseudocode of the ILS algorithms.

	Iterated Local Search (ILS)
1 :	$\boldsymbol{x}_{\text{ini}} \leftarrow \mathit{GetInitialSequence}(S)$
2 :	$\boldsymbol{x}^* \leftarrow \mathit{LocalSearch}(\boldsymbol{x}_{\text{ini}})$
3 :	$S^* \leftarrow \{\boldsymbol{x}^*\}$
4 :	**While** (not stopping) **Do**
5 :	**If** $(\mathit{rand}() < \rho)$ **Then**
6 :	$\boldsymbol{x}_{\text{ini}} \leftarrow \mathit{GetInitialSequence}(S^*)$
7 :	**Else**
8 :	$\boldsymbol{x}_{\text{ini}} \leftarrow \mathit{GetInitialSequence}(S)$
9 :	**End If**
10 :	$\boldsymbol{x}^* \leftarrow \mathit{LocalSearch}(\boldsymbol{x}_{\text{ini}})$
11 :	$S^* \leftarrow S^* \cup \{\boldsymbol{x}^*\}$
12 :	**End While**
13 :	$\boldsymbol{x}_{\text{best}} \leftarrow \arg\min_{\boldsymbol{x}^* \in S^*} f(\boldsymbol{x}^*)$
14 :	**Report** $\boldsymbol{x}_{\text{best}}$

local search procedure for the problem at hand. The local search is initiated to a randomly selected sequence $\boldsymbol{x}_{\text{ini}}$ and generates a trajectory that eventually reaches the nearest local minimizer $\boldsymbol{x}^*$. This is achieved by iteratively selecting downhill moves within the close neighborhood of the current sequence.

In discrete spaces such as the ones in the studied CWM problems, the close neighborhood of a sequence is defined as the finite set of sequences with the smallest possible distance from it. Typically, Hamming distance is used for this purpose. The local search procedure usually scans the whole neighborhood of the current sequence and makes the move with the highest improvement (*neighborhood-best* strategy). Alternatively, it can make a move to the first improving sequence found in the neighborhood (*first-best* strategy). The detected local minimizer is archived in a set S^*. Then, a new trajectory is started from a new initial sequence [24].

In its simplest form, ILS generates new trajectories by randomly sampling new initial sequences in the search space according to a (typically Uniform) distribution. This is the well-known *Random Restarts* approach. The most common stopping criteria are the detection of a prespecified number of local minimizers or a maximum computational budget in terms of running time or function evaluations. Although random restarts were shown to be sufficient in various problems, relevant research suggests that efficiency can be increased if already detected local minimizers from the set S^* are exploited during the search [24]. Typically, this refers to the generation of new initial sequences by perturbing previously detected local minimizers.

The two initialization approaches can also be combined. Naturally, this scheme introduces new parameters to the algorithm. Specifically, the user needs

Table 2. Pseudocode of the TS algorithms.

Tabu Search (TS)	
1 :	$TL \leftarrow \emptyset$
2 :	$\boldsymbol{x} \leftarrow GetInitialSequence(S)$
3 :	$UpdateTabuList(TL, \boldsymbol{x})$
4 :	$\boldsymbol{x}_{\text{best}} \leftarrow \boldsymbol{x}$
5 :	**While** (not stopping) **Do**
6 :	$\quad \boldsymbol{x}' \leftarrow ProbeNeighborhood(N(\boldsymbol{x}), TL)$
7 :	$\quad UpdateTabuList(TL, \boldsymbol{x}')$
8 :	$\quad$ **If** $(f(\boldsymbol{x}') < f(\boldsymbol{x}_{\text{best}}))$ **Then**
9 :	$\quad\quad \boldsymbol{x}_{\text{best}} \leftarrow \boldsymbol{x}'$
10 :	$\quad$ **End If**
11 :	$\quad$ **If** (trajectory termination) **Then**
12 :	$\quad\quad \boldsymbol{x} \leftarrow NewInitialSequence(S, \boldsymbol{x}_{\text{best}}, \rho)$
13 :	$\quad$ **End If**
14 :	**End While**
15 :	**Report** $\boldsymbol{x}_{\text{best}}$

to specify a probability $\rho \in [0, 1]$ of using perturbation-based restarts as well as the criteria for selecting the local minimizers from the set S^*.

The ILS algorithm is given in pseudocode in Table 1. Each call of *rand*() returns a real-valued random number in the range $[0, 1]$, while the function *GetInitialSequence*() implements the sampling procedures for the search space S and the set S^*. For a comprehensive presentation of ILS the reader is referred to [24].

3.2 Tabu Search

Tabu Search (TS) is among the most popular and well-studied metaheuristics. Since its formal introduction in [13,14], TS has been applied on numerous problems spanning various fields of discrete optimization [11,15,26]. The basic motivation for the development of TS originated from the necessity of search algorithms to overcome local minimizers. This was achieved by equipping the algorithms with descent and hill-climbing capabilities.

In descent mode, the local search procedure of TS follows the baseline of the ILS approach described in the previous section. After the detection of a local minimizer, the algorithm begins ascending by reversing from downhill to uphill moves in the neighborhood $N(\boldsymbol{x})$ of the current sequence $\boldsymbol{x}$. This continues until a local maximizer is reached. Subsequently, a new descent phase takes place etc.

In order to avoid retracing the same trajectories, a memory structure that stores the most recent moves and prevents the algorithm from revisiting them is

used in TS. In practice, the memory comprises a finite list structure, also called *tabu list* (TL), where the most recently visited sequence replaces the oldest one.

The use of memory cannot fully prevent TS from getting trapped in misleading trajectories that drive the search away from global minimizers. In such cases, it is beneficial to restart the algorithm on a new sequence if the current trajectory has not improved the best solution for a prespecified number of iterations or elapsed time.

Similarly to ILS, new initial sequences can be generated either randomly within the whole search space S or through perturbations of already detected local minimizers. The latter approach can be effective in problems where local minimizers are closely clustered.

A simple form of the TS algorithm is reported in Table 2, where the parameter $\rho \in [0, 1]$ defines the probability of restarting the algorithm on a perturbation of the best-so-far solution $\boldsymbol{x}_{\text{best}}$. Other crucial parameters are the size of the tabu list, s_{tabu}, as well as the number of non-improving steps, T_{nis}, before restarting a trajectory. Further details on TS and its applications can be found in [11,15,26].

3.3 Algorithm Portfolios

Algorithm Portfolios (APs) [17] define schemes composed of multiple individual algorithms that share the available computational budget. An AP may consist of multiple copies of one algorithm with the same or different parameters (*homogeneous AP*) or different algorithms (*heterogeneous AP*). All the algorithms run concurrently in either one or multiple processors (CPUs). If a single CPU is used, the algorithms' execution is alternated according to a time assignment schedule. In multi-core or parallel systems, the algorithms share the hardware resources, i.e., number of available CPUs [16].

Relevant studies have shown that proper resource allocation between the constituent algorithms can render APs more efficient than the standalone algorithms, both in serial [17] and parallel [16] computational environments. Moreover, information exchange between the algorithms (*interactive APs*) can be highly beneficial [25]. Motivated from these studies, a new parallel AP with a sophisticated time budget allocation scheme that is based on market-trading procedures was proposed in [27] and successfully applied on the CWM problems in [28]. The specific AP comprised various search algorithms. Among them, TS was shown to be the most effective one [28].

The previous studies offered useful insight on the performance of APs on CWM problems, leaving prosperous ground for further investigation. The AP in [28] was based on the special trading-based time allocation rather than the plain interactive AP model. The experimental results offered clear evidence that trajectory-based approaches were dominant in terms of solution quality. Moreover, the highly-effective TS algorithm was considered only with the neighborhood best strategy, which is an effective but also computationally demanding approach.

Another important issue in parallel APs is the effect of the number of algorithms and, consequently, the number of nodes that are concurrently used. The experiments in [28] were conducted on a computer cluster where a large number of processors were available. However, it would be interesting to evaluate the APs also on the widely accessible multi-core processors, which typically offer only a small number of CPUs to the user. For instance, modern Intel© i7 processors consist of 4 actual cores that offer 8 CPUs by using hyper-threading technology[1]. Each CPU can concurrently run multiple algorithms in different computation threads at the cost of slower execution, since the algorithms are alternatively executed. Given a prespecified running-time budget, it is compelling to investigate whether it is preferable to use small number of algorithms (not exceeding the number of available CPUs) in order to attain faster execution or use higher numbers of algorithms (thereby promoting exploration) with slower execution.

Another interesting issue that emanates from previous TS applications is related to the criteria of accepting a new sequence through comparisons with the ones stored in TL. The typical comparison has been based solely on the Hamming distance between the compared sequences, i.e., a pairwise comparison of their corresponding components. Thus, a new sequence was accepted only if it had non-zero distance from all stored sequences in TL. Although this approach adheres to the typical rules applied in various TS applications, it can become inefficient in CWM problems.

The reason lies on the specific properties of CWM matrices. Specifically, a given sequence $\boldsymbol{x}$ defines the same CWM with all right-hand shifted sequences produced from it. In simple words, the sequence $\boldsymbol{x}$ defines a whole class of sequences that produce the same CWM. These equivalent sequences have non-zero Hamming distances between them. Thus, the comparison criterion in previous TS approaches cannot prevent the acceptance of a sequence that is equivalent with one already included in TL. Tabu lists of large size as in [28] can ameliorate this deficiency but they impose additional computational burden. For this reason, it is preferable to modify the comparison procedure such that a candidate new sequence is accepted only if it differs from all sequences in TL as well as from all their right-hand shifts.

The present work attempts to shed light on the aforementioned issues. The employed parallel APs are outlined in Table 3. The number of nodes, m, refers to the number of threads required by the AP and can exceed the number of available CPUs. The parallel AP is based on a standard master-slave parallelization model, where the master (node 0) is devoted to book-keeping and information-sharing between the algorithms. Both homogeneous and heterogeneous APs consisting of the TS and ILS algorithms are studied. The simple Random Restart variant of ILS was used, along with the local search described in the previous sections. Further details for the algorithms are given in the following section.

[1] http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html.

Table 3. Pseudocode of the parallel Algorithm Portfolio approach with m nodes.

	Algorithm Portfolio (m nodes)
	Master Node ($i = 0$)
1 :	**Initialize** $(m-1)$ slave nodes and assign an algorithm to each one.
2 :	$\boldsymbol{x}_{\text{best}} \leftarrow GetInitialSequence(S)$
3 :	$SendSequence(i, \boldsymbol{x}_{\text{best}})$, $i = 1, \ldots, m-1$
3 :	**While** (nodes still running) **Do**
4 :	$GetMessage(i)$
5 :	**If** (node i improved $\boldsymbol{x}_{\text{best}}$) **Then**
6 :	$UpdateBest(\boldsymbol{x}_{\text{best}})$
7 :	**Else If** (node i requests best sequence update) **Then**
8 :	$SendSequence(i, \boldsymbol{x}_{\text{best}})$
9 :	**End If**
10 :	**End While**
11 :	**Report** $\boldsymbol{x}_{\text{best}}$
	Slave Nodes ($i = 1, \ldots, m-1$)
1 :	**Initialize** assigned algorithm.
2 :	$ReceiveSequence(0, \boldsymbol{x}_{\text{best}})$
3 :	**While** (allocated time not exceeded) **Do**
4 :	**Execute** one iteration of the algorithm.
5 :	**If** (new $\boldsymbol{x}_{\text{best}}$ found) **Then**
6 :	$SendSequence(0, \boldsymbol{x}_{\text{best}})$
7 :	**Else If** (best sequence update is needed) **Then**
8 :	$RequestSequenceUpdate(0)$
9 :	**End If**
10 :	**End While**
11 :	**Finalize** node

4 Experimental Analysis

The experimental analysis consisted of two phases. In the first phase, all algorithms were applied on the representative 33-dimensional $CW(33, 25)$ problem, in order to statistically analyze their performance. The specific problem was selected due to its guaranteed solution existence, moderate size, high weight ($k^2 = 25$), and reasonable convergence times of the algorithms. The second phase consisted of the application of the best-performing algorithms on the more challenging 48-dimensional $CW(48, 36)$ problem. This is a well-studied problem that was used as benchmark in previous studies [28]. The number of sequence components that assume each value of the set $\{-1, 0, +1\}$ for both problems is reported in Table 4.

Table 4. Details of the considered representative problems.

Problem	Length	Weight	Dim.	+1	−1	0
$CW(33, 25)$	33	25	33	15	10	8
$CW(48, 36)$	48	36	48	21	15	12

Table 5. Parameter values for the employed algorithms.

Param.	Description	Value(s)
m	number of nodes (threads)	$\{8, 16, 64\}$
nss	neighborhood search strategy	{neighb. best (nb), first best (fb)}
s_{tabu}	tabu list size	$\{5, 10\}$
T_{nis}	non-improving iterations before restart	$\{100, 1000\}$
ρ	probability of perturbing best solution	$\{0.00, 0.01\}$
p_{type}	algorithm parameters' type	{fixed (f), random (r)}
$T_{\max}$	maximum running time	300 s

We considered APs composed solely of TS or ILS algorithms, henceforth denoted as "TS" or "ILS", respectively. Also, we considered mixed APs embracing both algorithms, henceforth denoted as "MIX". Initially, an extensive experimental study was conducted for all combinations (full factorial design) of the parameter values reported in Table 5. Specifically, for each portfolio type (TS, ILS, or MIX), we considered the cases of $m = 8$, 16, and 64 threads running on a single processor with 8 CPUs available (note that the number of slave nodes is $m - 1$). In TS-based APs all slave nodes were occupied by TS algorithms, while in ILS-based APs they were devoted to ILS. In MIX APs, the TS algorithm was assigned to the odd-indexed nodes $(1, 3, \ldots)$ and ILS algorithms were running on even-indexed nodes $(2, 4, \ldots)$.

All experiments were conducted on a single-processor Intel© i7-4770 3.40 GHz machine with 8 GB DDR3 RAM, providing 8 available CPUs under Ubuntu Linux 14.04. There was no suppression of the operating system procedures during the runs. For the parallelization, the Open MPI libraries were used with the GCC 4.8.4 compiler. All source codes were developed in the C programming language.

In the TS and ILS algorithms, both the neighborhood-best (nb) and first-best (fb) strategies were considered for neighborhood search. New trajectories were either initialized on random perturbations of the best-so-far solution with probability $\rho = 0.01$ or solely on random new points (denoted as $\rho = 0.00$). In the first case, mild perturbations of the best solution were used, consisting of 1 up to 3 distinct random swaps of the sequence's components.

The TS algorithms require some additional parameters. The tabu list size s_{tabu} in our experiments was set to rather small values, namely 5 and 10. These values are significantly smaller than in previous studies where it was set equal

Table 6. Results for the 3 best-performing approaches per AP type and number of nodes, as well as for the 5 overall best APs for the $CW(33, 25)$ problem. The "*" symbol denotes randomized-parameters APs and, if followed by a number, e.g., "*s", it denotes that the upper bound of the corresponding randomized parameter is s.

TS-based APs									
	m	*nss*	s_{tabu}	T_{nis}	ρ	p_{type}	Suc.(%)	Time	Loc. Min.
	8	*	5	*100	0.00	r	100.0	24.6(26.4)	14080.9
	8	fb	5	100	0.00	f	100.0	26.8(28.2)	17535.1
	8	*	10	*100	0.00	r	100.0	32.9(34.4)	10488.2
	16	*	5	*100	*0.01	r	100.0	25.7(27.4)	7574.2
	16	*	5	*100	0.00	r	100.0	31.6(26.5)	8937.5
	16	fb	5	100	0.00	f	100.0	38.4(30.2)	12494.7
	64	fb	5	100	0.00	f	100.0	25.0(25.5)	2032.3
	64	fb	5	100	0.01	f	100.0	25.5(23.8)	2100.9
	64	*	5	100	0.00	r	100.0	29.0(31.8)	2063.8
ILS-based APs									
	m	*nss*	s_{tabu}	T_{nis}	ρ	p_{type}	Suc.(%)	Time	Loc. Min.
	8	nb	-	-	0.00	f	100.0	11.0(14.6)	32512.7
	8	nb	-	-	0.01	f	100.0	9.6(11.7)	28486.7
	8	fb	-	-	0.00	f	100.0	6.6(4.8)	42418.4
	16	fb	-	-	0.01	f	100.0	2.8(4.3)	8762.9
	16	nb	-	-	0.00	f	100.0	8.5(9.0)	12413.9
	16	nb	-	-	0.01	f	100.0	12.2(11.7)	17587.3
	64	fb	-	-	0.00	f	100.0	4.2(4.5)	3447.3
	64	fb	-	-	0.01	f	100.0	4.3(5.3)	3388.9
	64	*	-	-	0.00	r	100.0	7.7(9.9)	3927.3
MIX APs									
	m	*nss*	s_{tabu}	T_{nis}	ρ	p_{type}	Suc.(%)	Time	Loc. Min.
	8	fb	5	100	0.00	f	100.0	10.3(12.1)	31847.9
	8	fb	10	100	0.01	f	100.0	9.9(10.2)	29267.2
	8	*	5	*100	*0.01	r	100.0	9.8(12.2)	20060.1
	16	fb	5	100	0.01	f	100.0	7.6(11.5)	12508.8
	16	fb	10	1000	0.01	f	100.0	8.3(11.8)	12792.8
	16	fb	10	100	0.01	f	100.0	7.6(7.5)	11695.5
	64	*	10	*1000	0.00	r	100.0	9.9(15.8)	2681.8
	64	fb	10	1000	0.00	f	100.0	8.0(6.1)	3318.4
	64	fb	5	100	0.00	f	100.0	9.1(7.9)	3967.3
OVERALL BEST APs									
Alg.	m	*nss*	s_{tabu}	T_{nis}	ρ	p_{type}	Suc.(%)	Time	Loc. Min.
ILS	16	fb	-	-	0.01	f	100.0	2.8(4.3)	8762.9
ILS	64	fb	-	-	0.01	f	100.0	4.3(5.3)	3388.9
ILS	64	fb	-	-	0.00	f	100.0	4.2(4.5)	3447.3
MIX	16	fb	5	100	0.01	f	100.0	7.6(11.5)	12508.8
MIX	16	fb	10	1000	0.01	f	100.0	8.3(11.8)	12792.8

to the length of the sequences (order of the CWM) [28]. The reduction was implied by the new scheme for comparisons between the current sequence and the stored ones in TL, as described in Sect. 3.3. The tolerance T_{nis} of non-improving moves before restart was set to 100 and 1000. Larger values of T_{nis} result in longer trajectories and, hence, better local exploration around recently visited minimizers. Smaller values promote global exploration because the algorithm is restarted more frequently. All combinations of the corresponding parameters were considered for each AP type.

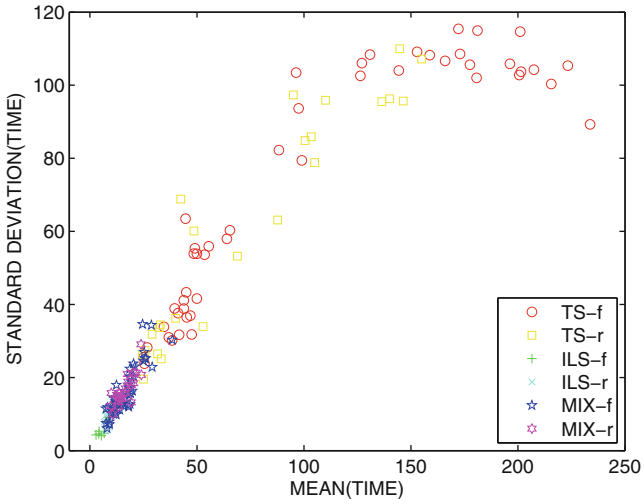

Fig. 1. Mean vs standard deviation of time required per AP type (TS, ILS, MIX) for fixed (-f) and random (-r) parameters.

In addition to the fixed-parameters APs, randomized-parameters variants were also studied. In these cases, the algorithms in the AP were allowed to randomly select new parameter values, among the available ones in Table 5, for each new trajectory. Thus, there was a total number of 162 different APs in our experiments. Each AP was independently applied on the $CW(33, 25)$ problem for a maximum time of $T_{\max} = 300\,\text{s}$. Since the algorithms in the APs involve stochastic decisions, 25 independent experiments were conducted per AP for statistical analysis purposes.

For each individual combination of type (TS, ILS, MIX) and number of nodes (8, 16, 64), the performances of the corresponding APs were pairwisely tested using the Wilcoxon ranksum test with 0.05 significance level in order to identify statistically significant differences. The comparisons were primarily based on the successes of the APs in detecting globally optimal solutions and, secondarily, on the required running times. Then the APs were sorted according to the achieved scores.

The three best-performing APs per case are reported in Table 6 along with their parameters. For each reported AP, the percentage of success in detecting a global minimizer, the mean and standard deviation (in parenthesis) of the required time in seconds, as well as the mean number of visited local minimizers per slave node, averaged over the 25 experiments are reported. For the MIX APs, the parameters s_{tabu} and T_{nis} refer to their constituent TS algorithms.

In a second round of comparisons, all the 162 APs were statistically compared against each other, aiming at finding the overall best-performing approaches. The Wilcoxon ranksum test with 0.05 significance level was used also in this case, and the APs were sorted according to their scores. The five most promising APs are reported in the lower part of Table 6. Furthermore, Fig. 1 illustrates the mean value versus the standard deviation of the time required per AP type (TS, ILS, MIX) for fixed (-f) and random (-r) parameters. Figure 2 shows the average time required per AP type (TS, ILS, MIX) for 8, 16, and 64 nodes.

Table 6 offers interesting evidence for each AP type. First, we can notice that the best TS-based APs required higher average running times and visited less local minimizers than ILS-based and MIX APs. This is also observed in Fig. 1 where TS-based APs occupy the upper-right part of the figure. The observed time-performance profiles are reasonable, since TS spends a fraction of its computational budget for procedures related to checking and updating the tabu list, as well as for hill-climbing. Nevertheless, TS was highly effective in detecting the global minimizer. Also, we can see that the small TL size, $s_{\text{tabu}} = 5$, was dominant in the best-performing TS-based APs because larger tabu lists require additional comparisons and, consequently, reduce convergence speed. This is also in line with the dominant $T_{\text{nis}} = 100$ parameter, which promotes shorter trajectories.

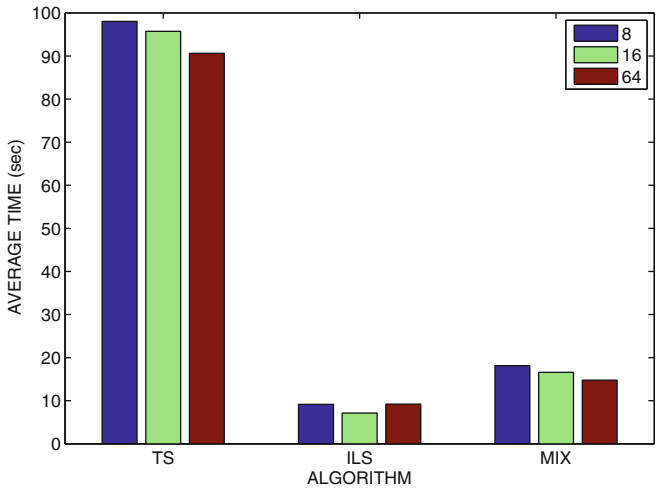

Fig. 2. Average time required per AP type (TS, ILS, MIX) for 8, 16, and 64 nodes.

Moreover, Table 6 reveals a trend of (almost linear) reduction of the number of visited minimizers with the number of nodes. This may seem counter-intuitive, because additional nodes offer higher numbers of concurrent trajectories (although at slower speed in the multi-core processor). However, it can be explained by the increase of the exploration capabilities of the algorithms, which lead to faster detection of a global minimizer. The rapid convergence is also reflected to the declining average execution times in Fig. 2. Thus, the number of concurrent trajectories seems to be highly beneficial in TS-based APs despite the possible slowdown in the AP's execution.

Another interesting observation is that TS-based APs with fixed parameters (p_{type} = "f") profited from the first-best neighborhood search (nss = "fb"). This is related to the previous comment on the significant time consumed for comparing the complete neighborhood of the current sequence with the whole tabu list. Obviously, making a move directly after a sequence of adequate quality is detected in the current neighborhood, can spare significant amount of execution time without reducing effectiveness.

Finally, the results show that APs with randomized parameters can perform equally well with the ones with fixed parameters, especially when random initialization is preferred against perturbations of the best solution. The latter can be a consequence of the inherent ability of TS to visit neighboring sequences through hill-climbing.

The ILS-based APs achieved the lowest average convergence times as depicted in Fig. 2. This is reasonable, since ILS exploits its computational budget solely in descent moves towards the nearest local minimizer. However, there is an interesting effect of the number of nodes on the average running time. As we can see in Fig. 2, doubling the number of nodes from 8 to 16 results in improved average time, but further increase to 64 nodes produces negative effects on performance since the trajectories are significantly slowed down. This verifies the existence of a trade-off between the number of concurrent trajectories and time efficiency for the ILS-based APs on multi-core processors.

The three best-performing ILS-based APs show a clear preference to fixed parameters configuration, as we notice in Table 6. Since ILS does not have an inherent mechanism for searching neighboring minimizers, there is a balanced preference between completely random new trajectories and the use of perturbations of the best solution. Also, the neighborhood-best approach seems to be more beneficial, i.e., ILS prefers to conduct steepest descent to the nearest minimizer. The number of visited local minimizers with respect to the employed nodes shows similar trends with the TS-based APs.

The performance profile of MIX APs combines performance aspects of both TS and ILS. The effect of the number of nodes on the average running time appears to follow the same trends with TS, since the running time of the AP is primarily consumed by the TS algorithms. On the other hand, the number of visited minimizers is comparable to the ILS-based APs. However, there are some peculiarities on the parameters of the three best-performing approaches, as revealed in Table 6.

Specifically, we observe that the first-best approach was dominant, obviously because it enhances the TS algorithms. However, APs with higher TL sizes and longer trajectories appear more frequently among the best ones, especially for higher number of nodes. Thus, there seems to be an interesting division of labor in MIX APs, where TS algorithms offer the AP's exploitation ability and ILS undertake the exploration task. Furthermore, perturbation-based initialization appears to be very competitive to the pure randomized one.

Overall, the synergism between TS and ILS in the MIX approaches seems to equip the APs with combined dynamics. The extra cost due to the tabu list-related procedures is counterbalanced by the first-best neighborhood search. This way, the spared computation time allows for longer trajectories and larger tabu lists.

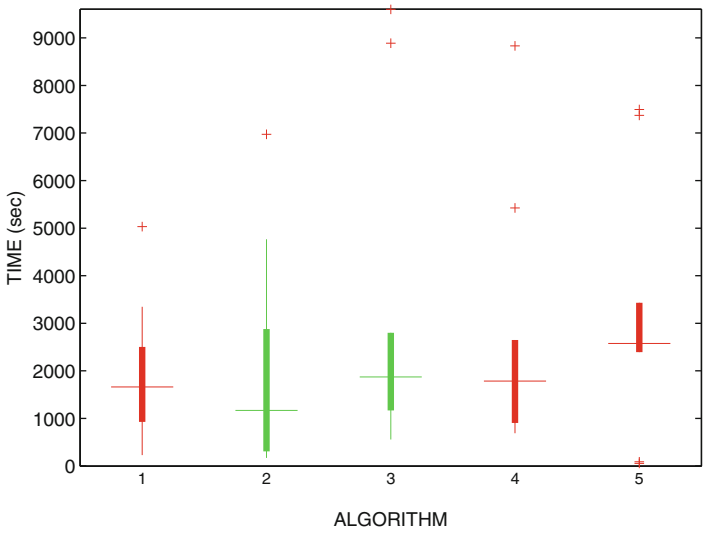

Fig. 3. Boxplots of running time of the best-performing APs on the $CW(48, 6)$ problem. Green color stands for 64-nodes APs, while red color stands for the 16-nodes APs. (Color figure online)

In a second round of comparisons, where each AP was compared against all the rest, three ILS and two MIX approaches were distinguished. As it is reported in Table 6, all these APs were based on fixed parameters and high number of nodes. Interestingly, a clear domination of the perturbation-based initialization of new trajectories is noticed.

The five distinguished approaches were further assessed on the more challenging $CW(48, 36)$ problem, which has been used as a benchmark in previous works. The maximum time per experiment for this case was $T_{\max} = 10800\,\text{s}$ (3 h). In all experiments and algorithms, a global minimizer was detected. The

boxplots of the running time of the APs over 25 experiments is illustrated in Fig. 3, where the APs appear in the same order as in Table 6.

Wilcoxon ranksum tests with 0.05 significance level showed no significant differences in terms of running time among them. However, their average times are favorably compared to those in previous studies [28], although strict comparisons would be questionable due to the completely different hardware and experimental configurations used in the present work. Nevertheless, it is a clear indication of the potential of the presented APs in solving CWM problems.

5 Conclusions

The present work enriched our insight on the performance of parallel APs on CWM problems. Enhanced TS- and ILS-based APs were used. Also, mixed APs composed by both algorithms were considered. Experimentation was focused on the widely accessible multi-core processor computational environment. Two representative test problems were used in order to investigate the performance of the APs as well as the influence of the requested number of nodes (threads), which defines also the number of the AP's algorithms, on the time efficiency and solution quality.

A rich variety of both homogeneous and heterogeneous APs were considered under various parameter settings, offering useful conclusions. ILS-based APs were significantly faster than TS-based ones, and they showed different response when the number of nodes was increased. Fixed parameters were shown to dominate randomized ones. Also, the effect of the time-consuming neighborhood-best strategy was counterbalanced by smaller tabu lists in TL-based APs. Shorter trajectories were clearly preferred in TS-based APs. Nevertheless, the best-performing mixed APs assumed also longer trajectories for the TS constituent algorithms, since running time was sparred by the ILS constituent algorithms of the AP.

Future work will consider further refinements of the AP as well as more extensive investigations of the identified trade-offs among their different properties.

Acknowledgements. Research is partially supported by the Paul and Heidi Brown Preeminent Professorship in Industrial & Systems Engineering, University of Florida.

References

1. Ang, M., Arasu, K., Ma, S., Strassler, Y.: Study of proper circulant weighing matrices with weigh 9. Discrete Math. **308**, 2802–2809 (2008)
2. Arasu, K., Dillon, J., Jungnickel, D., Pott, A.: The solution of the waterloo problem. J. Comb. Theor. Ser. A **71**, 316–331 (1995)
3. Arasu, K., Gulliver, T.: Self-dual codes over fp and weighing matrices. IEEE Trans. Inf. Theor. **47**(5), 2051–2055 (2001)
4. Arasu, K., Gutman, A.: Circulant weighing matrices. Cryptogr. Commun. **2**, 155–171 (2010)

5. Arasu, K., Leung, K., Ma, S., Nabavi, A., Ray-Chaudhuri, D.: Determination of all possible orders of weight 16 circulant weighing matrices. Finite Fields Appl. **12**, 498–538 (2006)
6. Chiarandini, M., Kotsireas, I., Koukouvinos, C., Paquete, L.: Heuristic algorithms for hadamard matrices with two circulant cores. Theoret. Comput. Sci. **407**(1–3), 274–277 (2008)
7. Cousineau, J., Kotsireas, I., Koukouvinos, C.: Genetic algorithms for orthogonal designs. Australas. J. Comb. **35**, 263–272 (2006)
8. van Dam, W.: Quantum algorithms for weighing matrices and quadratic residues. Algorithmica **34**, 413–428 (2002)
9. Eades, P.: On the existence of orthogonal designs. Ph.D. thesis, Australian National University, Canberra (1997)
10. Eades, P., Hain, R.: On circulant weighing matrices. Ars Comb. **2**, 265–284 (1976)
11. Gendreau, M., Potvin, J.Y.: Tabu search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, pp. 41–59. Springer, New York (2010)
12. Geramita, A., Sebery, J.: Orthogonical Designs: Quadratic Forms and Hadamard Matrices. Lecture Notes in Pure and Applied Mathematics. Marcel Dekker, Inc., New York (1979)
13. Glover, F.: Tabu search - part I. ORSA J. Comput. **1**, 190–206 (1989)
14. Glover, F.: Tabu search - part II. ORSA J. Comput. **2**, 4–32 (1990)
15. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell (1997)
16. Gomes, C.P., Selman, B.: Algorithm portfolio design: theory vs. practice. In: Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence, pp. 190–197 (1997)
17. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science **27**, 51–53 (1997)
18. Kotsireas, I.S., Parsopoulos, K.E., Piperagkas, G.S., Vrahatis, M.N.: Ant-based approaches for solving autocorrelation problems. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 220–227. Springer, Heidelberg (2012)
19. Kotsireas, I.: Algorithms and metaheuristics for combinatorial matrices. In: Pardalos, P., Du, D.Z., Graham, R.L. (eds.) Handbook of Combinatorial Optimization, pp. 283–309. Springer, New York (2013)
20. Kotsireas, I., Koukouvinos, C., Pardalos, P., Shylo, O.: Periodic complementary binary sequences and combinatorial optimization algorithms. J. Comb. Optim. **20**(1), 63–75 (2010)
21. Kotsireas, I., Koukouvinos, C., Pardalos, P., Simos, D.: Competent genetic algorithms for weighing matrices. J. Comb. Optim. **24**(4), 508–525 (2012)
22. Kotsireas, I.S., Parsopoulos, K.E., Piperagkas, G.S., Vrahatis, M.N.: Ant-based approaches for solving autocorrelation problems. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 220–227. Springer, Heidelberg (2012)
23. Koukouvinos, C., Seberry, J.: Weighing matrices and their applications. J. Stat. Plan. Infer. **62**(1), 91–101 (1997)
24. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, pp. 363–397. Springer, New York (2010)
25. Peng, F., Tang, K., Chen, G., Yao, X.: Population-based algorithm portfolios for numerical optimization. IEEE Trans. Evol. Comp. **14**(5), 782–800 (2010)
26. Pham, D., Karaboga, D.: Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks. Springer, London (2000)

27. Souravlias, D., Parsopoulos, K.E., Alba, E.: Parallel algorithm portfolio with market trading-based time allocation. In: Proceedings International Conference on Operations Research 2014 (OR 2014) (2014)
28. Souravlias, D., Parsopoulos, K.E., Kotsireas, I.S.: Circulant weighing matrices: a demanding challenge for parallel optimization metaheuristics. Optim. Lett. (2015)
29. Strassler, Y.: The classification of circulant weighing matrices of weight 9. Ph.D. thesis, Bar-Ilan University (1997)