MISTA 2015

A biased random-key genetic algorithm for scheduling divisible loads

Julliany S. Brandão · Thiago F. Noronha · Mauricio G. C. Resende · Celso C. Ribeiro

Abstract A divisible load is an amount $W \geq 0$ of computational work that can be arbitrarily divided into chunks and distributed among a set P of worker processors to be processed in parallel. The Divisible Load Scheduling Problem consists in (a) selecting a subset of active workers, (b) defining the order in which the chunks will be transmitted to each of them, and (c) deciding the amount of load α_i that will be transmitted to each active worker, so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. We propose a biased random-key genetic algorithm for solving the divisible load scheduling problem. Computational results show that the genetic algorithm outperforms the best heuristic in the literature.

Keywords Divisible load scheduling, Random-key genetic algorithms, parallel processing, scientific computing

Julliany S. Brandão
Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil, and Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Av. Maracanã 229, Rio de Janeiro, RJ 20271-110, Brazil
E-mail: jbrandao@ic.uff.br

Thiago F. Noronha
Universidade Federal de Minas Gerais, Avenida Antônio Carlos 6627, Belo Horizonte, MG 24105, Brazil
E-mail: tfn@dcc.ufmg.br

Mauricio G. C. Resende
Mathematical Optimization and Planning, Amazon.com
333 Boren Avenue North, Seattle, WA 98109, USA (work of this author was done when he was employed by AT&T Labs Research)
E-mail: resendem@amazon.com

Celso C. Ribeiro
Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
E-mail: celso@ic.uff.br

1 Introduction

A *divisible load* is an amount $W \geq 0$ of computational work that can be arbitrarily divided and distributed among different processors to be processed in parallel. The processors are arranged in a star topology and the load is stored in a central *master* processor. The latter splits the load into chunks of arbitrary sizes and transmits each of them to other processors, called *workers*. The master does not process the load itself.

The master can only send load to one worker at a time. Any worker can only start processing after it has completely received its respective chunk of the load. The workers are heterogeneous in terms of processing power, communication speed, and setup time for start communicating with the master. Not all available processors should necessarily be used for processing the load.

The Divisible Loading Scheduling Problem (DLSP) was introduced in [12], motivated by an application in intelligent sensor networks. Applications of DLSP arise from a number of scientific problems, such as parallel database searching [11], parallel image processing [25], parallel video encoding [26,35], processing of large distributed files [37], and task scheduling in cloud computing [28], among others.

In this work, we deal with the same DLSP variant treated in [1, 10]. Let $P = \{1, ..., n\}$ be the set of worker processors indices. Each worker $i \in P$ has (i) a setup time $g_i \geq 0$ to start the communication with the master, (ii) a communication time $G_i \geq 0$ needed to receive each unit of the load from the master and (iii) a processing time $w_i \geq 0$ needed to process each unit of the load. Therefore, it takes $g_i + \alpha_i \cdot G_i$ units of time for the master to transmit a load chunk of size $\alpha_i \geq 0$ to the worker $i \in P$. Furthermore, it takes an additional $w_i \cdot \alpha_i$ units of time for this worker to process the chunk of load assigned to it.

The scheduling problem consists of (a) selecting a subset $A \subseteq P$ of active workers, (b) defining the order in which the chunks will be transmitted to each active worker and (c) deciding the amount of load α_i that will be transmitted to each worker $i \in A$, so as to minimize the makespan, i.e., the total elapsed time since the master began to send data to the first worker, until the last worker stops its computations. This problem was proved to be NP-hard in [42].

Since the load can be split arbitrarily, all workers stop at the same time in the optimal solution [5]. In addition, if the order in which the chunks are transmitted to the workers is fixed, then the best solution can be computed in $O(n)$ time, using the AlgRap algorithm developed in [1]. In other words, given a permutation of the workers in P, AlgRap computes the set of active workers and the amount of load that has to be sent to each of them to minimize the makespan. Therefore, DLSP can be reduced to the problem of finding the best permutation of the processors, i.e., the one which induces the minimum makespan. In order to find this permutation, we propose a biased random-key genetic algorithm [18, 19, 22], which has been successfully used for solving many permutation based combinatorial optimization problems [17, 19–21, 23, 30].

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

2 Related work

There are many variants of DLSP in the literature. Divisible load scheduling may be performed in a *single round* or in *multiple rounds*. In the single round case [1, 3–5, 7, 9, 10, 13, 15, 16, 24, 27, 32, 36, 38, 39], each active worker receives and processes a single chunk of the load. In the multi-round case [1, 4, 6, 8, 14–16, 31, 33, 39–41], each active worker receives and processes multiple chunks of load. After the master finishes the transmission of the first round of load to all active workers, it immediately starts the transmission of the next batch of load chunks following the same order, until all the load is distributed among the workers.

The workers may be *homogeneous* or *heterogeneous*. If the workers are homogeneous, the values of g_i, G_i, and w_i are the same for all processors $i \in P$ [4,8,9,24,40,41]. Contrarily, in the heterogeneous case the values of g_i, G_i, and w_i may be different for each worker [1, 3, 5, 6, 10, 13–16, 27, 31–33, 36, 38–41].

The system can be *dedicated* or *non-dedicated*. When the system is dedicated, it is assumed that all resources (processors, memory, network, etc.) are used to process a single computational load [1, 5–7, 9, 10, 13, 15, 16, 31, 36, 38–41]. Non-dedicated systems may be used to simultaneously process different computational loads [6, 7, 32].

In this work, we consider the single round DLSP with dedicated and heterogeneous workers [1, 4, 5, 10, 15, 38, 42], that was proved to be *NP*-hard in [42]. Blazewicz and Drozdowski [10] showed that once a permutation of the workers is given, a solution with minimum makespan can be obtained in $O(n \log n)$ time, where $n = |P|$ is the number of workers. Later, Abib and Ribeiro [1] proposed the faster AlgRap algorithm that finds this solution in $O(n)$ time.

Non-linear programming formulations for the single round DLSP with dedicated and heterogeneous workers was proposed in [13]. The first mixed integer linear programming formulation was proposed in [4] and was later improved and extended in [1]. Computational experiments reported in [1] have shown that the CPLEX branch-and-cut algorithm based on this formulation was able to find optimal solutions for 490 out of 720 instances with up to 160 processors. However, for the largest unsolved instances, the computational gaps were very high, which motivated the development of heuristics for solving DLSP.

To the best of our knowledge, the best heuristic in the literature for the DLSP variant studied in this paper is the HeuRet algorithm of Abib and Ribeiro [1]. At each iteration, their algorithm (i) estimates the *performance* e_i of each worker in $i \in P$ and (ii) builds a solution by taking the processors in P in a non-increasing order of the e_i values. The algorithm sets $e_i = G_i$, for all $i \in P$, in the first iteration and makes use of the exact AlgRap algorithm to compute an initial solution s_0 with makespan T_0. Next, at each forthcoming iteration k, the estimated performance e_i of each worker $i \in P$ is updated from the values of g_i, w_i, G_i, and T_{k-1} and a new solution with makespan T_k is built by algorithm AlgRap considering the new order defined on the workers. The procedure stops when $T_{k-1} < T_k$, i.e. when the new solution degenerates the makespan of the previous one.

The heuristic proposed in the next section improves upon HeuRet because, instead of defining a greedy local search on the performance estimation of the workers, it performs a global search in the space of worker permutations in order to find the permutation that induces the optimal or a near-optimal solution.

3 Biased random-key genetic algorithm

Genetic algorithms with random keys, or random-key genetic algorithms (RKGA), were first introduced in [2] for combinatorial optimization problems whose solutions can be represented by a permutation vector. 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 a RKGA. Parents are allowed to be selected for mating more than once in a given generation.

A biased random-key genetic algorithm (BRKGA) differs from a RKGA in the way parents are selected for crossover [19]. 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 non-elite solution. The selection is said biased because one parent is always an elite individual and has a higher probability of passing its genes to the new generation.

The BRKGA-DLS biased random-key genetic algorithm for DLSP evolves a population of chromosomes that consists of vectors of real numbers (keys). Each solution is represented by a $|P|$-vector, in which each component is a real number in the range $[0, 1]$ associated with a worker processor in P. Each solution represented by a chromosome is decoded by a heuristic that receives the vector of keys and builds a feasible solution for DLSP using algorithm AlgRap [1]. Decoding consists of two steps: first, the processors are sorted in a non-decreasing order of their random keys; next, the resulting order is used as the input for AlgRap. The makespan of the solution provided by AlgRap is used as the fitness of the chromosome.

We use the parametric uniform crossover scheme proposed in [34] 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 0.60 and from the least fit parent with probability 0.40. Instead of the mutation operator, the concept of mutants is used: a fixed number of mutant solutions are introduced in the population in each generation, randomly generated in the same way as in the initial population.

The keys associated to each worker 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 on the current population. As illustrated in Figure 1, 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 biased random-key genetic algorithm from the random-key genetic algorithm of Bean [2], 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. $|MID| = |REST| - |BOT|$ offspring solutions are created. In our implementation, the population size was set to $|TOP| + |MID| + |BOT| = 5 \times |P|$, with the sizes of sets TOP, MID, and BOT set to $0.15 \times 5 \times |P|$, $0.7 \times 5 \times |P|$, and $0.15 \times 5 \times |P|$, respectively, as suggested by Noronha et al. [30].

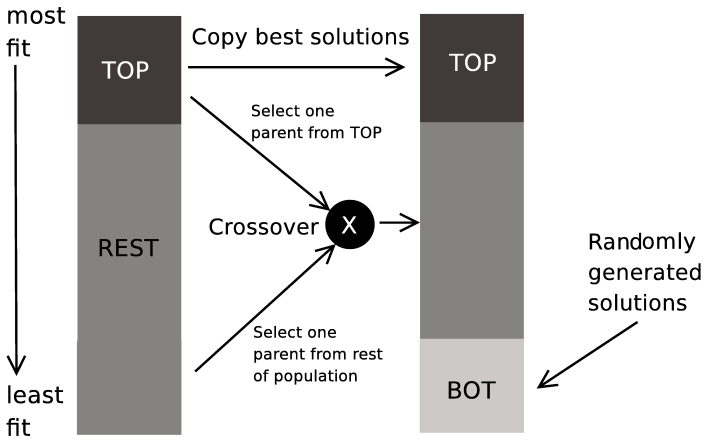

Fig. 1 Population evolution between consecutive generations of a BRKGA.

4 Computational experiments

We report computational experiments to assess the performance of the biased random-key genetic algorithm BRKGA-DLS. This algorithm was implemented in C++. The experiments were performed on a Quad-Core AMD Opteron(tm) Processor 2350, with 16 GB of RAM memory.

The set of instances used in the experiments was proposed in [1]. There are 120 grid configurations with $n = 10, 20, 40, 80, 160$ worker processors and eight combinations of the parameter values g_i, G_i and w_i, $i = 1, \ldots, n$, each of them ranging either in the interval [1, 100] or in the interval [1000, 100000]. Three instances were generated for each combination of n, g_i, G_i, and w_i. Six different values of the load $W = 100, 200, 400, 800, 1600, 3200$ are considered for each grid configuration, corresponding to 18 instances for each combination of parameters g_i, G_i and w_i, and amounting to a total of 720 test instances. Each heuristic was run ten times for each instance, with different seeds for the random number generator of [29].

The first experiment consisted in evaluating if BRKGA-DLS efficiently identifies the relationships between keys and good solutions and converges faster to near-optimal solutions. We compare its performance with that of a multi-start procedure that uses the same decoding heuristic as BRKGA-DLS. Each iteration of the multi-start procedure, called MS-DLS, applies the same decoding heuristic of BRKGA-DLS, but using randomly generated values for the keys. In this experiment, BRKGA-DLS was run for 1000 generations and MS-DLS for $1000 \times q$ iterations, where $q = 5 \times |P|$ is the population size of BRKGA-DLS. The average solution values found by BRKGA-DLS were 4.95% better than those provided by MS-DLS over the 720 instances. Also, the worst (resp. best) solution values found by BRKGA-DLS were 5.98% (resp. 3.78) smaller than the respective worst (resp. best) solution values obtained with MS-DLS. These results indicate that BRKGA-DLS identifies the relationships between keys and good solutions, making the evolutionary process converge to better solutions faster than MS-DLS.

In the second experiment, we compared BRKGA-DLS with HeuRet and MS-DLS. We first evaluated how many optimal solutions have been obtained by each heuristic

over the 720 test instances. The default CPLEX branch-and-cut algorithm based on the formulation in [1] was also run for the 720 instances. Version 12.6 of CPLEX was used and the maximum CPU time was set to 24 hours. CPLEX was able to prove the optimality for 497 out of the 720 test instances. The first line of Table 1 shows that BRKGA-DLS found optimal solutions for 413 instances (i.e. 83.1%) out of the 497 instances for which the optimal solutions are known, while HeuRet found 320 optimal solutions and MS-DLS found only 177 of them. The second line of Table 1 gives the number of instances for which each heuristic found the best known solution value. The third line of this table shows the number of runs for which BRKGA-DLS and MS-DLS found the best known solution values (we recall that HeuRet is a deterministic algorithm, while the others are randomized). Finally, the last line of this table gives, for each of the three heuristics, a score that represents the sum over all instances of the number of methods that found strictly better solutions than the specific heuristic being considered. The lower a score is, the best the corresponding heuristic is. It can be seen that BRKGA outperformed both HeuRet and MS-DLS heuristics with respect to the number of optimal and best solutions found, as well as with respect to the score value. In particular, BRKGA-DLS obtained better scores than HeuRet for all but one instance and found the best known solution values for 645 out of the 720 test instances, while HeuRet found the best solution values for only 313 instances.

Table 1 Summary of the numerical results obtained with BRKGA-DLS, HeuRet and MS-DLS for 720 test instances.

	MS-DLS	HeuRet	BRKGA-DLS
Optimal values (over 497 instances)	177	320	413
Best values (over 720 instances)	189	313	645
Best values (over 7200 runs)	2166	-	6191
Score value	803	112	1

The third and last experiment provides a more detailed comparison between HeuRet and BRKGA-DLS, based on 20 larger and more realistic instances with $|P| = 320$ and $W = 10,000$. The values of G_i and g_i have been randomly generated in the ranges $[1, 100]$and $[100, 100, 000]$, respectively. However, differently from [1], the values of w_i have been randomly generated in the interval $[200, 500]$. These values are more realistic, since the processing rate of a real computer is always larger than its communication rate. In this experiment, BRKGA-DLS was made to stop after $|P|$ generations without improvement in the best solution found. The results are reported in Table 2. The first column shows the instance name. The second and third columns display, respectively, the makespan and the computation time (in seconds) obtained by HeuRet. The next two columns provide the average makespan over ten runs of BRKGA, the corresponding coefficient of variation, defined as the ratio of the standard deviation to the average. The average computation time in seconds of BRKGA over ten runs is given in the sixth column. The last column shows the percent relative reduction between the average solution found by BRKGA-DLS with respect to that found by MS-DLS. It can be seen that the average makespan obtained by BRKGA-DLS is always smaller than that given by HeuRet. In addition, the coefficient of variation of BRKGA-DLS is very small, indicating that it is a robust heuristic. The percent relative reduction of BRKGA-DLS with respect to MS-DLS amounted to 3.19% for instance dls.320.10 and to 2.38% on average.

As in real-life applications the load size may be very large, and the total communication and processing times may take many hours, reductions in the elapsed time of this magnitude may be very significant. Although the running times of BRKGA-DLS are larger than those of HeuRet, their average values never exceeded the time taken by HeuRet by more than 30 seconds. Since practical applications of parallel processing take long elapsed times, the trade-off between the reduction in the elapsed time and this small additional running time needed by BRKGA accounts very favorably to BRKGA-DLS. In addition, we notice that the BRKGA-DLS genetic algorithm itself may be efficiently parallelized with a high speed-up and distributed over the set P of processors, making its computation time irrelevant when compared with that of the parallel application.

Table 2 BRKGA vs. HeuRet on the largest instances with 320 processors.

	HeuRet		BRKGA			
Instance	makespan	time (s)	makespan	CV (%)	time (s)	reduction (%)
dls.320.01	312813.64	0.01	306613.22	0.04	27.24	1.98
dls.320.02	321764.07	0.01	313847.15	0.07	16.72	2.46
dls.320.03	402264.85	0.01	392059.46	0.11	23.72	2.54
dls.320.04	348474.15	0.01	341436.89	0.02	28.16	2.02
dls.320.05	342086.46	0.01	334946.37	0.03	21.67	2.09
dls.320.06	311824.17	0.01	305601.28	0.02	21.93	2.00
dls.320.07	325732.30	0.01	316467.42	0.02	23.19	2.84
dls.320.08	323171.95	0.01	315065.11	0.03	26.23	2.51
dls.320.09	312326.77	0.01	305948.81	0.02	25.03	2.04
dls.320.10	296984.47	0.01	287521.34	0.12	24.04	3.19
dls.320.11	290559.21	0.01	284822.15	0.04	20.56	1.97
dls.320.12	343076.56	0.01	333085.72	0.05	19.53	2.91
dls.320.13	287276.21	0.01	281525.27	0.04	22.77	2.00
dls.320.14	311054.47	0.01	303796.42	0.06	26.81	2.33
dls.320.15	362369.67	0.01	352642.18	0.04	20.41	2.68
dls.320.16	287083.60	0.01	281082.29	0.09	25.38	2.09
dls.320.17	339666.43	0.01	329893.61	0.04	23.53	2.88
dls.320.18	368795.14	0.01	361281.06	0.07	22.08	2.04
dls.320.19	347671.73	0.01	338075.70	0.03	27.59	2.76
dls.320.20	372427.24	0.01	364013.40	0.06	18.28	2.26
Averages	330371.15	0.01	322486.24	0.05	23.24	2.38

5 Conclusions

We considered the single round divisible load scheduling problem with dedicated and heterogeneous workers. A new biased random-key genetic algorithm has been proposed for the problem.

The BRKGA-DLS heuristic improves upon the best heuristic in the literature, since it performs a global search in the space of worker permutations in order to find the best order in which the active worker processors will receive load from the master.

Computational experiments on 720 test instances with up to 160 processors have shown that BRKGA-DLS found optimal solutions for 413 instances (out of the 497 instances where the optimal solution is known), while the HeuRet heuristic found optimal solutions for only 320 of them. Moreover, BRKGA-DLS obtained better scores than

HeuRet for all but one instance and found solutions as good as the best known solution for 645 out of the 720 test instances. To summarize, BRKGA-DLS outperformed the previously existing HeuRet heuristic with respect to all measures considered in Table 1.

For the new set of larger and more realistic instances with 320 processors, BRKGA-DLS found solution values 2.38% better than HeuRet on average. In addition, the processing times of BRKGA-DLS are relatively small and never exceeded 30 seconds. Therefore, parallel processing applications dealing with large amounts of data and taking long elapsed times can benefit from BRKGA-DLS, since the additional running time needed by BRKGA-DLS may result in a significant reduction in the makespan.

References

1. Abib, E.R., Ribeiro, C.C.: New heuristics and integer programming formulations for scheduling divisible load tasks. In: Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling, pp. 54–61. Nashville (2009)
2. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing **2**, 154–160 (1994)
3. Beaumont, O., Bonichon, N., Eyraud-Dubois, L.: Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In: International Symposium on Parallel and Distributed Processing, pp. 1–7. IEEE, Miami (2008)
4. Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Transactions on Parallel and Distributed Systems **16**, 207–218 (2005)
5. Beaumont, O., Legrand, A., Robert, Y.: Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In: 12th Heterogeneous Computing Workshop, pp. 98–111. IEEE Computer Society Press, Nice (2003)
6. Berlinska, J., Drozdowski, M.: Heuristics for multi-round divisible loads scheduling with limited memory. Parallel Computing **36**, 199–211 (2010)
7. Berlińska, J., Drozdowski, M., Lawenda, M.: Experimental study of scheduling with memory constraints using hybrid methods. Journal of Computational and Applied Mathematics **232**, 638–654 (2009)
8. Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment load distribution in tree networks with delays. IEEE Transactions on Aerospace and Electronic Systems **31**, 555–567 (1995)
9. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press (1996)
10. Blazewicz, J., Drozdowski, M.: Distributed processing of divisible jobs with communication startup costs. Discrete Applied Mathematics **76**, 21–41 (1997)
11. Błażewicz, J., Drozdowski, M., Markiewicz, M.: Divisible task scheduling–concept and verification. Parallel Computing **25**, 87–98 (1999)
12. Cheng, Y.C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems **24**, 700–712 (1988)
13. Drozdowski, M.: Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems. 321. Politechnika Poznanska (1997)
14. Drozdowski, M., Lawenda, M.: Multi-installment divisible load processing in heterogeneous systems with limited memory. Parallel Processing and Applied Mathematics **3911**, 847–854 (2006)
15. Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing **6**, 19–29 (2003)
16. Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research **172**, 545–559 (2006)
17. Duarte, A., Mart, R., Resende, M., Silva, R.: Improved heuristics for the regenerator location problem. International Transactions in Operational Research **21**, 541–558 (2014)
18. Ericsson, M., Resende, M.G.C., Pardalos, P.M.: A genetic algorithm for the weight setting problem in OSPF routing. Journal of Combinatorial Optimization **6**, 299–333 (2002)
19. Gonçalves, J.F., Resende, M.G.: Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics **17**, 487–525 (2011)

20. Gonçalves, J.F., Resende, M.G.: A biased random key genetic algorithm for 2D and 3D bin packing problems. International Journal of Production Economics **145**, 500–510 (2013)
21. Gonçalves, J.F., Resende, M.G.: An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. International Transactions in Operational Research **21**, 215–246 (2014)
22. Gonçalves, J.F., Resende, M.G.C.: An evolutionary algorithm for manufacturing cell formation. Computers and Industrial Engineering **47**, 247–273 (2004)
23. Gonçalves, J.F., Resende, M.G.C., Costa, M.D.: A biased random-key genetic algorithm for the minimization of open stacks problem. International Transactions in Operational Research (2015). DOI 10.1111/itor.12109
24. Kim, H.J.: A novel optimal load distribution algorithm for divisible loads. Cluster Computing- The Journal of Networks Software Tools and Applications **6**, 41–46 (2003)
25. Lee, C.k., Hamdi, M.: Parallel image processing applications on a network of workstations. Parallel Computing **21**, 137–160 (1995)
26. Li, P., Veeravalli, B., Kassim, A.A.: Design and implementation of parallel video encoding strategies using divisible load analysis. IEEE Transactions on Circuits and Systems for Video Technology **15**, 1098–1112 (2005)
27. Li, X., Bharadwaj, V., Ko, C.: Divisible load scheduling on single-level tree networks with buffer constraints. IEEE Transactions on Aerospace and Electronic Systems **36**, 1298–1308 (2000)
28. Lin, W., Liang, C., Wang, J.Z., Buyya, R.: Bandwidth-aware divisible task scheduling for cloud computing. Software: Practice and Experience **44**, 163–174 (2014)
29. Matsumoto, M., Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation **8**, 3–30 (1998)
30. Noronha, T.F., Resende, M.G.C., Ribeiro, C.C.: A biased random-key genetic algorithm for routing and wavelength assignment. Journal of Global Optimization **50**, 503–518 (2011)
31. Shokripour, A., Othman, M., Ibrahim, H.: A method for scheduling last installment in a heterogeneous multi-installment system. In: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, pp. 714–718. Chengdu (2010)
32. Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: A new method for job scheduling in a non-dedicated heterogeneous system. Procedia Computer Science **3**, 271–275 (2011)
33. Shokripour, A., Othman, M., Ibrahim, H., Subramaniam, S.: New method for scheduling heterogeneous multi-installment systems. Future Generation Computer Systems **28**, 1205–1216 (2012)
34. Spears, W., deJong, K.: On the virtues of parameterized uniform crossover. In: R. Belew, L. Booker (eds.) Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236. Morgan Kaufman, San Mateo (1991)
35. Turgay, A., Yakup, P.: Optimal scheduling algorithms for communication constrained parallel processing. Lecture Notes in Computer Science **2400**, 197–206 (2002)
36. Wang, M., Wang, X., Meng, K., Wang, Y.: New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. International Journal of Pattern Recognition and Artificial Intelligence **27** (2013)
37. Wang, R., Krishnamurthy, A., Martin, R., Anderson, T., Culler, D.: Modeling communication pipeline latency. ACM Sigmetrics Performance Evaluation Review **26**, 22–32 (1998)
38. Wang, X., Wang, Y., Meng, K.: Optimization algorithm for divisible load scheduling on heterogeneous star networks. Journal of Software **9**, 1757–1766 (2014)
39. Wolniewicz, P.: Divisible job scheduling in systems with limited memory. Ph.D. thesis, Poznan University of Technology, Poznań (2003)
40. Yang, Y., Casanova, H.: RUMR: Robust scheduling for divisible workloads. In: Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing, pp. 114–125. Seattle (2003)
41. Yang, Y., Casanova, H.: UMR: A multi-round algorithm for scheduling divisible workloads. In: Proceedings of the 17th International Parallel and Distributed Processing Symposium, pp. 24–32. Nice (2003)
42. Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., Legrand, A., et al.: On the complexity of multi-round divisible load scheduling. Tech. Rep. 6096, INRIA Rhône-Alpes (2007)