INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 24 (2017) 1061–1077
DOI: 10.1111/itor.12429

A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems

Julliany S. Brandão[a,b], Thiago F. Noronha[c], Mauricio G. C. Resende[d] and Celso C. Ribeiro[a]

[a]*Institute of Computing, Universidade Federal Fluminense Niterói, RJ 24210-346, Brazil*
[b]*Centro Federal de Educação Tecnológica Celso Suckow da Fonseca Rio de Janeiro, RJ 20271-110, Brazil*
[c]*Department of Computer Science, Universidade Federal de Minas Gerais Belo Horizonte, MG 24105, Brazil*
[d]*Amazon.com, Mathematical Optimization and Planning Seattle, WA 98109, USA*
E-mail: dscjbrandao@gmail.com [Brandão]; tfn@dcc.ufmg.br [Noronha]; mresende@gmail.com [Resende]; celso@ic.uff.br [Ribeiro]

Received 30 November 2015; received in revised form 30 April 2016; accepted 31 May 2016

Abstract

A divisible load is an amount W of computational work that can be arbitrarily divided into independent chunks of load. In many divisible load applications, the load can be parallelized in a master–worker fashion, where the master distributes the load among a set P of worker processors to be processed in parallel. The master can only send load to one worker at a time, and the transmission can be done in a single round or in multiple rounds. The multi-round divisible load scheduling problem consists in (a) selecting the subset $A \subseteq P$ of workers that will process the load, (b) defining the order in which load will be transmitted to each of them, (c) defining the number m of transmission rounds that will be used, and (d) deciding the amount of load that will be transmitted to each worker $i \in A$ at each round $k \in \{1, \ldots, m\}$, so as to minimize the makespan. We propose a heuristic approach that determines the transmission order, the set of the active processors and the number of rounds by a biased random-key genetic algorithm. The amount of load transmitted to each worker is computed in polynomial time by closed-form formulas. Computational results showed that the proposed genetic algorithm outperformed a closed-form state-of-the-art heuristic, obtaining makespans that are 11.68% smaller on average for a set of benchmark problems.

Keywords: divisible loads; divisible load scheduling; multi-round; biased random-key genetic algorithms; metaheuristics

1. Introduction

A *divisible load* is an amount W of computational work that can be arbitrarily divided and distributed among different processors to be computed in parallel. The processors are arranged in a star topology and the load is stored in a central *master* processor. The master splits the load into chunks

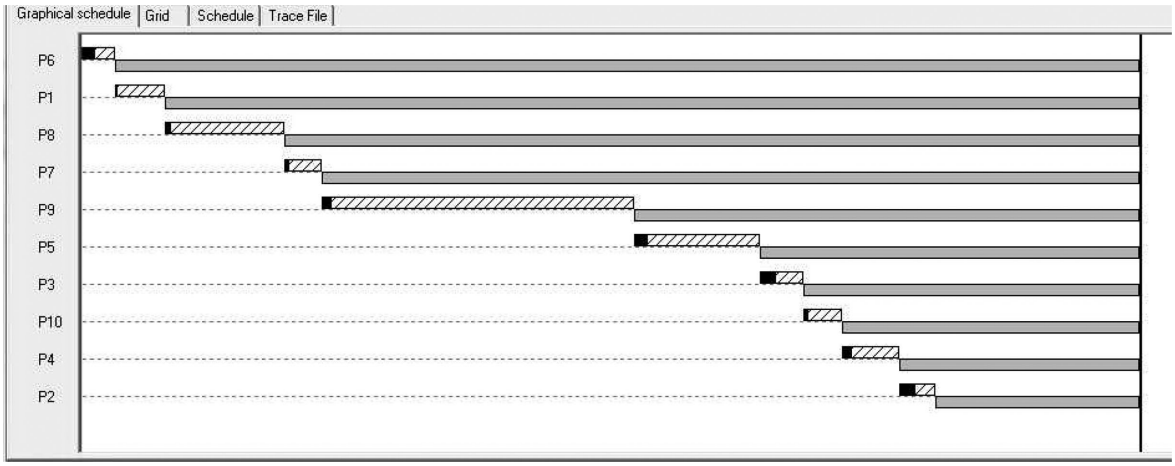


Fig. 1. Example of an optimal single-round scheduling.

of arbitrary sizes and transmits each of them to the *worker* processors. We assume that the master can only send load to one worker at a time, and that it does not compute the load itself. In addition, a worker can only start the computation of a piece of load after its transmission was completed, i.e., after this worker has entirely received its load.

The workers are heterogeneous in terms of processing power, communication speed, and setup times. Due to communication and setup times, it might be faster not to use all the available worker processors to compute the load. Those used in this computation are referred to as *active workers*. The order used by the master to send the load to each worker is called the *activation order*. Once the latter is set, the transmission can be performed either in a single round or in multiple rounds.

In *single-round systems* (Abib and Ribeiro, 2009; Beaumont et al., 2008, 2005, 2003b; Berlińska et al., 2009; Bharadwaj et al., 1996; Blazewicz and Drozdowski, 1997; Drozdowski, 1997; Drozdowski and Wolniewicz, 2003, 2006; Li et al., 2000; Shokripour et al., 2011; Wang et al., 2013, 2014; Wolniewicz, 2003), each active worker receives its load in a single chunk, and the setup costs are incurred only once for each worker. However, in this case, the ith worker in the transmission order remains idle until the master sends the load to all workers that appear earlier in the activation order. An example of an optimal single-round schedule is shown in Fig. 1. All ten workers are active and the activation order is $< 6, 1, 8, 7, 9, 5, 3, 10, 4, 2 >$. Black boxes represent the setup time to start the communication with the master, dashed boxes correspond to the amount of time needed to receive the respective load chunk, and gray boxes show the time spent by the worker to process this load. This example does not show setup times to start processing the load. It can be seen that the master only starts the communication with a worker after it finishes the communication with the previous worker in the transmission order. One can also see that each worker receives a single load chunk and that it only starts processing this chunk after its transmission is finished. Since the load can be arbitrarily split, all processors stop and finish their work at the same time in the optimal solution minimizing the makespan (Beaumont et al., 2003b).

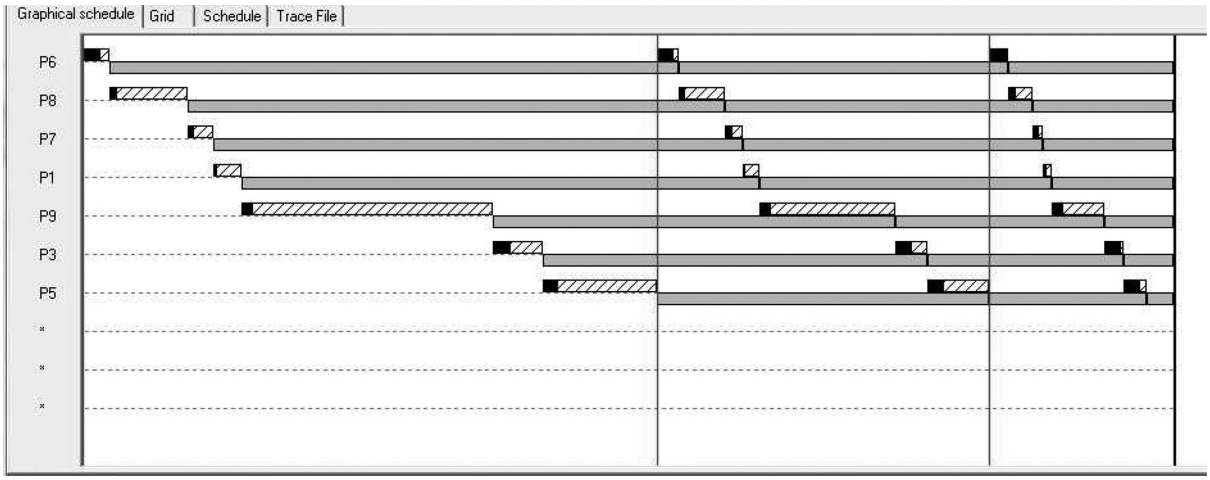


Fig. 2. Example of an optimal three-round scheduling.

In *multi-round systems* (Beaumont et al., 2005, 2003a; Berlinska and Drozdowski, 2010; Bharadwaj et al., 1995; Drozdowski and Lawenda, 2006, 2007; Hsu et al., 2008; Lin et al., 2008; Maciej and Marcin, 2008; Shokripour et al., 2010, 2012; Wolniewicz, 2003; Yang and Casanova, 2003a, 2003b; Yang et al., 2007, 2005), the transmission is divided into rounds and every active worker receives a load chunk at each round. The activation order is the same for all rounds (Yang and Casanova, 2003a, 2003b; Yang et al., 2005). Multi-round systems may reduce the makespan by decreasing the idle times, at the additional cost of repeated setup times at each round. Figure 2 illustrates an optimal three-round schedule. The activation order is $< 6, 8, 7, 1, 9, 3, 5, 2, 4, 10 >$, but only the seven first workers are active. It can be seen that the master only starts the communication with a worker after it finishes the communication with the previous worker in the activation order. Idle times in the first round are smaller, because the load chunks are spread among the three rounds. One can also see that each worker only starts processing a load chunk after its transmission is finished. Again, as the load can be arbitrarily split, all processors stop at the same time in the optimal solution minimizing the makespan (Beaumont et al., 2003b).

We focus on the scheduling of multi-round systems (Beaumont et al., 2003b; Shokripour et al., 2012). Let $W \in \mathbb{R}_+$ be the amount of load to be processed, 0 (zero) be the index of the master processor, and $P = \{1, \ldots, |P|\}$ be the set of indices of the worker processors. Each processor $i \in P$ has (a) a setup time $g_i \in \mathbb{R}_+$ to start the communication with the master, (b) a communication time $G_i \in \mathbb{R}_+$ needed to receive each unit of the load from the master, (c) a setup time $s_i \in \mathbb{R}_+$ to start the load computation, and (d) a processing time $w_i \in \mathbb{R}_+$ needed to process each unit of the load. Therefore, it takes $g_i + G_i \cdot \theta_i^j$ units of time for the master to transmit a load chunk of size $\theta_i^k \geq 0$ to processor $i \in P$ in round k. Furthermore, it takes an additional $s_i + w_i \cdot \theta_i^k$ units of time for this worker to process the chunk of load assigned to it.

The multi-round divisible load scheduling (MR-DLS) problem consists in (a) defining an activation order π, (b) deciding the number n of active workers, (c) choosing the number m of rounds, and (d) determining the amount of load θ_i^k that will be transmitted to each processor $i \in P$ at each

round $k \in \{1, \ldots, m\}$, with $\sum_{i \in P} \sum_{k \in \{1,\ldots,m\}} \theta_i^k = W$, so as to minimize the *makespan*. The latter is defined as the total elapsed time since the master began to send data to the first worker, until the last worker stops its computation. We assume that processor $\pi_i \in P$ is the ith worker to be activated in the activation order π, for $i = 1, \ldots, |P|$. The first n workers in π are active: $A = \{\pi_1, \ldots, \pi_n\}$ and $\theta_{\pi_i}^k = 0$, for all $i \in \{n+1, \ldots, |P|\}$ and $k \in \{1, \ldots, m\}$.

In this work, we propose a biased random-key genetic algorithm (BRKGA) (Ericsson et al., 2002; Gonçalves and Resende, 2011, 2004) for solving MR-DLS. BRKGAs have been successfully used for solving many permutation based combinatorial optimization problems, see, e.g. (Duarte et al., 2014; Gonçalves and Resende, 2011, 2013, 2014; Gonçalves et al., 2016; Noronha et al., 2011), including the single-round divisible load scheduling (DLS) problem (Brandão et al., 2015). Computational results showed that the proposed genetic algorithm outperformed a closed-form state-of-the-art heuristic (Shokripour et al., 2012), obtaining makespans that are 11.68% smaller on average for a set of benchmark problems.

The remainder of this paper is organized as follows. Related work is presented and discussed in the next section. The proposed heuristic is described in Section 3. Computational experiments are reported in Section 4. Concluding remarks are drawn in the last section.

2. Related work

There are many variants of the DLS problem. The main differences between them are discussed below, followed by a literature review of the particular variant considered in this paper.

In the case of *homogeneous* worker processors, the values of g_i, G_i, s_i, and w_i are the same for all processors $i \in P$ (Beaumont et al., 2005; Bharadwaj et al., 1995, 1996; Kim, 2003; Lin et al., 2008; Maciej and Marcin, 2008; Yang and Casanova, 2003b; Yang et al., 2005). When the workers are *heterogeneous*, these values may be different for each worker (Abib and Ribeiro, 2009; Beaumont et al., 2008, 2003a, 2003b; Berlinska and Drozdowski, 2010; Drozdowski, 1997; Drozdowski and Lawenda, 2006, 2007; Drozdowski and Wolniewicz, 2003, 2006; Hsu et al., 2008; Li et al., 2000; Shokripour et al., 2010, 2011, 2012; Wang et al., 2013, 2014; Wolniewicz, 2003; Yang and Casanova, 2003a, 2003b; Yang et al., 2007, 2005). Heterogeneous systems are more common than homogeneous systems.

When the system is *dedicated*, it is assumed that all resources (such as processors, memory, and network) are used to process one single computational load (Abib and Ribeiro, 2009; Beaumont et al., 2003b; Berlinska and Drozdowski, 2010; Berlińska et al., 2009; Bharadwaj et al., 1996; Blazewicz and Drozdowski, 1997; Drozdowski, 1997; Drozdowski and Wolniewicz, 2003, 2006; Shokripour et al., 2010; Wang et al., 2013, 2014; Wolniewicz, 2003; Yang and Casanova, 2003a, 2003b). *Nondedicated* systems may be used to simultaneously process different computational loads (Berlinska and Drozdowski, 2010; Berlińska et al., 2009; Shokripour et al., 2011). Most authors assume that the systems are dedicated.

Variants of the DLS problem that are *buffer constrained* (Berlinska and Drozdowski, 2010; Berlińska et al., 2009; Drozdowski and Lawenda, 2006; Drozdowski and Wolniewicz, 2003, 2006; Li et al., 2000; Maciej and Marcin, 2008; Wolniewicz, 2003; Yang and Casanova, 2003a, 2003b) assume a limitation to the maximum chunk size that can be received by each worker processor.

When such a limitation does not exist, the problem is said to be *unconstrained* (Abib and Ribeiro, 2009; Beaumont et al., 2005, 2003b; Bharadwaj et al., 1995, 1996; Blazewicz and Drozdowski, 1997; Drozdowski, 1997; Lin et al., 2008; Shokripour et al., 2010, 2011, 2012; Wang et al., 2013, 2014; Yang et al., 2007). Many authors assume unconstrained systems, because there is usually plenty of secondary memory to temporarily store the load at the workers.

The variant of the MR-DLS problem considered in this work is unconstrained and assumes dedicated and heterogeneous worker processors. It was proved to be NP-hard in (Yang et al., 2007). The main heuristics in the literature are discussed below.

Yang and Casanova (2003b) and Yang et al. (2005) were the first to tackle multi-round scheduling problems with heterogeneous workers. They proposed the UMR heuristic. The workers are sorted in nondecreasing order of their G_i values. This order is used as the activation order. Let the *round length* of any worker i at any round k be the sum of the setup, communication, and computation times of i at round k. UMR imposes as an additional constraint that at any given round all active workers should have the same round length. With this additional constraint, the authors are able to derive closed-form formulas to compute the number of active workers n and the number of rounds m, as well as analytical expressions to compute the value of θ_i^k for all $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, m\}$. Computational experiments showed that UMR outperformed the algorithms in Bharadwaj et al. (1996) and Rosenberg (2001) designed for homogeneous and single-round systems, respectively.

Hsu et al. (2008) proposed the extended smallest communication ratio (ESCR) heuristic. The activation order is computed as in Yang and Casanova (2003b) and Yang et al. (2005). However, differently from Yang and Casanova (2003b) and Yang et al. (2005), all workers in P are activated, i.e., $n = |P|$. ESCR calculates the least common multiplier (LCM) of $\{G_i + w_i : i \in P\}$ and uses this value to compute the round length. The number of rounds m is computed using closed-form formulas based on the LCM. Analytical expressions are used to compute the value of θ_i^k for all $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, m\}$. Computational experiments showed that ESCR outperformed the algorithm in Beaumont et al. (2003a), although the latter was designed for single-round systems.

Beaumont et al. (2003b) proposed and evaluated four different heuristics for the MR-DLS problem. The one that obtained the best results was the linear programming with adaptive period (LPAP) heuristic. Again, the activation order is computed as in Yang and Casanova (2003b) and Yang et al. (2005). However, the round length is not the same at each round. Instead, LPAP computes an upper bound T to the makespan. Based on this estimation, the round length T_k of each round k is set to $T_k = \sqrt{T}$. Once this round length is set, LPAP uses linear programming to find the largest chunk size θ_i^k that can be sent to each processor $i \in P$ at this round k so that the round length is not larger than T_k. Since the upper bound T becomes tighter at each iteration of this algorithm, the round length of any round $k + 1$ is always smaller than the round length of round k. Computational experiments showed that LPAP outperforms the algorithm of Bharadwaj et al. (1996).

Shokripour et al. (2012) proposed two heuristics: computational based method and communication based method (CBM). The best results were obtained by CBM. Let *round size* be the sum of the load chunks all the active workers receive at any given round. CBM imposes as an additional constraint that all rounds have the same round size $Q = W/m$. The activation order π in which load is sent to the workers is obtained by sorting the workers by nondecreasing order of their G_i values. Processor π_i is the ith worker in the activation order π, for $i = 1, \ldots, |P|$. The number of active

workers (n) in this order and the number of rounds (m) are computed as follows. Let $\bar{n} \in \{1, \ldots, |P|\}$ be any candidate value for n. First, CBM assumes that the number of rounds is $\bar{m} = q(\bar{n}, \pi)$, where

$$q(\bar{n}, \pi) = \sqrt{\frac{W\left[\left(1 + \sum_{i=2}^{\bar{n}} \Delta_{\pi_i}\right)(G_{\pi_1} + w_{\pi_1}) - \left(1 + \sum_{i=2}^{\bar{n}} E_{\pi_i}\right)\sum_{j=1}^{\bar{n}} \Delta_{\pi_j} G_{\pi_j}\right]}{\left(1 + \sum_{i=2}^{\bar{n}} E_{\pi_i}\right)\left[\left(1 + \sum_{i=2}^{\bar{n}} \Delta_{\pi_i}\right)\left(\sum_{j=2}^{\bar{n}} \phi_{\pi_j} G_{\pi_j} + \sum_{j=1}^{\bar{n}} g_{\pi_j}\right) - \sum_{i=2}^{\bar{n}} \phi_{\pi_i} \sum_{j=1}^{\bar{n}} \Delta_{\pi_j} G_{\pi_j}\right]}}.$$

Then, the amount of load that can be processed using $\bar{n}$ active workers and $\bar{m} = q(\bar{n}, \pi)$ rounds, following the activation order π, is estimated by

$$f(\bar{n}, \pi) = q(\bar{n}, \pi) \cdot \left(\frac{\left(1 + \sum_{i=2}^{\bar{n}} \Delta_{\pi_i}\right)\left(\sum_{j=2}^{\bar{n}} (\phi_{\pi_j} G_{\pi_j} + g_{\pi_j}) - s_{\pi_1}\right)}{w_{\pi_1} - \sum_{j=2}^{\bar{n}} \Delta_{\pi_j} G_{\pi_j}} + \sum_{i=2}^{\bar{n}} \phi_{\pi_i}\right),$$

where $\Delta_{\pi_i} = (G_{\pi_1} + w_{\pi_1})/(G_{\pi_i} + w_{\pi_i})$ and $\phi_{\pi_i} = (G_{\pi_1} + w_{\pi_1})/(G_{\pi_i} + w_{\pi_i})$. Finally, CBM sets the number of active workers to the smallest value of $\bar{n}$ such that $f(\bar{n}, \pi) \geq W$, i.e.,

$$n = \text{argmin}_{\bar{n}=1,\ldots,|P|}\{f(\bar{n}, \pi) \,:\, f(\bar{n}, \pi) \geq W\}, \tag{1}$$

and the number of rounds $m = q(n, \pi)$.

Once the values of n and m have been computed, the fraction α_{π_i} of the load size Q processed by each worker $\pi_i \in \{1, \ldots, n\}$ at every round $k = \{1, \ldots, m-1\}$ is given by

$$\alpha_{\pi_i} = \begin{cases} \left(1 - \frac{1}{Q}\sum_{i=2}^{n} \phi_{\pi_i}\right) \Big/ \left(1 + \sum_{i=2}^{n} \Delta_{\pi_i}\right), & \text{if} \quad i = 1, \\ \alpha_{\pi_1} \Delta_{\pi_i} + \frac{1}{Q}\phi_{\pi_i}, & \text{if} \quad i = 2, \ldots, n. \end{cases}$$

Consequently, the fraction β_{π_i} of the load size Q processed by each worker $\pi_i \in \{1, \ldots, n\}$ in the last round is given by

$$\beta_{\pi_i} = \begin{cases} \left(1 - \frac{1}{Q}\sum_{i=2}^{n} \Gamma_{\pi_i}\right) \Big/ \left(1 + \sum_{i=2}^{n} E_{\pi_i}\right), & \text{if} \quad i = 1, \\ \beta_{\pi_1} E_{\pi_i} + \frac{1}{Q}\Gamma_{\pi_i}, & \text{if} \quad i = 2, \ldots, n, \end{cases}$$

where $\delta_{\pi_i} = (s_{\pi_i} - s_{\pi_{i+1}} - g_{\pi_{i+1}})/(w_{\pi_{i+1}} + G_{\pi_{i+1}})$, $\epsilon_{\pi_i} = w_{\pi_i}/(w_{\pi_{i+1}} + G_{\pi_{i+1}})$, $E_{\pi_i} = \prod_{j=2}^{i} \epsilon_{\pi_j}$, and $\Gamma_{\pi_i} = \sum_{j=2}^{i} (\delta_{\pi_j} \prod_{l=j+1}^{i} \epsilon_{\pi_l})$.

The pseudo-code of CBM is presented in Fig. 3. It receives as input the value W, the set P, and the values of g_i, G_i, s_i, and w_i for every $i \in P$. The transmission order π is computed in line 1. The number n of active workers is computed in lines 2 to 5, while the number m of rounds is calculated in line 6. The round size Q is computed in line 7. The loop from lines 8 to 11 computes the value of θ_i^k for every active worker $\pi_i \in \{1, \ldots, n\}$. The size of the load chunk received by each worker π_i at each of the $m-1$ first rounds is set in line 9, while that for the last round is set in line 10. The values of $\theta_{\pi_i}^k$ for nonactive workers are set in the loop of lines 12 to 14. The solution, represented by π, n, m, and θ, is returned in line 15. Computational experiments showed that heuristic CBM

procedure CBM(W, P, g_i, G_i, s_i, w_i)
1 Let π be a permutation of P with $G_{\pi_i} \leq G_{\pi_{i+1}}$, for $i = 1, \ldots, |P| - 1$;
2 Set $n \leftarrow 1$;
3 **while** $f(n, \pi) < W$ **and** $n < |P|$ **do**;
4 $n \leftarrow n + 1$;
5 **end-while**;
6 Set $m \leftarrow q(n, \pi)$;
7 Set $Q \leftarrow W/m$;
8 **for** $i = 1, \ldots, n$ **do**;
9 Set $\theta^k_{\pi_i} \leftarrow Q \cdot \alpha_{\pi_i}$, for all $k \in \{1, \ldots, m-1\}$;
10 Set $\theta^m_{\pi_i} \leftarrow Q \cdot \beta_{\pi_i}$;
11 **end-for**
12 **for** $i = n + 1, \ldots, |P|$ **do**;
13 Set $\theta^k_{\pi_i} \leftarrow 0$, for all $k \in \{1, \ldots, m\}$;
14 **end-for**;
15 **return** $< \pi, n, m, \theta >$;
end CBM.

Fig. 3. Pseudo-code of the CBM heuristic for MR-DLS.

outperformed heuristics LPAP (Beaumont et al., 2003b) and ESCR (Hsu et al., 2008). To the best of our knowledge, CBM is the best heuristic for problem MR-DLS at the time of writing.

3. Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGAs), were first introduced by Bean (1994) for combinatorial optimization problems whose solutions may be represented by permutation vectors. Solutions are represented as vectors of randomly generated real numbers called keys. A deterministic algorithm, called a decoder, takes as input a solution vector and associates with it a feasible solution of the combinatorial optimization problem, for which an objective value or fitness can be computed. Two parents are selected at random from the entire population to implement the crossover operation in the implementation of an RKGA. Parents are allowed to be selected for mating more than once in the same generation.

A BRKGA differs from an RKGA in the way parents are selected for crossover (see Gonçalves and Resende, 2011, for a review). In a BRKGA, each element is generated combining one element selected at random from the elite solutions in the current population, while the other is a nonelite solution. The selection is said to be biased because one parent is always an elite solution, and has a higher probability of passing its genes to the new generation.

The BRKGA for MR-DLS, called GA-KEY, evolves a population of chromosomes that consists of vectors of real numbers. Each solution is represented by a vector of $|P| + 2$ components, in which each key is a real number in the range $[0, 1)$. Each of the $|P|$ first keys is associated with a worker processor in P. The $(|P| + 1)$th key is associated with the number n of active workers, while the $(|P| + 2)$th key is associated with the number m of rounds. Each solution represented by

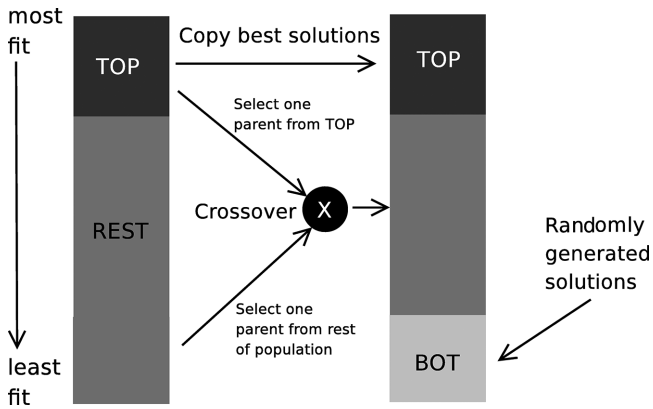


Fig. 4. Population evolution between consecutive generations of a BRKGA.

a chromosome is decoded by an algorithm that receives the vector of keys and builds a feasible solution for MR-DLS, i.e., the decoder returns $< \pi, n, m, \theta >$.

Given any chromosome c from the population, the decoding algorithm builds a solution as follows. The activation order π is determined by sorting the worker processors in a nondecreasing order of their keys. Let $x \in [0, 1)$ be the value of the $(|P| + 1)$th key in chromosome c. The number of active workers is an integer number uniformly distributed in the interval $[1, |P|]$ obtained from the value of x by $n = \lfloor |P| \cdot x + 1 \rfloor$. Since, in principle, there is no upper bound to the number of rounds, we use a different approach to compute the number m of rounds. Let $y \in [0, 1)$ be the value of the $(|P| + 2)$th key in chromosome c. The number of rounds is set as $m = \lfloor \frac{1}{1-y} \rfloor$. An advantage of this approach is that smaller values of m are more likely to be chosen, which is opportune as the larger is the number of rounds, the larger is the amount of time spent in communication and computation setup times. The values of θ_i^k, for all $i \in P$ and $k \in \{1, \ldots, m\}$, are calculated as in CBM (Shokripour et al., 2012), i.e., following lines 7 (we recall that the value of m is now given by the last random key) to 14 of the pseudo-code in Fig. 3. The makespan of the resulting solution is used as the fitness of the chromosome.

We use the parametric uniform crossover scheme proposed in Spears and de Jong (1991) to combine two parent solutions and produce an offspring. In this scheme, the offspring inherits each of its keys from the best fit of the two parents with probability 0.6 and from the least fit parent with probability 0.4. This genetic algorithm does not use the standard mutation operator, where parts of the chromosomes are changed with small probability. Instead, the concept of mutants is used: mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population. Mutants play the same role of the mutation operator in traditional genetic algorithms, diversifying the search and helping the procedure to escape from locally optimal solutions.

The $(|P| + 2)$ keys in the chromosome are randomly generated in the initial population. At each generation, the population is partitioned into two sets: *TOP* and *REST*. Consequently, the size of the population is $|TOP| + |REST|$. Subset *TOP* contains the best solutions in the population. Subset *REST* is formed by two disjoint subsets: *MID* and *BOT*, with subset *BOT* being formed by the worst elements in the current population. As illustrated in Fig. 4, the chromosomes in *TOP* are

simply copied to the population of the next generation. The elements in *BOT* are replaced by newly created mutants. The remaining elements of the new population are obtained by crossover with one parent randomly chosen from *TOP* and the other from *REST*. This distinguishes a BRKGA from the RKGA of Bean (1994), where both parents are selected at random from the entire population. Since a parent solution can be chosen for crossover more than once in a given generation, elite solutions have a higher probability of passing their random keys to the next generation. In this way, $|MID| = |REST - BOT|$ offspring solutions are created. In our implementation, the population size was set to $\gamma = |TOP| + |MID| + |BOT| = 5 \times |P|$, with the sizes of sets *TOP*, *MID*, and *BOT* set to $0.15 \times \gamma$, $0.7 \times \gamma$, and $0.15 \times \gamma$, respectively, as suggested in Brandão et al. (2015), Buriol et al. (2005, 2007), Chan et al. (2015), Gonçalves et al. (2009, 2016), Gonçalves and Resende (2011), Noronha et al. (2011), and Zheng et al. (2014).

4. Computational experiments

In this section, we report computational experiments to assess the performance of the BRKGA GA-KEY. This algorithm was implemented in C++ and compiled with GNU C++ version 4.6.3. The experiments were performed on a 3.40 GHz i7-4770 Intel Core CPU with 16 GB of RAM memory.

Our experiments were carried out on the same set of instances used by Shokripour et al. (2012). The six instances have 50 workers and the value of W is equal to 50, 250, 450, 650, 850, and 1050. The values g_i, s_i, and w_i, $i = 1, \ldots, |P|$, were randomly generated, with $G_i = 20 \times w_i$.

Since Shokripour et al. (2012) reported numerical results showing that heuristic CBM outperformed both LPAP (Beaumont et al., 2003b) and ESCR (Hsu et al., 2008), the experiments in this section will address exclusively the comparison between the newly proposed GA-KEY genetic algorithm and CBM.

In the first experiment, we investigate if GA-KEY efficiently identifies the relationships between keys and good solutions, converging to better solutions than those obtained by CBM (Shokripour et al., 2012). To do so, we compare the performance of GA-KEY with that of a multi-start procedure (MS-KEY) that uses the same decoding heuristic. Each iteration of MS-KEY applies the decoding heuristic starting from a randomly generated vector of random keys. Therefore, nothing is learned from one iteration to the next. GA-KEY and MS-KEY were run ten times for each instance, with different seeds for the random number generator (Matsumoto et al., 1998). GA-KEY was made to stop after $|P|$ generations without improvement in the cost of the best solution found, while MS-KEY after $|P| \times \gamma$ iterations without improvement, where γ is the population size of GA-KEY.

Table 1 reports the results of this experiment. The first column identifies the load W of each instance. The two next columns provide the makespan and the running time in seconds obtained by the CBM heuristic. The three next columns provide average results over ten runs for the makespan and the running time in seconds obtained by MS-KEY, as well as the relative improvement obtained in the makespan by MS-KEY with respect to that previously computed by CBM. The three last columns display average results over ten runs for the makespan and the running time in seconds obtained by GA-KEY, as well as the relative improvement obtained in the makespan by GA-KEY with respect to that computed by CBM. The average relative improvement of GA-KEY over CBM

Table 1
Results of GA-KEY and MS-KEY compared with those of CBM

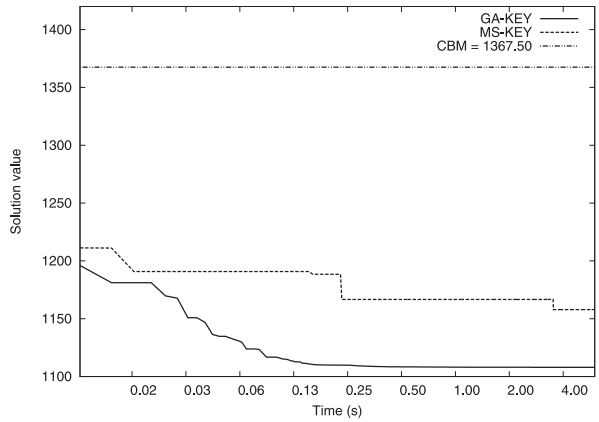
	CBM		MS-KEY			GA-KEY		
W	Makespan	Time (seconds)	Makespan	Gap (%)	Time (seconds)	Makespan	Gap (%)	Time (seconds)
50	1367.50	0.01	1183.85	13.43	0.25	1109.77	18.85	0.20
250	4614.69	0.01	4124.45	10.62	0.24	3954.85	14.30	0.21
450	7537.98	0.01	6865.69	8.92	0.24	6599.60	12.45	0.21
650	10,203.79	0.01	9600.56	5.91	0.28	9228.82	9.55	0.21
850	12,887.14	0.01	12,333.86	4.29	0.25	11,861.24	7.96	0.21
1050	15,578.63	0.01	15,065.73	3.29	0.25	14,492.70	6.97	0.22
Average	8698.29	0.01	8195.69	7.74	0.25	7874.50	11.68	0.21

amounted to up to 18.85%, observed for $W = 50$. The average relative improvements observed for MS-KEY and GA-KEY were, respectively, 7.74% and 11.68%.

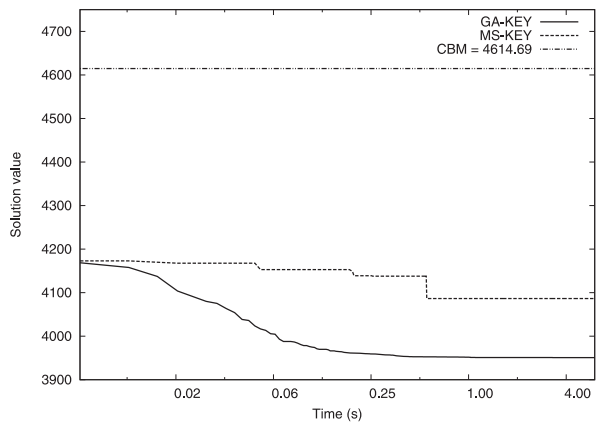
Figures 5 and 6 display six plots exhibiting how the value of the best solution obtained by each of the genetic algorithms MS-KEY and GA-KEY evolves with the running time and compare them with that obtained by CBM, for each of the instances considered in the experiment. They show that GA-KEY systematically finds better solutions faster and that it converges to its best value before one second for all test instances.

In the second experiment, we evaluate and compare the run time distributions (or time-to-target plots—or ttt-plots, for short) of MS-KEY and GA-KEY. ttt-plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. Run time distributions have also been advocated by Hoos and Stützle (1998) as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. In this experiment, both MS-KEY and GA-KEY were made to stop whenever a solution with cost smaller than or equal to a given target value was found. The target is set as the cost of the best-known solution for the instance. The heuristics were run 200 times each, with different initial seeds for the pseudo-random number generator. Next, the empirical probability distributions of the time taken by each heuristic to find a target solution value are plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by (Aiex et al., 2002, 2007). We associate a probability $p_i = (i - \frac{1}{2})/200$ with the ith smallest running time t_i and plot the points (t_i, p_i), for $i = 1, \ldots, 200$. The more to the left is a plot, the better is the algorithm corresponding to it.

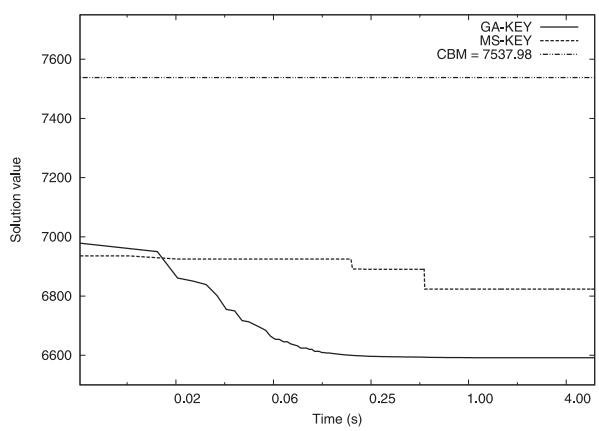
Time-to-target plots for the six instances considered in this study are shown in Figs. 7 and 8. Figure 7 shows that GA-KEY finds solutions with makespan equal to 1109.28, 3954.92, and 6595.53 for the three first instances with probability close to 100% in less than one second of running time, while MS-KEY may take up to 60 seconds to find solutions as good as those with the same probability. Similar results are observed for the other instances in Fig. 8: GA-KEY finds solutions with makespan equal to 9230.49, 11860.48, and 14492.37 for the three last instances with probability close to 100% in less than five seconds of running time, while MS-KEY may take up to 120 seconds to find same quality solutions with the same probability. We used the tool proposed by Ribeiro et al. (2012) to perform a direct numerical comparison of MS-KEY and GA-KEY.



(a) $W = 50$

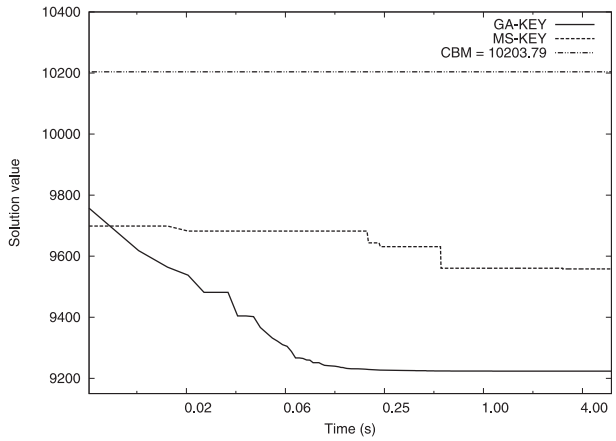


(b) $W = 250$

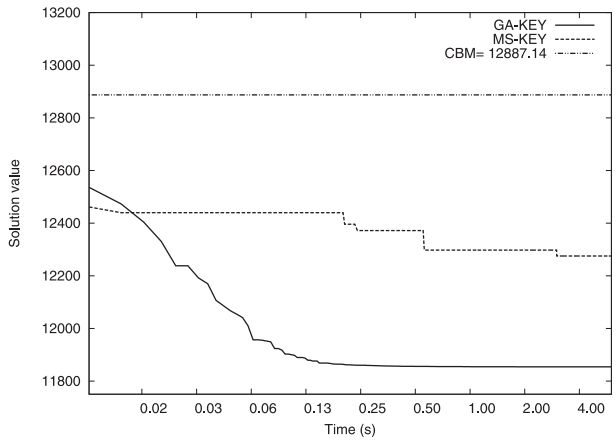


(c) $W = 450$

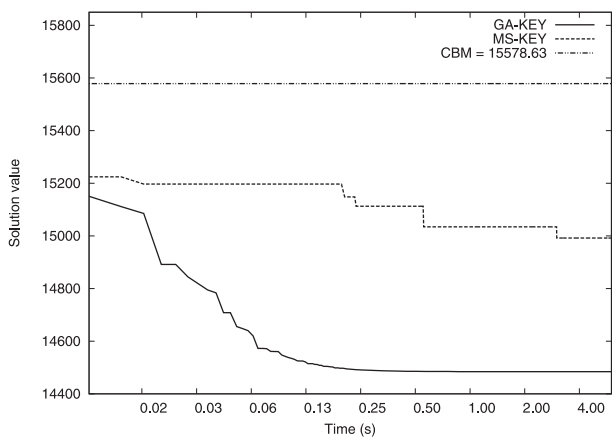
Fig. 5. Best solution value versus running time along the first six seconds for CBM, MS-KEY, and GA-KEY.



(a) $W = 650$

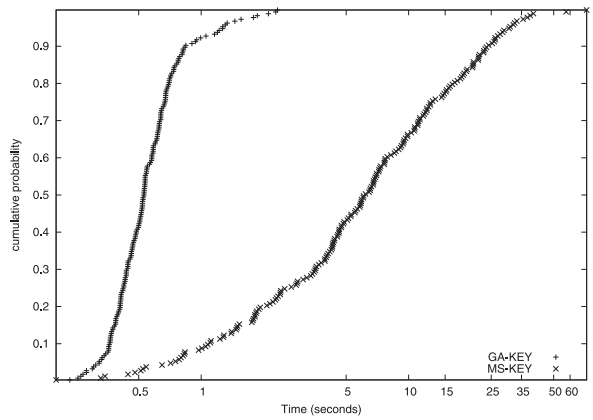


(b) $W = 850$

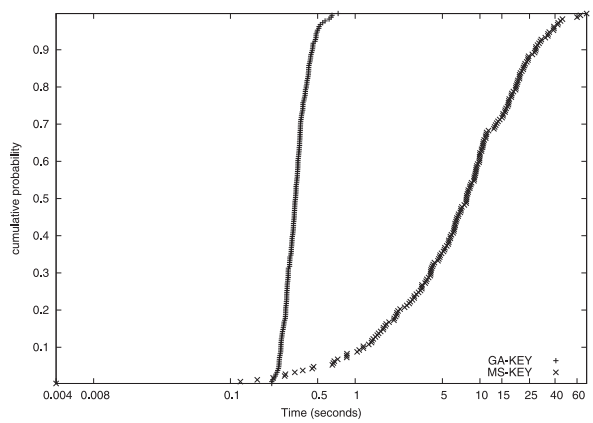


(c) $W = 1050$

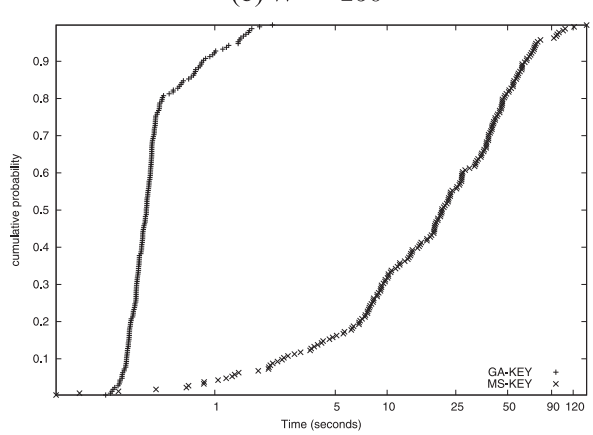
Fig. 6. Best solution value versus running time along the first six seconds for CBM, MS-KEY, and GA-KEY.



(a) $W = 50$

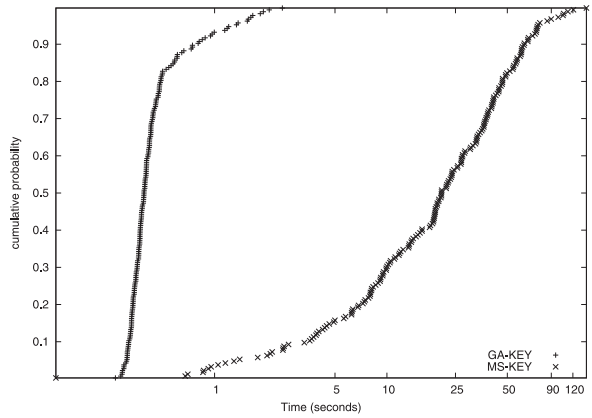


(b) $W = 250$

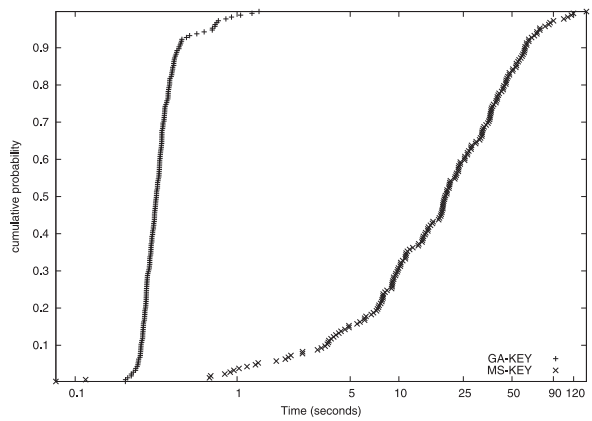


(c) $W = 450$

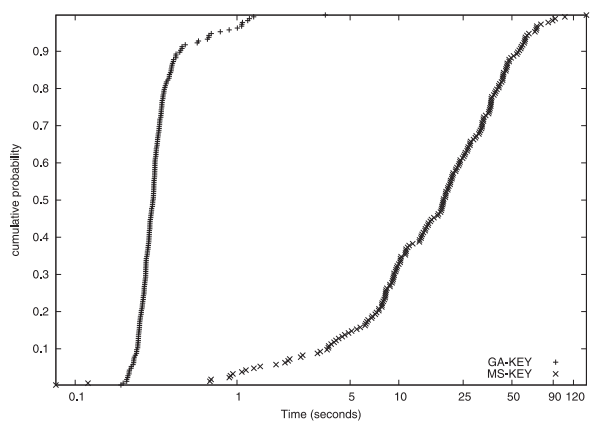
Fig. 7. ttt-plots for instances (a) $W = 50$, target = 1109.28, and $P(T_{\text{GA-KEY}} \leq T_{\text{MS-KEY}}) = 0.96$; (b) $W = 250$, target = 3954.92, and $P(T_{\text{GA-KEY}} \leq T_{\text{MS-KEY}}) = 0.97$; and (c) $W = 450$, target = 6595.53, and $P(T_{\text{GA-KEY}} \leq T_{\text{MS-KEY}}) = 0.98$.



(a) $W = 650$



(b) $W = 850$



(c) $W = 1050$

Fig. 8. ttt-plots for instances (a) $W = 650$, target = 9230.49, and $P(T_{\text{GA-KEY}} \leq T_{\text{MS-KEY}}) = 0.99$; (b) $W = 850$, target = 11, 860.48, and $P(T_{\text{GA-KEY}} \leq T_{\text{MS-KEY}}) = 0.99$; and (c) $W = 1050$, target = 14, 492.37, and $P(T_{\text{GA-KEY}} \leq T_{\text{MS-KEY}}) = 0.99$.

Let $T_{\mathrm{GA-KEY}}$ (resp. $T_{\mathrm{MS-KEY}}$) be the continuous random variable representing the time needed by GA-KEY (resp. MS-KEY) to find a solution as good as the target, and let $P(T_{\mathrm{GA-KEY}} \leq T_{\mathrm{MS-KEY}})$ be the probability that GA-KEY finds a solution as good as the target in less time than MS-KEY. These probabilities are computed for the six test instances by the software developed by Ribeiro et al. (2015) and reported in the captions of Figs. 7 and 8. We can see from these results that the probability that GA-KEY finds the target in less time than MS-KEY is greater than or equal to 96% for all test instances, showing that BRKGA identifies the relationships between the keys and the good solutions throughout the evolutionary process.

5. Concluding remarks

We considered the unconstrained multi-round divisible load scheduling problem with dedicated and heterogeneous processors. Data transmission is divided into rounds and every active worker receives a load chunk at each round. Multi-round systems may reduce the makespan by decreasing the idle times, at the additional cost of repeated setup times at each round. A biased random-key genetic algorithm was proposed for approximately solving this problem. This GA-KEY heuristic improves upon the best algorithm in the literature in terms of solution quality, since it performs a global search in the space of processor permutations in order to find the best transmission order of the processors and the best values for the number of active processors and for the number of rounds. Computational experiments showed that the makespans obtained by the proposed heuristic improved those obtained by CBM by 11.68%, on average.

Acknowledgments

The work of Celso C. Ribeiro was partially supported by CNPq research grant 303958/2015-4 and by FAPERJ research grant E-26/201.198/2014. The work of Mauricio G. C. Resende was done when he was employed by AT&T Labs Research.

References

Abib, E.R., Ribeiro, C.C., 2009. New heuristics and integer programming formulations for scheduling divisible load tasks. Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling. Nashville, pp. 54–61.

Aiex, R., Resende, M., Ribeiro, C.C., 2002. Probability distribution of solution time in GRASP: an experimental investigation. *Journal of Heuristics* 8, 343–373.

Aiex, R., Resende, M., Ribeiro, C.C., 2007. TTTPLOTS: a Perl program to create time-to-target plots. *Optimization Letters* 1, 355–366.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing* 2, 154–160.

Beaumont, O., Bonichon, N., Eyraud-Dubois, L., 2008. Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In: *International Symposium on Parallel and Distributed Processing*. IEEE, Miami, pp. 1–7.

Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y., 2005. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Transactions on Parallel and Distributed Systems* 16, 207–218.

Beaumont, O., Legrand, A., Robert, Y., 2003a. The master-slave paradigm with heterogeneous processors. *IEEE Transactions on Parallel and Distributed Systems* 14, 897–908.

Beaumont, O., Legrand, A., Robert, Y., 2003b. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. 12th Heterogeneous Computing Workshop. IEEE Computer Society Press, Nice, pp. 98–111.

Berlinska, J., Drozdowski, M., 2010. Heuristics for multi-round divisible loads scheduling with limited memory. *Parallel Computing* 36, 199–211.

Berlińska, J., Drozdowski, M., Lawenda, M., 2009. Experimental study of scheduling with memory constraints using hybrid methods. *Journal of Computational and Applied Mathematics* 232, 638–654.

Bharadwaj, V., Ghose, D., Mani, V., 1995. Multi-installment load distribution in tree networks with delays. *IEEE Transactions on Aerospace and Electronic Systems* 31, 555–567.

Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G., 1996. *Scheduling divisible loads in parallel and distributed systems*, Wiley, IEEE Computer Society Press.

Blazewicz, J., Drozdowski, M., 1997. Distributed processing of divisible jobs with communication startup costs. *Discrete Applied Mathematics* 76, 21–41.

Brandão, J.S., Noronha, T.F., Resende, M.G.C., Ribeiro, C.C., 2015. A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions Operational Research* 22, 823–839.

Buriol, L.S., Resende, M.G., Ribeiro, C.C., Thorup, M., 2005. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks* 46, 36–56.

Buriol, L.S., Resende, M.G., Thorup, M., 2007. Survivable IP network design with OSPF routing. *Networks* 49, 51–64.

Chan, F., Tibrewal, R.K., Prakash, A., Tiwari, M., 2015. A biased random key genetic algorithm approach for inventory-based multi-item lot-sizing problem. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 229, 157–171.

Drozdowski, M., 1997. *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*. Politechnika Poznanska, Poznan.

Drozdowski, M., Lawenda, M., 2006. Multi-installment divisible load processing in heterogeneous systems with limited memory. *Parallel Processing and Applied Mathematics* 3911, 847–854.

Drozdowski, M., Lawenda, M., 2007. Multi-installment divisible load processing in heterogeneous distributed systems. *Concurrency and Computation: Practice and Experience* 19, 2237–2253.

Drozdowski, M., Wolniewicz, P., 2003. Divisible load scheduling in systems with limited memory. *Cluster Computing* 6, 19–29.

Drozdowski, M., Wolniewicz, P., 2006. Optimum divisible load scheduling on heterogeneous stars with limited memory. *European Journal of Operational Research* 172, 545–559.

Duarte, A., Marti, R., Resende, M.G.C., Silva, R.M.A., 2014. Improved heuristics for the regenerator location problem. *International Transactions in Operational Research* 21, 541–558.

Ericsson, M., Resende, M.G.C., Pardalos, P.M., 2002. A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization* 6, 299–333.

Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C., 2009. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research* 189, 1171–1190.

Gonçalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17, 487–525.

Gonçalves, J.F., Resende, M.G.C., 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics* 145, 500–510.

Gonçalves, J.F., Resende, M.G.C., 2014. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research* 21, 215–246.

Gonçalves, J.F., Resende, M.G.C., 2004. An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering* 47, 247–273.

Gonçalves, J.F., Resende, M.G.C., Costa, M.D., 2016. A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research* 23, 25–46. Available at http://dx.doi.org/10.1111/itor.12109

Hoos, H., Stützle, T., 1998. Evaluation of Las Vegas algorithms—pitfalls and remedies. In Cooper, G., Moral, S. (eds), Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Madison, pp. 238–245.

Hsu, C.H., Chen, T.L., Park, J.H., 2008. On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems. *The Journal of Supercomputing* 45, 129–150.

Kim, H.J., 2003. A novel optimal load distribution algorithm for divisible loads. *Cluster Computing —The Journal of Networks Software Tools and Applications* 6, 41–46.

Li, X., Bharadwaj, V., Ko, C., 2000. Divisible load scheduling on single-level tree networks with buffer constraints. *IEEE Transactions on Aerospace and Electronic Systems* 36, 1298–1308.

Lin, X., Deogun, J., Lu, Y., Goddard, S., 2008. Multi-round real-time divisible load scheduling for clusters. *High Performance Computing 2008*, Springer, pp. 196–207.

Maciej, D., Marcin, L., 2008. Scheduling multiple divisible loads in homogeneous star systems. *Journal of Scheduling* 11, 347–356.

Matsumoto, M., Nishimura, T., 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8, 3–30.

Noronha, T.F., Resende, M.G.C., Ribeiro, C.C., 2011. A biased random-key genetic algorithm for routing and wavelength assignment. *Journal of Global Optimization* 50, 503–518.

Ribeiro, C.C., Rosseti, I., Vallejos, R., 2012. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization* 54, 405–429.

Ribeiro, C.C., Rosseti, I., Vallejos, R., 2015. tttplots-compare: a perl program to compare time-to-target plots or general runtime distributions of randomized algorithms. *Optimization Letters* 9, 601–614.

Rosenberg, A.L., 2001. Sharing partitionable workloads in heterogeneous nows: greedier is not better. Proceedings of the 3rd IEEE International Conference on Cluster Computing, IEEE Computer Society, p. 124.

Shokripour, A., Othman, M., Ibrahim, H., 2010. A method for scheduling last installment in a heterogeneous multi-installment system. Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, Chengdu, pp. 714–718.

Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S., 2011. A new method for job scheduling in a non-dedicated heterogeneous system. *Procedia Computer Science* 3, 271–275.

Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S., 2012. New method for scheduling heterogeneous multi-installment systems. *Future Generation Computer Systems* 28, 1205–1216.

Spears, W., de Jong, K., 1991. On the virtues of parameterized uniform crossover. In Belew, R., Booker, L. (eds) Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufman, San Mateo, pp. 230–236.

Wang, M., Wang, X., Meng, K., Wang, Y., 2013. New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. *International Journal of Pattern Recognition and Artificial Intelligence* 27(7).

Wang, X., Wang, Y., Meng, K., 2014. Optimization algorithm for divisible load scheduling on heterogeneous star networks. *Journal of Software* 9, 1757–1766.

Wolniewicz, P., 2003. Divisible job scheduling in systems with limited memory. Ph.D. thesis, Poznan University of Technology, Poznań.

Yang, Y., Casanova, H., 2003a. RUMR: Robust scheduling for divisible workloads. Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing, Seattle, pp. 114–125.

Yang, Y., Casanova, H., 2003b. UMR: a multi-round algorithm for scheduling divisible workloads. Proceedings of the 17th International Parallel and Distributed Processing Symposium, Nice, pp. 24–32.

Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., Legrand, A., 2007. On the complexity of multi-round divisible load scheduling. Technical Report 6096, INRIA Rhône-Alpes.

Yang, Y., van der Raadt, K., Casanova, H., 2005. Multi-round algorithms scheduling divisible loads. *IEEE Transactions on Parallel and Distributed Systems* 16, 1092–1102.

Zheng, J.N., Chien, C.F., Gen, M., 2014. Multi-objective multi-population biased random-key genetic algorithm for the 3-D container loading problem. *Computers & Industrial Engineering* 89, 80–87.