Optim Lett
DOI 10.1007/s11590-015-0927-y

ORIGINAL PAPER

Circulant weighing matrices: a demanding challenge for parallel optimization metaheuristics

D. Souravlias[1] · **K. E. Parsopoulos**[1] · **I. S. Kotsireas**[2]

Received: 30 December 2014 / Accepted: 20 July 2015

Abstract Circulant weighing matrices constitute a special type of combinatorial matrices that have attracted scientific interest for many years. The existence and determination of specific classes of circulant weighing matrices remains an active research area that involves both theoretical algebraic techniques as well as high-performance computational optimization approaches. The present work aims at investigating the potential of four established parallel metaheuristics as well as a special Algorithm Portfolio approach, on solving such problems. For this purpose, the algorithms are applied on a hard circulant weighing matrix existence problem. The obtained results are promising, offering insightful conclusions.

Keywords Circulant weighing matrices · Parallel metaheuristics · Algorithm portfolios

1 Introduction

Circulant weighing matrices (CWMs) constitute a special type of combinatorial matrices and a fruitful research area for decades [5]. Applications are met in diverse scientific

✉ I. S. Kotsireas
ikotsire@wlu.ca

D. Souravlias
dsouravl@cs.uoi.gr

K. E. Parsopoulos
kostasp@cs.uoi.gr

1 Department of Computer Science and Engineering, University of Ioannina, 45110 Ioannina, Greece

2 Department of Physics and Computer Science, Wilfrid Laurier University, Waterloo, ON, Canada

Published online: 06 August 2015

fields spanning from coding theory, where they are used to construct linear codes with good properties [4], to quantum information processing for the improvement of quantum algorithms [9]. Applications can also be found in other research fields such as statistical experimentation and optical multiplexing [22].

Most research has been focused on the (non-)existence of finite or infinite classes of CWM matrices. For this purpose, a number of algebraic methodologies have been used to identify necessary existence conditions [3,5,10,11,13]. Recent works have shed light on the classification of specific CWMs [2,6,27]. Complementary to the theoretical approaches, computational optimization methods have been used in cases where theory cannot cover the entire spectra of CWMs. In such cases, the existence problem is transformed into an equivalent discrete minimization problem [7,8,18,20,21].

The present work investigates the application of established as well as novel parallel metaheuristics on CWM problems. The employed algorithms are prevailing in metaheuristics research. Specifically, we consider the trajectory-based Tabu Search (TS) [14] and Variable Neighborhood Search (VNS) [23], as well as the population-based Differential Evolution (DE) [30] and Particle Swarm Optimization (PSO) [24]. Parallelization of these algorithms is straightforward in a multi-trajectory or multi-population framework, and allows the use of communication strategies that promote cooperation in order to increase efficiency. Path-Relinking (PR) [26] is also employed to enhance performance. Moreover, we consider a recently proposed Algorithm Portfolio (AP) [29] approach, which is based on a parallel cooperative scheme with information exchange procedures based on stock trading.

In the core of the studied parallel implementations lies a typical master-slave parallelization model. In the TS, VNS, DE, and PSO approaches, each slave node executes an identical copy of the corresponding algorithm and exchanges information with the rest of the slave nodes through the master node. In this case, the parameters of the algorithm are empirically defined after preliminary experimentation. In the AP approach, a different configuration of the same algorithm on each slave node was found to be superior, exempting the user from the preprocessing phase. For all algorithms, each slave sends its best solution to the master node, where it is stored in an external archive. The slave nodes periodically request to receive elite solutions from the master. The acquired solutions are exploited either as initial conditions of new trajectories or as new individuals in the populations. All algorithms are assessed on a challenging problem of detecting CWMs of a specific class.

The rest of the paper is structured as follows: Sect. 2 presents the problem's formulation, while all algorithms are outlined in Sect. 3. Experimental settings and results are reported in Sect. 4 and Sect. 5 concludes the paper.

2 Problem formulation

A *weighing matrix* $W = W(n, k)$ of *order* n and *weight* k is an $n \times n$ square matrix with entries $w_{ij} \in \{-1, 0, 1\}$, which satisfies the condition $W W^\top = k I_n$, where $W^\top$ stands for the transpose of W, and I_n is the $n \times n$ identity matrix. A *Circulant Weighing Matrix* [5], denoted by $CW(n, k)$, is a weighing matrix where each row (except the first one) is obtained by applying a right cyclic shift to its preced-

ing row. If $G = \{g^1, g^2, \ldots, g^n\}$ is a cyclic group of order n, the columns $CW_{1,j}$ of the first row of $CW(n,k)$ can be labeled using the cyclic group G. If $P = \{g^j;\ CW_{1,j} = 1,\ j = 1, \ldots, n\}$ and $N = \{g^j;\ CW_{1,j} = -1,\ j = 1, \ldots, n\}$. Then it clearly holds that $|P| + |N| = k$. By definition, the first row of a CWM is adequate to define the complete matrix. Most works related to CWMs aim at investigating the existence (and the number) of circulant matrices $CW(n,k)$ for specific order n and weight k. Various theoretical developments that provide necessary conditions for the existence of CW matrices of various weights are reported in [2,6,11,31]. Additionally, conditions for the existence of some infinite classes of CWMs are given in [3,10,13,28].

In cases where theoretical approaches are not adequate, metaheuristics have been applied on CWM problems. These approaches require a proper objective function that is globally minimized on the appropriate ternary sequences (first rows of the CWMs). On that account, we adopt the objective function formulation proposed in [19]. Specifically, for a ternary sequence $\mathbf{x} \in \{-1, 0, +1\}^n$ of length n, the *Periodic Autocorrelation Function* (PAF) [19] is defined as:

$$PAF_{\vec{x}}(s) = \sum_{i=1}^{n} x_i\, x_{i+s}, \qquad s = 0, 1, \ldots, n-1. \tag{1}$$

The present work considers only sequences of size n and weight k, with $PAF_{\mathbf{x}}(s) = 0$, for all s. Note that each ternary sequence $\mathbf{x}$ has a symmetric sequence, and both have equal PAF values [18]. Let $\mathcal{X}_{(n,k)}$ denote the set of all ternary sequences of length n that contain exactly k^2 non-zero elements from which, $k(k+1)/2$ elements are equal to $+1$ and $k(k-1)/2$ elements are equal to -1. Then, the objective function can be represented as a permutation minimization problem with half the number of the ternary sequence elements:

$$\min_{\vec{x} \in \mathcal{X}_{(n,k)}} f(\vec{x}) = \sum_{s=0}^{\lceil \frac{n}{2} \rceil - 1} |PAF_{\vec{x}}(s)| = \sum_{s=0}^{\lceil \frac{n}{2} \rceil - 1} \left| \sum_{i=1}^{n} x_i\, x_{i+s} \right|, \tag{2}$$

where $i + s$ is taken modulo n when $i + s > n$. As example, consider a $CW(24, 9)$ matrix and let its first row be the sequence,

$$A = [0, 0, -1, 0, 0, -1, 1, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1].$$

The $CW(24, 9)$ property dictates that $PAF_{\mathrm{A}}(s) = 0$, for $s = 1, \ldots, 23$, as can be verified for A. Also, $PAF_{\mathrm{A}}(0) = 9$, since this quantity simply represents the sum of the squares of the nine non-zero elements of sequence A. Alternatively to the absolute value, squared sum of the PAF values can be used if quadratic objective functions are desirable or required from the employed algorithm. In our case, the considered algorithms do not impose such restrictions.

3 Employed algorithms

In this section, we outline the employed metaheuristics. For all the algorithms, we denote as $\mathcal{X}$ an n-dimensional search space, $f(\mathbf{x})$ the objective function under minimization, and t the iteration counter.

Tabu Search (TS) [14] is a local search algorithm, originally proposed to solve combinatorial optimization problems. The algorithm is based on two essential components, namely *Local Search* (LS) and a short-term memory, called the *Tabu List* (TL). TS is initialized on a randomly selected position in the search space. At each iteration, the current position moves to the best position of its neighborhood in terms of objective value. Recently visited positions are included in TL and constitute prohibited moves for the algorithm in order to prevent cyclic behavior. TS moves to the best neighbor of the current position regardless of improving it. This property equips TS with a *hill-climbing* capability, which is necessary for alleviating local minimizers. Also, TS restarts after a number T_{noimp} of non-improving iterations. The best position that is visited by the algorithm is the approximation of the problem's solution. TS has been successfully used in combinatorial matrices problems. Our implementation closely follows the one in [7].

Variable Neighborhood Search (VNS) [23] is another trajectory-based algorithm, originally designed to solve combinatorial optimization problems [16]. The algorithm couples an LS procedure with a perturbation mechanism, both guided by a systematic change of K predefined neighborhood structures. Given an initial position $\mathbf{x} \in \mathcal{X}$ and an initial neighborhood index $k = 1$, the algorithm randomly perturbs $\mathbf{x}$ and receives a new position $\mathbf{x}'$ in its neighborhood $N_{\mathbf{x},k}$. Then, local search is applied to find the best point $\mathbf{x}'' \in N_{\mathbf{x}',k}$. If $\mathbf{x}''$ does not improve $\mathbf{x}'$, the local search is applied anew in the $k+1$ neighborhood. Otherwise, $\mathbf{x}''$ becomes the new $\mathbf{x}'$ and the index k is reset to 1, continuing the same procedure for the new $\mathbf{x}'$. If there is no success after exceeding all the K neighborhoods, the algorithm backtracks to the original $\mathbf{x}$ and changes its neighborhood. If all neighborhoods are exceeded without success, the algorithm is applied on a new initial point $\mathbf{x}$ for $k = 1$. For the studied permutation problems, we consider the basic VNS in [16]. The k-th neighborhood of a sequence is defined as the set of all sequences produced through k distinct interchanges of mutually different components of $\mathbf{x}$. VNS is also restarted after a number of non-improving iterations.

Differential Evolution (DE) [30] is a population-based algorithm originally proposed for numerical optimization. The algorithm uses a population of N search agents, called *individuals*, to probe the search space $\mathcal{X}$ by sampling new points through *mutation* and *crossover* operators. DE employs two parameters, F and CR, which are user-defined scalars in the range $[0, 1]$. In this work, the mutation operator DE/rand/1/bin is selected among various alternatives due to its nice diversity-preserving properties. DE was originally designed for real-valued search spaces. Thus, the question arises on how it can be applied on the studied permutation problems. This is addressed by using the *smallest position value* (SPV) representation scheme [33], which maps real-valued individuals into permutations of a predefined reference vector. Specifically, this scheme considers the real values as weights that determine the priority of the corresponding components of the reference vector.

Particle Swarm Optimization (PSO) [12] is a population-based method that works similarly to DE. PSO employs a population, called a *swarm*, of N search points, called *particles*, to probe the search space. If $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ denotes the swarm in a given iteration, each particle is an n-dimensional vector $\mathbf{x}_i \in \mathcal{X}$ that iteratively moves to new positions according to adaptable position shifts $\mathbf{v}_i$, called the *velocities*. Also, during its exploration, each particle stores in memory the best position $\mathbf{p}_i$ it has ever visited in $\mathcal{X}$. The particles communicate among them with communication channels that form *neighborhood topologies*. Thus, each particle updates its velocity using both its own best position $\mathbf{p}_i$ as well as the best position $\mathbf{p}_{g_i}$ discovered by its neighbors. Detailed presentation of PSO and its variants can be found in [24]. The SPV technique [33] is also adopted in PSO to transform the real-valued particles into permutations of sequences.

3.1 Parallelization model

The employed metaheuristics can be easily parallelized with subsequent reduction in running time and improved solution quality [1]. We adopt a parallelization framework based on a typical *master-slave* model where each one of M slave nodes executes an identical copy of the algorithm. Communication among the slave nodes is achieved via the exchange of messages through the master node. The communication is inherently asynchronous and exploits a *migration scheme* [1] that involves the periodic exchange of solutions.

More specifically, the master node retains an external archive of M elite solutions, one detected by each algorithm. When an algorithm discovers a new elite solution it forwards it to the master node, replacing its previously stored solution. Periodically, each algorithm requests to acquire the best elite solution stored in the archive of the master node. The period is denoted by T_{mig}. Next, the acquired elite solution is incorporated in the algorithm either as a new starting point (TS and VNS) or as a new population member (DE and PSO).

In addition, the acquired elite solution is used for a *Path-Relinking* (PR) phase [26]. In this context, PR is initiated with a user-defined probability ρ_{PR} to locate the best permutation between the slave node's own best solution (starting point) and the acquired one (target point). This is achieved by iteratively permuting the current solution such that a new one with the lowest Hamming distance from the target point is discovered. The best solution found by this local search procedure is used as a new initial point or a new population member for the algorithm.

3.2 Algorithm portfolios

Algorithm Portfolios (APs) combine different algorithms (*heterogeneous* APs) or different configurations of the same algorithm (*homogeneous* APs) to tackle hard optimization problems [17]. Since their appearance, APs have gained increasing popularity [15,25,29,32]. The principle idea behind their development is based on the concept of *investment*. This implies investment either on the algorithms that com-

prise the portfolio [25,32] or, for population-based approaches, on the individuals that compose the population [34].

In the present work, we adopt the AP framework proposed in [29], where a different aspect of investment is explored. This AP framework consists of a number of M metaheuristic algorithms that interact with each other during the optimization process and act as investors. The AP is allocated a fixed budget of total *running time* and employs a master-slave parallelization model where each algorithm runs on a single slave. The total allowed running time is distributed equally among the algorithms and each one divides its own running time into *investment time* T_{inv} and *execution time* T_{exec}. The execution time is devoted to the specific algorithm's execution. The investment time is consumed by each algorithm on buying solutions discovered by the other algorithms, if necessary.

The master node retains in memory an archive of the M *elite solutions*. Each elite solution corresponds to the best solution discovered by each algorithm (slave node) of the AP. The master node assigns a price to each elite solution in terms of the time that will be demanded for an algorithm to buy it. Specifically, the master node sorts the solutions in descending order with respect to their objective values. If p_i is the position of the i-th solution after sorting, then its cost is defined as,

$$C_i = \frac{p_i \times BC}{M},$$

where $BC = \beta\, T_{\text{inv}}$ is a fixed base cost and $\beta \in [0, 1]$. The slave nodes interact with each other via the exchange of asynchronous messages through the master node. Specifically, if the i-th slave discovers a new elite solution denoted by $\mathbf{x}_{\text{new}}$, it sends the solution to the master where it replaces its previously stored one, $\mathbf{x}_i$ [29].

Communication also occurs when an algorithm cannot improve its own elite solution for an amount of time, T_{noimp}. In this case, the master node initiates a solution selection mechanism to propose elite solutions to the algorithm that would like to make a purchase (buyer algorithm). This mechanism is based on the *Return On Investment* (ROI) index, defined as,

$$ROI_j = \frac{f - f_j}{C_j}, \qquad j \in \{1, 2, \ldots, M\},$$

where f denotes the objective value of the algorithm's own elite solution, f_j is the objective value of the candidate buying solution, and C_j is its corresponding cost determined by the pricing procedure described above [29].

Among the offered elite solutions, the buyer algorithm decides to buy the one that maximizes the ROI index and has a better objective value than its own best solution. If the buyer algorithm decides to acquire the j-th elite solution, $\mathbf{x}_j$, then it pays to the seller algorithm (the one that discovered it) an amount of time equal to C_j (which was the price of $\mathbf{x}_j$). Specifically, the buyer algorithm reduces its own investment time by C_j, whereas the seller algorithm extends its own execution time by C_j. This way the better-performing algorithms dynamically gain more running time than the rest as

Table 1 Pseudocode of slave nodes for the AP approach

```
Initialize the AP with one algorithm per slave node.
Allocate to each algorithm execution time T_exec and investment time T_inv.
solFound ← False
While (T_exec > 0  AND  solFound = False) Do
    Apply the algorithm for some iterations.
    Send to master node a new elite solution x_new, if detected.
    If (x_new is global optimum) Then
        solFound ← True
    Else if (No improvement is achieved for T_noimp  AND  T_inv > 0) Then
        Request to buy an elite solution from the master node.
        If (Elite solution x_j is successfully bought) Then
            x_c ← crossover (x_new, x_j)
            x_m ← mutation (x_c)
            Incorporate the solution x_m in the algorithm.
        End If
    End If
End While
```

Table 2 Pseudocode of master node for the AP approach

```
solFound ← False
While (solFound = False) Do
    Wait for requests from slaves.
    If (slave i requests to send a new elite solution x_new) Then
        Replace x_i with x_new in the archive of elite solutions
    Else if (slave i requests to buy a solution) Then
        Sort elite solutions in descending order w.r.t. their objective values.
        Price elite solutions.
        Select elite solution x_j with cost C_j, j ≠ i, with the maximum ROI.
        Increase T_exec of slave j by C_j.
        Decrease T_inv of slave i by C_j.
        Send elite solution x_j to slave i.
    End If
    If (x_i is global optimum) Then
        solFound ← True
    End If
End While
```

they sell solutions more often. Yet, the total running time that is allocated to the AP remains unchanged [29].

Moreover, for the CWM problems of interest, we can further enhance the solution quality by combining the elite solution of the algorithm, $\mathbf{x}_{\text{new}}$, and the one it purchases, $\mathbf{x}_j$. The combination is attained by applying *crossover* and *mutation* between the two solutions. Specifically, crossover is implemented by retaining the solution components that are equal in the two solutions, in a new vector $\mathbf{x}_c$. Then, mutation is applied on the rest of the components through random permutations. The resulting new solution, $\mathbf{x}_m$, initiates a new trajectory (in TS and VNS) or replaces the worst particle of the population (in DE and PSO), infusing stochasticity. Pseudocode of the procedures that take place in the master and slave nodes is provided in Tables 1 and 2, respectively.

Table 3 Parameter setting for the considered algorithms

Algorithm	Parameters and Values
TS	Tabu list size $s_{TL} = 48$; Size of period $T_{\text{noimp}} = 1000$ iterations
VNS	Number of neighborhoods $K = 2$; Size of period $T_{\text{noimp}} = 1000$ iterations
DE	Population size $N = 100$; Mutation and crossover parameters $F = 0.7$, $CR = 0.3$
PSO	Swarm size $N = 100$; Parameters $\chi = 0.729$, $c_1 = c_2 = 1.49$; Neighborhood ring (radius 1)
Common	Size of period $T_{\text{mig}} = 100$ iterations; Probability $\rho_{PR} = 0.05$
AP	Investment time fraction 0.3 (i.e., 30 %); $\beta = 0.05$; Size of period $T_{\text{noimp}} = 5000$ iterations

4 Experimental results

We considered parallel implementations of TS, VNS, DE, and PSO, based on the description of Sect. 3.1, as well as an AP based on different variants of the TS, which was identified as the best-performing algorithm. Table 3 reports the parameter configuration of the employed algorithms. All experiments were conducted on the `glacier` cluster of the WestGrid consortium (`http://www.westgrid.ca`), using 8 and 16 nodes (one master node and the rest were slave nodes). The implementation was based on the OpenMPI project (`http://www.open-mpi.org/`). Also, two different running time budgets, namely 12 and 24 h, were considered for all algorithms. A number of 25 independent experiments were conducted for each algorithm and test case. Henceforth, we denote the experimental configurations with their corresponding number of CPUs followed by the running time, i.e., 8/12, 8/24, 16/12, and 16/24, respectively.

The primary objective of the experiments was to computationally verify the existence of a hard CWM class, namely the existence of $CW(48, 36)$ matrices. A secondary objective was the comparison between the algorithms. For this purpose, we recorded the number of successful experiments (out of 25) as well as the number of *unique solutions* detected by each algorithm, i.e., solutions that are not cyclic permutations of other solutions detected by the same algorithm. An experiment was considered as *successful* if a ternary sequence that defines a $CW(48, 36)$ matrix was detected, minimizing the objective function of Eq. (2). Regarding our first objective, overall the algorithms detected the 22 unique solutions given below, with "−" denoting "−1", and "+" denoting "+1":

```
0+++-0-++-+0-0-+0+++-0+-0-+++0++++-0+0-+0--+-0--
0++0+-++0-00--+-0-+-++--0++0-+++0+00+-++0++-----+
+++-0-+0-+++0+-0-0+-++-0-+++0--0-+--0+-0+0-++++0
++-000-+----+-0+--+++00+++-+000-+++++-0--+-++00-+
+00-++--000-+-+-+-0+---++00++++000-++-++-0--+++
00-+-+--00++---++0+0-+-+00++--+-00+++++++0-0+--+
-0+0+-++00+++++-00-+-+-+-0-0-+++00-++---00-+++-+
0++--+++000-+-+-00+-----0++++-++000+--++00+-++-+
+0-+++00+0-++-0+-0+-++++-0+++-00-0-+-+0+-0----++
+0+--+00++-++-00++-+++0-0-+-+00-+----00++++-++0
00++----0-+++-+-000+-+++00++++-+0-+--++-000-+++-
---+00++-++-00+++-+++0-0++-+00-+----00++-+-++0+0
```

Table 4 For each algorithm and experimental configuration, the number of successes (suc) over 25 experiments (and the corresponding percentage) are reported. Also, the number of unique solutions (uni) and the corresponding percentage with respect to the total number of solutions found by the algorithm are reported

Algorithm	Time (h)		Number of CPUs	
			8 (%)	16 (%)
TS	12	Suc	14 (56.0)	22 (88.0)
		Uni	7 (50.0)	5 (22.7)
	24	Suc	20 (80.0)	24 (96.0)
		Uni	4 (18.2)	6 (25.0)
VNS	12	Suc	3 (12.0)	4 (16.0)
		Uni	1 (33.3)	1 (25.0)
	24	Suc	5 (20.0)	6 (24.0)
		Uni	2 (40.0)	2 (33.3)
DE	12	Suc	8 (32.0)	10 (40.0)
		Uni	1 (12.5)	2 (20.0)
	24	Suc	15 (60.0)	18 (72.0)
		Uni	3 (20.0)	4 (22.2)
PSO	12	Suc	7 (28.0)	11 (44.0)
		Uni	2 (28.6)	3 (27.2)
	24	Suc	13 (52.0)	17 (68.0)
		Uni	4 (30.8)	5 (29.4)
AP	12	Suc	15 (60.0)	25 (100.0)
		Uni	7 (46.7)	10 (40.0)
	24	Suc	15 (60.0)	25 (100.0)
		Uni	5 (33.3)	9 (36.0)

```
+-++000+++-+++0--+++-+00-+---000++-+-++0+----+00+
-+0++++0++0-0+--+-0-+-+0--0++--0++0+0-+-++0++--0
00+++--++0+-+++-+00+++-+000++-++++0-----+00-+-+-0
-+-0-00+++-0-++-+--0-+0+++-0+00-+++0++++--+0-+0-
+-+--00+++--++0+0+++-+00++--+-00++-++++0-0---+00-
0+0--++00+++++-00-++-++-0-0++++00-++---00-+-+-+-
-+0+-0+---+++0++++00+0-++-0+-0--++++-0-++-00-0-+
-+0++0-0----+0-+-+0-+0+++-0++0+0+-+-0++--0--0++
++-0-0-++--+-00-+++++00+++-+0+0-+-+++-00----++-00+
-0++---0-0++0+-++0--0---++0-+-++0+0++0-+++0+-0+-+
```

Regarding the second objective, Table 4 reports the successes (both in number and percentage over 25 experiments) of each algorithm. Additionally, the number of unique solutions and the corresponding percentage with respect to the total number of detected solutions are reported. A first reading of the results clearly shows that TS outperforms the rest of the distinct algorithms by achieving significantly a higher number of successes. This is verified also for the number of unique solutions. Obviously, the exhaustive local neighborhood search, along with the hill-climbing capabilities and the cooperation of the parallel scheme, equipped TS with satisfactory trade-off between exploration and exploitation.

However, all algorithms were outperformed by the AP in most test cases. In fact, the AP was the only approach that achieved 100 % successes in half experimental configurations. Even in the 8/24 case, where TS outperformed AP in successes, its

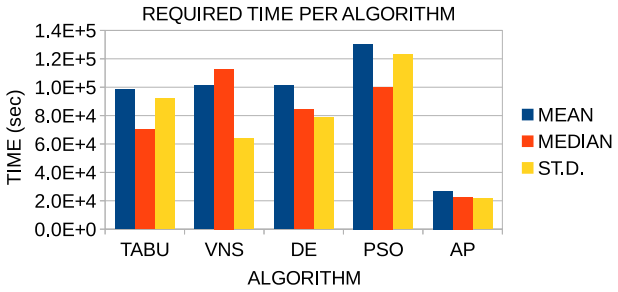

Fig. 1 Statistics for the required running time (in seconds) per algorithm for the successful experiments over all problem configurations

unique solutions were 4 out of 20 successful experiments against 5 out of 15 successful experiments of the AP. This evidence verifies the gain from the special solution trading scheme implemented in the AP against the simple cooperation of the parallelized algorithms. The rest of the algorithms, especially DE and PSO, could easily bypass a solution. Yet, the performance between DE and PSO was very similar, with their success percentages deviating by 6 % at most.

Table 4 also reveals a consistent improvement of the four algorithms when the number of CPUs or the running time is doubled. In fact, increasing the running time appears to be more effective than increasing the number of CPUs in most cases. However, this is not verified for the AP, where the number of CPUs seems to be of primary importance. At first sight, this observation appears to contradict the previous one. However, it can be easily explained by the evidence illustrated in Fig. 1, which reports the required running time (in seconds) per algorithm only for the successful experiments over all problem configurations. As we can see, the AP approach requires only a small fraction of the running time required by the rest of the algorithms to detect a solution. Thus, providing additional time does not have a crucial impact. On the other hand, adding slave nodes increases the search capacity of the AP with a consequent surge in successes.

We further investigated the reported running times by conducting Wilcoxon ranksum tests for each pair of algorithms (only for the successful experiments) at significance level 95 %. The tests revealed that only the AP had statistically significant differences in running time with the rest of the algorithms. Overall, the AP approach was shown to be the most efficient and effective among the considered ones. Given that AP differs from the simple parallel TS only in the sophisticated solution trading scheme, we can infer that the proposed AP framework can be highly beneficial.

5 Conclusions

Circulant weighing matrices (CWMs) have been a fertile research area for several decades. We proposed a framework for the application of parallel metaheuristics on a challenging problem that emerges in this research area. Our approach included four essential metaheuristics as well as an Algorithm Portfolio scheme that employs a sophisticated solution-trading mechanism. The experimental results were highly

promising, providing a number of solutions on a CWM existence problem. This offers motivation for further research on the application of efficient metaheuristics on combinatorial matrices problems.

Acknowledgments The authors would like to thank the Shared Hierarchical Academic Research Computing Network (SHARCNET) as well as the WestGrid HPC consortium for offering the necessary computational resources.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley, London (2005)
2. Ang, M., Arasu, K., Ma, S., Strassler, Y.: Study of proper circulant weighing matrices with weigh 9. Discrete Math. **308**, 2802–2809 (2008)
3. Arasu, K., Dillon, J., Jungnickel, D., Pott, A.: The solution of the waterloo problem. J. Comb. Theory Ser. A **71**, 316–331 (1995)
4. Arasu, K., Gulliver, T.: Self-dual codes over fp and weighing matrices. IEEE Trans. Inf. Theory **47**(5), 2051–2055 (2001)
5. Arasu, K., Gutman, A.: Circulant weighing matrices. Cryptogr. Commun. **2**, 155–171 (2010)
6. 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)
7. Chiarandini, M., Kotsireas, I., Koukouvinos, C., Paquete, L.: Heuristic algorithms for hadamard matrices with two circulant cores. Theor. Comput. Sci. **407**(1–3), 274–277 (2008)
8. Cousineau, J., Kotsireas, I., Koukouvinos, C.: Genetic algorithms for orthogonal designs. Australas. J. Comb. **35**, 263–272 (2006)
9. van Dam, W.: Quantum algorithms for weighing matrices and quadratic residues. Algorithmica **34**, 413–428 (2002)
10. Eades, P.: On the existence of orthogonal designs. Ph.D. thesis, Australian National University, Canberra (1997)
11. Eades, P., Hain, R.: On circulant weighing matrices. Ars Comb. **2**, 265–284 (1976)
12. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings Sixth Symposium on Micro Machine and Human Science, pp. 39–43. Piscataway, NJ (1995)
13. Geramita, A., Sebery, J.: Orthogonical designs: quadratic forms and hadamard matrices. Lecture Notes in Pure and Applied Mathematics (1979)
14. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986)
15. Gomes, C.P., Selman, B.: Algorithm portfolio design: theory vs. practice. In: Proceedings Thirteenth conference on Uncertainty in artificial intelligence, pp. 190–197 (1997)
16. Hansen, P., Mladenović, N., Brimberg, J., Moreno Pérez, J.A.: Variable neighborhood search. In: M. Gendreau, J.Y. Potvin (eds.) Handbook of Metaheuristics, vol. 146, chap. 3. Springer, Berlin (2010)
17. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science **27**, 51–53 (1997)
18. Kotsireas, I.: Algorithms and metaheuristics for combinatorial matrices. In: P. Pardalos, D.Z. Du, R.L. Graham (eds.) Handbook of Combinatorial Optimization, pp. 283–309. Springer, New York (2013)
19. Kotsireas, I., Koukouvinos, C., Pardalos, P., Shylo, O.: Periodic complementary binary sequences and combinatorial optimization algorithms. J. Combin. Optim. **20**(1), 63–75 (2010)
20. Kotsireas, I., Koukouvinos, C., Pardalos, P., Simos, D.: Competent genetic algorithms for weighing matrices. J. Combin. Optim. **24**(4), 508–525 (2012)
21. Kotsireas, I., Parsopoulos, K., Piperagkas, G., Vrahatis, M.: Ant-based approaches for solving autocorrelation problems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7461 LNCS, pp. 220–227 (2012)
22. Koukouvinos, C., Seberry, J.: Weighing matrices and their applications. J. Stat. Plan. Inference **62**(1), 91–101 (1997)
23. Mladenovic, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)

24. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization and Intelligence: Advances and Applications. Information Science Publishing (IGI Global), Hershey, USA (2010)
25. Peng, F., Tang, K., Chen, G., Yao, X.: Population-based algorithm portfolios for numerical optimization. IEEE Trans. Evol. Comput. **14**(5), 782–800 (2010)
26. Ribeiro, C., Resende, M.: Path-relinking intensification methods for stochastic local search algorithms. J. Heuristics **18**(2), 193–214 (2012)
27. Schmidt, B., Smith, K.W.: Circulant weighing matrices whose order and weight are products of powers of 2 and 3. J. Comb. Theory Ser. A **120**(1), 275–287 (2013)
28. Seberry, J., Whiteman, A.: Some results on weighing matrices. Bull. Aust. Math. Soc. **12**, 433–447 (1975)
29. Souravlias, D., Parsopoulos, K.E., Alba, E.: Parallel algorithm portfolio with market trading-based time allocation. In: Proceedings International Conference on Operations Research 2014 (OR2014) (2014)
30. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
31. Strassler, Y.: The classification of circulant weighing matrices of weight 9. Ph.D. thesis, Bar-Ilan University (1997)
32. Tang, K., Peng, F., Chen, G., Yao, X.: Population-based algorithm portfolios with automated constituent algorithms selection. Inf. Sci. **279**, 94–104 (2014)
33. Tasgetiren, F., Chen, A., Gencyilmaz, G., Gattoufi, S.: Smallest position value approach. Stud. Comput. Intel. **175**, 121–138 (2009)
34. Yevseyeva, I., Guerreiro, A.P., Emmerich, M.T.M., Fonseca, C.M.: A portfolio optimization approach to selection in multiobjective evolutionary algorithms. Proc. PPSN **2014**, 672–681 (2014)