Ant Colony Optimization algorithms for the Permutation Flow Shop Problem with Weighted Tardiness

Hakim Boulahya
hboulahy@ulb.ac.be

Université Libre de Bruxelles

Abstract. In this article we present two implementations of Ant Colony Optimization metaheuristic algorithms for the Permutation Flow Shop Problem with Weighted Tardiness minimization objective (PFSP-WT). We implement Ant Colony System and $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and compare the results. The algorithms were extended using local search methods to improve local solutions, and provide better results. To reach better performance and tune parameters `irace`, an off-line automated parameters tuning tool, was used. We then conclude by analysis the results and propose the best found algorithm to provide a good solution to the PFSP-WT problem.

Keywords: PFSP · PFSP-WT · ACO · ACS · MMAS · Ant Colony Optimization.

1 Introduction

The PFSP-WT is the problem where jobs have to be schedule on a multi-machine setup. Let $\pi = (1, 2, ..., n)$ be a permutation of jobs that represents a schedule with n the number of jobs. The interpretation of the schedule is as follow: all machine execute the job in the same orders defined in π. A job can only be executed if the previous job on the same machine is completed and if the execution of the same job in the previous machine is also completed. The objective of this problem it to minimize the total weigheted tardiness of jobs $\sum_{i=1}^{n} w_i T_i$, where $T_i = max(C_i - d_i, 0)$ represents the tardiness of job i and w_i the weight of job i. C_i is the completion time of job i in the last machine and d_i the due date of job i. The completion time of job i in machine j is defined as follow:

$$C_{i,j} = max(C_{i-1,j}, C_{i,j-1}) + p_{ij} \tag{1}$$

where p_{ij} represents the processing time of job i on machine j. The completion time of the jobs in the first machine is the accumulation of processing times of all the jobs, where the completion time of the first job set to its processing time.

In the following sections, we described two Ant Colony Optimization (ACO) algorithms: Ant Colony System (ACS) and $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (MMAS)

to solve this problem. We then describe two local search methods to enhance the algorithms. Finally we define the parameters and methodology used to test the implementation and analyse the results.

2 Ant Colony Optimization algorithms for PFSP-WT

In this section we define the representation of the solution and the different variables and methods use to implement ACS and MMAS. Both algorithms are implemented as follow: while the termination condition is not met, a number of ants will build a schedule, and only the best-iteration ant solution is kept if it is better than the global-best solution. The global-best solution is initialized as described in 2.3.

A termination condition corresponds to a set of three sub-conditions: either the number of iterations is reach, number of steps is reached or a maximum set time has been reach. A step is the unit of measure that counts the number of schedules build by all the ants.

For the implementaion of ACS our work is inspired from the variables and methods studied by Den Besten et al. in [1]. They studied the PFSP-WT problem where only a single machine is taking into consideration. For MMAS, we extended the definition of the variables defined in [4] with the exhaustive description of ACO algorithms from the reference book of Dr. Dorigo [2], The objective function was also modified to minimize the total weigheted tardiness (where Stutzle minimize the makespan) as defined in [1].

2.1 Solution representation

For the PFSP-WT problem, a solution is a permutation of jobs $\pi = (1, 2, ..., n)$ that must be scheduled and executed. The objective is to minimize the sum weighted tardiness $\sum_{i=1}^{n} w_i T_i$ of a permutation of jobs.

2.2 Schedule construction

Remark When not specified τ_{ij} and η_{ij} correspond to $\tau_{ij}(t)$ and $\eta_{ij}(t)$, where $\tau_{ij}(t)$ is the pheromone trail of the job i in position j of the current schedule and η_{ij} the heuristic desirabity of the trail.

An ant will build a schedule as follow: at first, the solution is empty. Then for all the positions the ant select with probability q_0 the unscheduled job i for position j that maximize $(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}$, that is:

$$\text{argmax}_{unscheduled}(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta} \tag{2}$$

Otherwise, an ant explores with probabilibty $1 - q_0$ where the probability distribution to select an unscheduled job is:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{unscheduled}(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}} \tag{3}$$

2.3 Heuristic information

To initialize the solution and the heuristic information η_{ij}, the Earliest Due Date (EDD) heuristic was implemented [1]. This heuristic puts the initial solution in non-decreasing order of the due dates d_i and initialize the heuristic information as $\eta_{ij} = 1/d_i$, where η_{ij} is the heuristic information: when combined with τ_{ij} it is the desirabity of putting job i in position j [1].

2.4 Ant Colony System

In addition of the construction defined in the previous section, ACS defined two methods to update the pheromones: global and local.

Global Pheromone Trail Update

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\Delta\tau_{ij}(t) \tag{4}$$

where $\Delta\tau_{ij}(t) = 1/T^*$ with T^* the total weighted tardiness of the current solution and ρ the parameter for trail evaporation.

Local Pheromone Trail Update During schedule construction, each time an ant add a job i in position j the following update rule [1] is applied:

$$\tau_{ij} = (1-\xi)\cdot\tau_{ij} + \xi\cdot\tau_0 \tag{5}$$

Where $0 < \xi \leq 1$ and τ_0 is set as the initial value of the pheromone trails set to $\tau_0 = 1/(n\cdot T_{EDD})$ [1], with n is the number of jobs and T_{EDD} the total weighted tardiness of the solution generated by the heuristic EDD.

2.5 $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System

Schedule construction $\mathcal{MAX}$–$\mathcal{MIN}$ is set to always explore unscheduled jobs, that is $q_0 = 1$.

Global Pheromone Trail Update

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{6}$$

Pheromone Trails Limits τ_{max} is set set to $1/\rho T^*$ and upated at each iteration, each time a new best-so-far is found. $\tau_{min} = \tau_{max}/a$ where a is set to 5 [4]. The trails are initialized to τ_{max}.

Pheromone Trails Initialization and Reinitialization If the algorithm after a number of iterations stagnate, then the pheromone trails are reinitialized to τ_{max}.

2.6 Local search

To improve the performance and the results of the algorithms, local search was added. The method chosen to perform local search is by using *insertion-moves* [4]. That is for all existing permutation (i, j), where i is the position where the job has to be removed and j the position where the job has to be inserted with $i \neq j$. This local search is performs in $O(n^2)$.

Two mechanisms were implemented to perform local search: *best-ant* and *all-ant*. With *best-ant* only the best-iteration ant is allowed to perform local search, this computes in $O(n^2)$ per iteration. With *all-ant*, all ants perform local search at each iteration. *all-ant* mechanism computes in $O(mn^2)$, where m is the number of ants.

3 Final Results

3.1 Methodology

This section presents the results of the executions of the algorithms. We show and compare the results of six algorithms: ACS, MMAS, ACS-LS-BEST, ACS-LS-ALL, MMAS-LS-BEST and MMAS-LS-ALL, where ACS and MMAS are the ACO algorithms without local search and the others are the respective algorithms with local search enabled using the method described in 2.6. For example, MMAS-LS-BEST is MMAS with local search enabled where only the best-iteration ant performs local search.

Setup To generate data, we executed ten runs of all six algorithms for each of the twenty instances [5] (1200 executions in total), with a budget of 30 seconds per execution. The experiments were run using an Intel(R) Xeon(R) E5-2620 CPU with 32 Cores and 64GB of RAM on CentOS 7.

3.2 Parameter settings

For setting the parameters, we used `irace` [3]. We tuned the parameters listed in Table 1 with a maximum number of experiments set to 200. Note that q_0 and ξ are not tuned for MMAS as there is only exploration and no local update in this algorithm.

Parameter	Range	ACS	MMAS	ACS-LS-BEST	ACS-LS-ALL	MMAS-LS-BEST	MMAS-LS-ALL
ants	(5, 100)	41	28	5	93	34	87
α	(0.00, 5.00)	2.56	3.88	2.83	3.69	4.41	0.34
β	(0.00, 10.00)	7.78	6.69	3.28	5.08	9.48	6.76
ρ	(0.01, 1.00)	0.32	0.53	0.61	0.05	0.41	0.7
ξ	(0.01, 1.00)	0.46	0.53	0.38	0.77	-	-
q_0	(0.0, 1.0)	0.07	-	0.04	0.24	-	-

Table 1: Parameters tuned and best configurations returned by `irace` for each algorithm

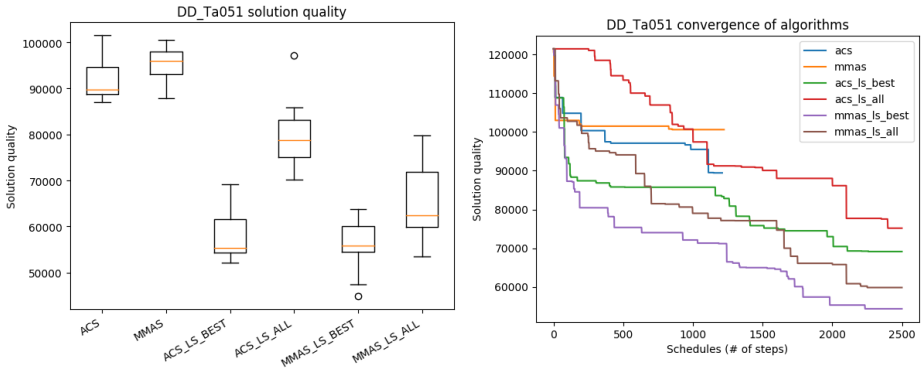
3.3 Analysis

In Table 2, the best total weighted tardiness for each instance returned by each algorithms is shown. As one can see algorithms with local search enabled always perform better. With those results, one would suggests that the preferred algorithm might depend on the instance itself. MMAS seems to performs better on instances with more jobs, where ACS is most of time better with instances with less jobs. Regarding the comparison between the different local search method described in 2.6, *best-ant* seems to always be a better choice compared to *all-ant*. One could say that the fact that only the best ant is allowed to search locally, enables the algorithm to look for better solution faster.

Figure 2 show the solutions quality and the convergence of the algorithms for an instance with 50 jobs where Figure 4 show the corresping plots for an instance with 100 jobs. In both cases, the algorithms with *best-ant* local search converge faster to better solutions, and the quality of the solution is often better.

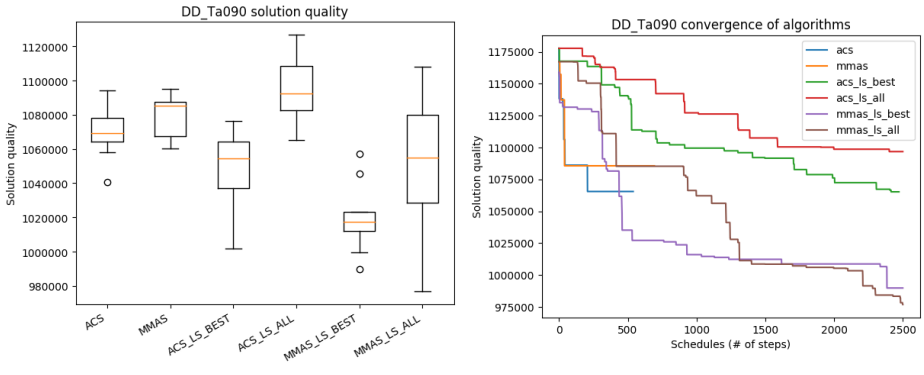
Instance	ACS	MMAS	ACS_LS_BEST	ACS_LS_ALL	MMAS_LS_BEST	MMAS_LS_ALL
DD_Ta051	87021	87907	52055	70168	**44847**	53427
DD_Ta053	108723	118469	**61368**	77178	69128	74998
DD_Ta054	102680	107279	**59899**	86431	60958	67105
DD_Ta055	73943	78490	**32623**	61840	41799	41248
DD_Ta052	98254	104456	60866	79199	**54697**	65945
DD_Ta056	70778	70824	**35330**	65928	40988	51824
DD_Ta057	90580	98918	**47534**	78845	48303	62596
DD_Ta058	85556	95524	**57207**	86977	58272	72187
DD_Ta059	81253	83349	52033	68901	**49384**	55720
DD_Ta060	146394	169210	117216	127819	117058	**105088**
DD_Ta081	954686	958138	900036	955650	879685	**881917**
DD_Ta082	859666	844413	**766361**	842497	802836	805140
DD_Ta083	1068515	1059775	991420	1041605	**980420**	1027187
DD_Ta084	1138753	1171783	**1089303**	1139172	1103457	1139404
DD_Ta085	1062634	1082410	1005153	1077911	1000383	**975061**
DD_Ta086	1060223	1092469	1042539	1035553	**993250**	1039797
DD_Ta087	895045	926469	873978	915212	**873015**	875673
DD_Ta088	1015562	1043175	991322	1008984	**977752**	978800
DD_Ta089	1099690	1137683	1088799	1120113	1100444	**1085154**
DD_Ta090	1040605	1060481	1001944	1065127	989727	**976735**

Table 2: Best total weighted tardiness for each instances returned by the algorithms



Solution quality Convergence of algorithms

Fig. 2: Boxplot and convergence for instance `Ta051`



Solution quality Convergence of algorithms

Fig. 4: Boxplot and convergence for instance `Ta090`

4 Conclusion

In this paper we describe the Permutation Flow Shop Problem with Weighted Tardiness minimization objective. We provided the implementation of two Ant Colony Optimization algorithms, ACS and MMAS, to solve the problem. Following the main implementation, we described two rather simple local search methods that significantly improve the solutions generated by the basic algorithms. One important result that the results highlighted is the importance of local search in those ACO algorithms. In the future, one might be interested in

studying the impact of the parameters (only tuned with `irace` in this context) on the algorithms and the results for this problem.

References

1. den Besten, M., Stützle, T., Dorigo, M.: Ant Colony Optimization for the Total Weighted Tardiness Problem. In: Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J., Schwefel, H.P. (eds.) Parallel Problem Solving from Nature PPSN VI. pp. 611–620. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2000)
2. Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press (2004). https://doi.org/10.7551/mitpress/1290.001.0001, https://direct.mit.edu/books/book/2313/ant-colony-optimization
3. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011), http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf
4. Stützle, T.: An Ant Approach to the Flow Shop Problem. In: In Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98. pp. 1560–1564. Verlag (1997)
5. Taillard, E.: Scheduling instances for flow shop sequencing (2019), http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html