An Artificial Bee Colony Algorithm for the Unrelated Parallel Machines Scheduling Problem

Francisco J. Rodriguez[1], Carlos García-Martínez[3], Christian Blum[2], and Manuel Lozano[1]

[1] Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
[2] ALBCOM Research Group, Technical University of Catalonia, Barcelona, Spain
[3] Department of Computing and Numerical Analysis, University of Córdoba, Córdoba, Spain
fjrodriguez@decsai.ugr.es, cgarcia@uco.es, cblum@lsi.upc.edu, lozano@decsai.ugr.es

Abstract. In this work, we tackle the problem of scheduling a set of jobs on a set of non-identical parallel machines with the goal of minimising the total weighted completion times. Artificial bee colony (ABC) algorithm is a new optimization technique inspired by the intelligent foraging behaviour of honey-bee swarm. These algorithms have shown a better or similar performance to those of other population-based algorithms, with the advantage of employing fewer control parameters. This paper proposes an ABC algorithm that combines the basic scheme with two significant elements: (1) a local search method to enhance the exploitation capability of basic ABC and (2) a neighbourhood operator based on iterated greedy constructive-destructive procedure. The benefits of the proposal in comparison to three different metaheuristic proposed in the literature are experimentally shown.

Keywords: discrete optimisation, metaheuristics, artificial bee colony, unrelated parallel machines schedulling problem.

1 Introduction

The *unrelated parallel machine scheduling with minimising total weighted completion times* (UPMSP) considers a set J of n independent jobs that have to be processed on a set M of m parallel non-identical machines. Each job $j \in J$ has to be processed by exactly one of the m parallel machines and no machine can process more than one job at the same time. A job j is processed on a given machine until completion, i.e., without pre-emption. If a job j is processed on a machine i, it will take a positive integral processing time p_{ij} whose value is determined arbitrarily. The objective is to schedule the jobs in such a way that the sum of the weighted completion times of the jobs is minimised:

$$\text{Minimise} \sum_{i=1}^{n} w_j * C_j,$$

where C_j represents the completion time of job j for a given schedule.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part II, LNCS 7492, pp. 143–152, 2012.

It is important to note that the jobs assigned to a specific machine are processed in non-decreasing order with respect to the ratio between processing time (p_{ij}) and weight (w_j). This order is known as the *weighted shortest processing time* order. According to [1], sequencing the jobs in each machine following this ordering produces an optimal scheduling for this machine.

According to the standard notation proposed by Azizoglu et al. [2] and Allahverdi et al. [3], the family of problems considered in this work is notated in the literature in the following manner: $R_m||\sum w_j * C_j$. Different real-world applications of scheduling on parallel machines can be found in the literature, covering a wide variety of fields. Some of these fields are human resources [4], production management [5,6,7], mail facilities [8], robotized systems [9], sport tournaments [10], and chemical processes [11].

A mixed integer linear programming formulation for the UPMSP problem is provided for the sake of completeness. Let $x_{ji}^t = 1$ if the job j is processed in the tth position (unit time) on machine i and 0, otherwise. And a variable C_{ji}^t denotes the completion time of the job j scheduled in the tth position on machine i. The model is stated by Azizoglu and Kirca [12] as:

$$\textbf{min} \sum_j \sum_i \sum_t w_j \cdot C_{ji}^t \cdot x_{ji}^t \tag{1}$$

$$\textbf{subject to:} \quad \sum_i \sum_t x_{ji}^t = 1 \ \forall j, \tag{2}$$

$$\sum_j x_{ji}^t \leq 1 \ \forall t, i, \tag{3}$$

$$C_{ji}^t = \sum_{r=1}^{n} \sum_{s=1}^{t-1} p_{ir} \cdot x_{ri}^s + p_{ij} \ \forall j, t, i, \tag{4}$$

$$x_{ji}^t \in \{0, 1\} \ \forall j, t, i. \tag{5}$$

In this paper, an approach using the Artificial Bee Colony (ABC) [13,14] method is explored for solving the UPMSP. The ABC algorithm is a new swarm optimization approach that is inspired by the intelligent foraging behaviour of honey-bee swarm. It consists of three essential components: food source positions, nectar-amount and three honey-bee classes (employed bees, onlookers and scouts). Each food source position represents a feasible solution for the problem under consideration. The nectar-amount for a food source represents the quality of such solution (represented by an objective function value). Each bee-class symbolizes one particular operation for generating new candidate food source positions. Specifically, employed bees search the food around the food source in their memory; meanwhile they pass their food information to onlooker bees. Onlooker bees tend to select good food sources from those founded by the employed bees, and then further search the foods around the selected food source. If the employed bee and onlookers associated with a food source cannot find a better neighboring food source, the latter is abandoned and the employed bee

associated with this food source becomes a scout bee that performs a search for discovering new food sources. After the scout finds a new food source, it becomes an employed bee again. Due to its simplicity and ease of implementation, the ABC algorithm has captured much attention. Besides, ABC has exhibited state-of-the-art performances for a considerable number of problems [15,16,17].

The proposed ABC extends the basic scheme by considering two significant elements in order to improve its performance when dealing with the UPMSP. In the first place, after producing neighbouring food sources (in the employed and onlooker bees phases), a local search is applied with a predefined probability to further improve the quality of some of the solutions. Secondly, we propose a new neighbourhood operator based on the solution constructive-destructive process performed by iterated greedy (IG) algorithms [18,19].

The remainder of this paper is organized as follows. In Section 2, we outline different metaheuristics proposed in the literature for the UPMSP. In Section 3, we present in detail the proposed ABC for the UPMSP. In Section 4, we present an empirical study that compares the behaviour of the ABC algorithm with regards to those of other metaheuristics from the literature. Finally, in Section 5, we discuss conclusions and further work.

2 Metaheuristics for the UPMSP

Since the introduction of this problem by McNaughton in [20], it has received much attention and many papers have been published in this area. The research efforts to deal with the SNIM-WCT problem have focused on three main research lines: exact procedures, approximation algorithms through solving relaxations of the problem, and metaheuristics procedures. Concerning the latter, Vredeveld et al. [21] presented two types of neighbourhood functions. The first function is called the *jump neighbourhood*. It consists of selecting a job j and a machine i so that job j is not scheduled on machine i. Then job j is moved to machine i. The second one is called *swap neighbourhood*. For this neighbourhood, two jobs j and k must be selected and assigned to different machines. The corresponding neighbouring solution is obtained by interchanging the machine allocations of the two selected jobs. These two neighbourhood functions are applied in two metaheuristic, a multistart local search and a tabu search.

Recently, Li et al. [22] presented a *genetic algorithm* approach to deal with unrelated parallel machines scheduling using three different performance criteria. In particular, the proposed approach initialises the population adding some solutions generated by heuristics methods. The remaining ones are generated randomly to provide enough diversity. Roulette wheel selection is used to choose a new population with respect to a fitness-proportional probability distribution. The crossover and mutation schemes are those proposed by Cheng et al. [23]. Elitism is considered by removing two chromosomes and adding the best two previous chromosomes to the new generation if they are not selected through the roulette-wheel-selection process. The experimental study performed compares the proposed genetic algorithm with a set of heuristics. Results show that the proposed algorithm outperforms the competing heuristics.

3 Proposed ABC for the UPMSP

In this section, we describe the proposed ABC algorithm for the UPMSP. The general scheme of the proposed approach is outlined in Figure 1. It starts by initialising a population P with $|P| - 1$ random solutions (Initialise()). The remaining solution is initialised by means of a greedy procedure (GreedyProcedure). Then, the following steps are repeated until time limit t_{max} is reached:

- *Employed bees phase.* In this step, employed bees produce new solutions by means of a neighbourhood operator (GenerateNeighbour()). In order to enhance the exploitation capability of ABC, a local search method (LocalSearch()) is applied to the solution obtained by the neighbourhood operator with a probability *probLS*.
- *Onlooker bees phase.* Onlookers bees look for new solutions from solutions of the population selected by means of a binary tournament selection (BinaryTournament()). Specifically, an onlooker bee selects the best food source among two food sources that were randomly picked up from the population. Later, each onlooker bee performs, as in employed bees phase, the neighbourhood operator on the selected solution and a local search procedure.
- *Scout bees phase.* In this phase, the scout bees determine the solutions that has not been improved for *limit* iterations and replace them by new random solutions (Initialise()).

At the end of execution, the best solution found (S_b) is returned by the algorithm (BestSolutionFound()).

3.1 The Greedy Procedure

The greedy constructive procedure used for the initialisation of a solution of the population and the re-construction of partial solutions in the neighbourhood operator works as follows. At each step, it considers placing a unassigned job in any of the machines. For each of these options, it calculates a contribution value according to a predefined heuristic (H). The option which causes the best heuristic value is selected. The procedure stops once all the unassigned jobs are allocated. The heuristic H that guided the greedy procedure is that proposed for this problem in [22]. This heuristic distinguishes two steps. In the first one, a job $\overline{j}_* \in \overline{J}$ (set of unassigned jobs) is selected according to:

$$\overline{j}_* = \text{argmin}\{t_i + p_{ij}/w_j : i = 1, \ldots, m; j = 1, \ldots, n\},$$

once the job $\overline{j}_*$ is selected, machine i_* in which it will be processed is the one that holds:

$$i_* = \text{argmin}\{t_i + p_{i\overline{j}_*}/w_{\overline{j}_*} : i = 1, \ldots, m\},$$

being t_i the completion time of the last job scheduled on the machine i.

```
Input: t_max, P, n_d, probLS, limit,H
Output: S_b
//Number of employed and onlooker bees
NE ← |P|;
NO ← |P|;
//Initialisation phase
for i ← 2 to |P| do
    S_i ← Initialise() ;
end
S_1 ← GreedyProcedure(H);
while computation time limit t_max not reached do
    //Employed bees phase
    for i ← 1 to NE do
        E ← GenerateNeighbour(S_i, n_d, H);
        E' ← LocalSearch(E);
        if E' is better than S_i then
            S_i ← E' ;
        end
    end
    //Onlooker bees phase
    for i ← 1 to NO do
        j ← BinaryTournament(S_1, ..., S_|P|);
        O ← GenerateNeighbour(S_j, n_d, H);
        O' ← LocalSearch(O);
        if O' is better than S_j then
            S_i ← O' ;
        end
    end
    //Scout bees phase
    for i ← 1 to |P| do
        if S_i does not change for limit iterations then
            S_i ← Initialise() ;
        end
    end
    S_b ← BestSolutionFound();
end
```

Fig. 1. ABC scheme

3.2 The Neigbourhood Operator

The proposed neighbourhood operator is based on the constructive-destructive procedure used in IG algorithms [18,19]. IG iteratively tries to refine a solution by removing elements from this solution, by means of a destructive procedure, and reconstructing the resulting partial solution using a greedy constructive procedure. In this case, the proposed neighbourhood operator consists of a unique

Table 1. 12 instance types considered concerning the unrelated parallel machines scheduling problem. The last table column provides the maximum CPU time limit for each instance type (in seconds).

Number of jobs (n)	Number of machines (m)	Time limit (s)
20	5	40
	10	40
50	5	100
	10	100
	20	100
100	5	200
	10	200
	20	200
200	5	400
	10	400
	20	400
	50	400

iteration of the constructive-destructive procedure. In the first place, n_d elements of the current solution are removed (destruction). Then, the partial solution obtained before is reconstructed using a greedy procedure (construction, see Section 3.1). The greedy procedure is guided by the heuristic commented in section 3.1.

4 Computational Experiments

This section describes the computational experiments performed to assess the performance of the ABC model presented in the previous section. Our own algorithms (ABC) as well as all competitor algorithms have been implemented in C++ and the source code has been compiled with gcc 4.5. All experiments were conducted on a computer with a 2.8 GHz Intel i7 processor with 12 GB of RAM running Fedora Linux V15. In this work we considered problem instances from 12 different combinations of the number of jobs (n) and the number of machines (m). These 12 instance types are shown in the first two columns of Table 1. Moreover, the same table shows—in the 3rd column—the maximum CPU time allotted for each instance type ($2 \cdot n$ seconds). For each of the 12 instance types ten problem instances were randomly generated, which is a common choice in recent works dealing with this and related problems [24,25]. The weights of the n jobs were selected uniformly at random from $\{1, \ldots, 10\}$ and the processing time of job j on machine i (p_{ij}, $i = 1, \ldots, n$ and $j = 1, \ldots, m$) was chosen uniformly at random from $\{1, \ldots, 100\}$.

Non-parametric tests [26] have been used to compare the results of the different optimization algorithms under consideration. The only condition to be fulfilled for the use of non-parametric tests is that the algorithms to be compared should have been tested under the same conditions (that is, the same set of problem instances, the same stopping conditions, the same number of runs, etc). Specifically, Wilcoxon's matched-pairs signed-ranks test is used to compare the results of two algorithms.

4.1 Tuning Experiments

In the first place, we have performed an experimental study in order to perform a fine-tuning of the ABC presented above. The goal of the first preliminary experiment is to identify the best combination of the values for the following algorithm parameters:

1. **Population size ($|P|$).** Experiments with $|P|$ in $\{10, 15, 20, 25, 30\}$ were develped.
2. **Destruction size (n_d).** For the percentage of elements dropped from a solution during the neighbourhood operator, values from $\{5\%, 10\%, 15\%, 20\% 25\%\}$ were considered.
3. **Probability of performing local search ($probLS$).** This parameter takes values from $\{0.05, 0.1, 0.2, 0.5, 1\}$).
4. **Local search procedure** ($typeLS$). Three different local search procedures [21] were tested: first-improvement local search with jump moves ($FI - JM$), first-improvement local search with swap moves ($FI - SW$), and first improvement local search with both kinds of moves ($FI - JSM$).
5. **Iterations to determine an exhausted food source ($limit$).** Values from $\{0.25 \cdot n, 0.5 \cdot n, n, 2 \cdot n\}$) were considered, where n is the number of jobs of the instance.

For each combination of values for the different parameters (full factorial design), we applied ABC to each of the 120 problem instances. Through a rank-based analysis on results obtained, we identified the parameter combination with the best average rank over all testing instances. This combination is specified in Table 2.

Table 2. Parameters values

Parameter	Value
Population size ($\|P\|$)	15
Elements dropped (n_d)	25%
Local search procedure ($typeLS$)	FI-JSM
Local search probability ($probLS$)	0.2
Iterations to abandon a food source ($limit$)	0.25 · n

4.2 Comparison with Other Metaheuristics

In this section, we compare ABC to different approaches found in the literature for tackling the UPMSP. More specifically, we considered the following approaches (see also Section 2):

- Iterative multistart method (**MultiS**) [21].
- Tabu search (**Tabu**) [21].
- Genetic algorithm (**GA**) [22].

The parameter values used for each considered algorithm are those recommended in the original works. In order to assure a fair comparison, each algorithm was applied under the same conditions as ABC that is, each algorithm was applied exactly once to each of the 120 problem instances. Moreover, the same CPU time limits were used as with ABC (see Table 1).

Table 3. ABC versus competitors using Wilcoxon's test (level of significance $\alpha = 0.05$, critical value = 13)

Competitor	$R+$	$R-$	Diff.?
GA	78	0	yes
Tabu	76.5	1.5	yes
MultiS	73	5	yes

The results of the considered algorithms in Tables 3 and 4 allow us to make the following observations:

- The proposed ABC statistically outperforms all competing algorithms (see Table 3).
- Concerning the results shown in Table 4, it is important to highlight that ABC obtains the best average results in all the instances. Moreover, the most significant differences with respect to their competitors are obtained on large instances.
- Only MultiS and Tabu are able to match the results of ABC on the smallest problem instances.

Table 4. Results of the studied algorithms averaged over the 10 instances of each of the 12 instance types

n	m	ABC	GA	Tabu	MultiS
200	**50**	**5004**	5138	5062	5063
200	**20**	**16988**	17345	17066	17045
200	**10**	**52919**	53677	53039	52979
200	**5**	**182102**	183400	182166	182130
100	**20**	**5601**	5858	5657	5608
100	**10**	**14921**	15138	14993	14926
100	**5**	**45961**	46412	46014	45062
50	**20**	**1833**	1908	1858	**1833**
50	**10**	**4611**	4737	4655	**4611**
50	**5**	**12625**	12787	12643	**12625**
20	**10**	**1299**	1330	**1299**	**1299**
20	**5**	**2512**	2570	**2512**	**2512**

5 Conclusions and Future Work

In this paper, we presented an ABC algorithm for the UPMSP. The proposed algorithm add a local search procedure and a novel IG-based neighbourhood operator to the basic ABC scheme. This neighbourhood operator is based on the constructive-destructive procedure of IG algorithms. The resulting ABC algorithm has proved to be superior, especially in the case of larger instances, to

three different metaheuristic existing in the literature for this problem. We can conclude from the experiments performed that this algorithm represents a very competitive alternative to the existing methods for the UPMSP.

We believe that the ABC algorithm presented in this paper is a significant contribution, worthy of future study. We will mainly focus on the following avenues of possible research: (1) to adapt the ABC approach for its application to other variants of scheduling problems on parallel machines and (2) to employ the IG-based neighbourhood operator in ABC approaches dealing with other challenging optimisation problems.

Acknowledgements. This work was supported by grant TIN2011-24124 of the Spanish government and by grant P08-TIC-4173 of the Andalusian regional government.

References

1. Elmaghraby, S., Park, S.: Scheduling jobs on a number of identical machines. AIIE Transactions 6(1), 1–13 (1974)
2. Azizoglu, M., Kirca, O.: On the minimization of total weighted flow time with identical and uniform parallel machines. European Journal of Operational Research 113(1), 91–100 (1999)
3. Allahverdi, A., Gupta, J., Aldowaisan, T.: A review of scheduling research involving setup considerations. Omega 27(2), 219–239 (1999)
4. Rosenbloom, E., Goertzen, N.: Cyclic nurse scheduling. European Journal of Operational Research 31, 19–23 (1987)
5. Buxey, G.: Production scheduling: Practice and theory. European Journal of Operational Research 39, 17–31 (1989)
6. Dodin, B., Chan, K.H.: Application of production scheduling methods to external and internal audit scheduling. European Journal of Operational Research 52(3), 267–279 (1991)
7. Pendharkar, P., Rodger, J.: Nonlinear programming and genetic search application for production scheduling in coal mines. Annals of Operations Research 95(1), 251–267 (2000)
8. Jarrah, A.I.Z., Bard, J.F., de Silva, A.H.: A heuristic for machine scheduling at general mail facilities. European Journal of Operational Research 63(2), 192–206 (1992)
9. Rochat, Y.: A genetic approach for solving a scheduling problem in a robotized analytical system. Journal of Heuristics 4, 245–261 (1998)
10. Croce, F.D., Tadei, R., Asioli, P.: Scheduling a round robin tennis tournamentunder courts and players availability constraints. Annals of Operations Research 92, 349–361 (1999)
11. Brucker, P., Hurink, J.: Solving a chemical batch scheduling problem by local search. Annals of Operations Research 96(1), 17–38 (2000)
12. Azizoglu, M., Kirca, O.: Scheduling jobs on unrelated parallel machines to minimize regular total cost functions. IIE Transactions 31(2), 153–159 (1999)
13. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. J. of Global Optimization 39(3), 459–471 (2007)

14. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8(1), 687–697 (2008)
15. Sundar, S., Singh, A.: A swarm intelligence approach to the quadratic minimum spanning tree problem. Information Sciences 180(17), 3182–3191 (2010)
16. Kashan, M.H., Nahavandi, N., Kashan, A.H.: DisABC: A new artificial bee colony algorithm for binary optimization. Applied Soft Computing 12(1), 342–352 (2012)
17. Akbari, R., Hedayatzadeh, R., Ziarati, K., Hassanizadeh, B.: A multi-objective artificial bee colony algorithm. Swarm and Evolutionary Computation 2, 39–52 (2012)
18. Jacobs, L., Brusco, M.: A local-search heuristic for large set-covering problems. Naval Research Logistics 42, 1129–1140 (1995)
19. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (2007)
20. McNaughton, R.: Scheduling with deadlines and loss functions. Management Science 6(1), 1–12 (1959)
21. Vredeveld, T., Hurkens, C.: Experimental comparison of approximation algorithms for scheduling unrelated parallel machines. Informs Journal on Computing 14(2), 175–189 (2002)
22. Lin, Y., Pfund, M., Fowler, J.: Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. Computers & Operations Research 38(6), 901–916 (2011)
23. Cheng, R., Gen, M., Tozawa, T.: Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. Computers & Industrial Engineering 29(1-4), 513–517 (1995)
24. Fanjul-Peyro, L., Ruiz, R.: Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207(1), 55–69 (2010)
25. Zaidi, M., Jarboui, B., Loukil, T., Kacem, I.: Hybrid meta-heuristics for uniform parallel machine to minimize total weighted completion time. In: Proc. of 8th International Conference of Modeling and Simulation, MOSIM 2010 (2010)
26. Garcia, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics 15, 617–644 (2008)