Project Report on Ant Colony Optimization applied to the Permutation Flow Shop Problem with Weighted Tardiness

Théo Verhelst

Université Libre de Bruxelles

Abstract. This report describes our work on the final assignment of the *INFO-F414 Swarm Intelligence* course. This assignment concerns the evaluation of different Ant Colony Optimization (ACO) algorithms applied to the *Permutation Flow Shop Problem* with *Weighted Tardiness*. We implemented two variants of the Ant System algorithm, namely the *ranked-based* variant and the *Max-Min* variant [5]. We also implemented the *Greedy Iterative Random Local Search* [1]. The algorithm parameters have been optimized using a grid search procedure, and the performance of the three algorithms are compared. Also, the addition of local search to the rank-based ant system has been evaluated. We use 20 instances of the permutation flow shop problems given in [6]. The iterated search approach gave the best results by a clear margin. The rank-based ant system outperforms the max-min variant on large problem instance. The addition of local search improved significantly the solution given by the rank-based ant system, although at the expense of convergence.

1 Introduction

This report presents the work we conducted for the final assignment of the *INFO-F414 Swarm Intelligence* course. It concerns the problem of scheduling a set of tasks on a set of machines, where each task has to be executed on all machines in order, but the relative order of the tasks can be modified. This type of problem occurs in many real-world domains, such as scheduling airplane landings, jobs in a factory, or task scheduling in an operating system.

This report is organized as follow: section 2 presents the permutation flow shop problem with weighted tardiness in a formal manner. Section 3 describes in details the three algorithms we used to solve the problem, the local search approach, as well as the representation chosen for the different data structures. Section 4 presents the experiments we conducted for optimizing and assessing the performance of the three algorithms and the local search. The results are shown and discussed in section 5. Finally, a conclusion is given in section 6.

2 Problem description

The permutation flow shop problem consists in scheduling a set of of n jobs. Each job needs to be executed on m different machines in order. Each job

$i \in N = \{1, \ldots, n\}$ needs a non-negative execution time $p_{i,j}$ on machine $j \in M = \{1, \ldots, m\}$. The objective is to find a permutation of the set $\{1, \ldots, n\}$ representing the order in which the jobs are scheduled, as to minimize some objective function. The completion time $C_{i,j}$ is the total time needed to complete the execution of job i on machine j, and is defined as

$$C_{i,j} = \max(C_{i-1,j}, C_{i,j-1}) + p_{i,j} \quad \forall i \in N, j \in M. \tag{1}$$

That is, the jobs starts its execution on a machine either when it finishes its execution on the previous machine, or when the previous jobs is done with the machine, whichever happens first. The first job starts on the first machine without delay, which is formally defined as $C_{i,0} = 0$ and $C_{0,j} = 0$, for all $i \in N$ and $j \in M$. Each jobs i has an associated non-negative due date d_i, which indicates the time at which the job should have finished its execution on the last machine. This allows to define the *tardiness* $T_i = \max(0, C_{i,m} - d_i)$. A different weight w_i is given to each job, and the *total weighted tardiness* is thus defined as $\sum_{i=1}^{n} w_i T_i$. Our objective in this project is to find a permutation π of the n jobs that minimizes the total weighted tardiness $f(\pi)$. This definition of the problem implies that

- all jobs are ready to be executed at instant 0;
- all machines are always available;
- a job can execute on one machine at a time;
- the job order is the same for all machines;
- no preemption is allowed.

3 Algorithms description

We implemented three different search algorithms. Two are variants of the Ant System, and one is an iterated search. All of these three algorithms start by using the NEH-edd heuristic [2], which is itself an adaptation for tardiness of the NEH heuristic [4]. We in turn modified it as to take into account the weighted total tardiness instead of the total tardiness as evaluation measure. In the case of the two ant systems, this initial solution is used to initialize the trail, whereas it used as the starting point of the iterated search approach. The NEH-edd heuristic consists in sorting all jobs by increasing due date, and iteratively inserting jobs at the best position among the scheduled jobs. At iteration $i \in N$, the job i in the sorted list of jobs is considered for insertion in the partial solution between all positions j and $j + 1$, for $j \in \{1, \ldots, i\}$. The insertion that yields the least weighted total tardiness is chosen, the job is inserted, and the next iteration begins. This stops when all jobs are scheduled. This heuristic requires the ability to evaluate solutions that do not schedule the entire set of n jobs.

3.1 Rank-based Ant System

The first variant of Ant System is the Rank-based Ant System (RB-AS), inspired from the course material. The solutions are constructed by each ant based solely

on the pheromone trail, in an iterative manner. For each position $j \in N$, a job is chosen amongst the set yet unscheduled jobs N', according to a discrete probability density whose terms are proportional to $\tau_{i,j}$ $\forall i \in N'$. At the end of each iteration, a number ω of ants with the best iteration solutions are selected for updating the trail. The trail at iteration t is updated as

$$\tau_{i,j}(t) = \rho\tau_{i,j}(t-1) + \sum_{r=1}^{\omega}(\omega - r - 1)\Delta\tau_{i,j}^{(r)} \tag{2}$$

where $\tau_{i,j}(t)$ denotes the willingness of an ant to place job j at position j in the solution, ρ is the trail persistence parameter, and

$$\Delta\tau_{i,j}^{(r)} = \begin{cases} 1/f(\pi_r) & \text{if job } i \text{ is } j\text{th position in the solution of ant } r \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The ω ants are of course ordered by the weighted tardiness of their solutions, as to give a greater weight to more efficient solutions.

3.2 Max-Min Ant System

The second variant of the Ant System we implemented is the Max-Min Ant System (MM-AS), and is inspired by [5]. The pheromone trail is initialazed in the same way as the Rank-based approach, using a initial solution with the NED-edd heuristic. The trail update uses only the best solution found in the current iteration (thus equivalent to formula 2 with $\omega = 1$), and is then constrained between two limit values τ_{min} and τ_{max}. τ_{max} is defined as

$$\tau_{max} = \frac{1}{(1-\rho)f(\pi_{best})} \tag{4}$$

where $f(\pi_{best})$ is the total weighted tardiness of the best solution found so far. τ_{min} is defined as $\tau_{min} = \tau_{max}/5$. Also, whenever the solution do not improve for a certain number of iterations, the trail is initialized again to avoid getting stuck in a local minima.

3.3 Iterated Greedy Random Local Search

The last algorithm is the Iterated Greedy Random Local Search (IG-RLS) presented in [1]. This algorithm starts with the NEH-edd heuristic, and apply a local search to the solution. Then, at each iteration, the a new solution is found using the *destruction-construction* procedure, and a local search on the result of this procedure. If the resulting solution π' is better than the previous solution π, it is kept for the next iteration. If not, it is nonetheless kept with a probability $\exp((f(\pi) - f(\pi'))/Temp)$, where $Temp$ is a temperature variable. This reduces premature convergence by increasing exploration. The temperature is defined in [1] as

$$Temp = \frac{1}{10n} \sum_{i=1}^{n} (LB_{Cmax} - d_i) \tag{5}$$

Where LB_{Cmax} is the lower bound on the makespan presented in [6]:

$$LB_{Cmax} = \max \left\{ \max_{i \in N} \left(\sum_{j=1}^{n} p_{i,j} \right), \max_{j \in N} \left(\sum_{i=1}^{n} p_{i,j} \right) \right\} \tag{6}$$

We designed a second version of this formula, in order to take into account the different weights of the jobs. The formula 5 thus becomes

$$Temp = \frac{1}{10n} \sum_{i=1}^{n} w_i (LB_{Cmax} - d_i) \tag{7}$$

This alternate formula is used depending on the value of a flag, and we consider this flag as a parameter subject to optimization.

The destruction-construction procedure consists in removing a set of d randomly chosen jobs from the solution, and inserting each of them iteratively in the position that minimizes the evaluation function. The local search chooses randomly two positions pos_1 and pos_2 in the solution, and either moves the job at pos_1 to pos_2, or swaps the jobs at pos_1 and pos_2. The choice between these two operations is made randomly with equal probability. This operation is repeated until no improvement on the solution is noticed for n iterations in a row.

3.4 Implementation

In all of these algorithms, a significant speedup mechanism presented in [3] is used. It is based on the observation that the tardiness of a job depends solely on the jobs before it in the solution. So, when a new solution is evaluated, the computation of the evaluation function will be identical to the one made for the previous solution, up to the first position where the two solutions differ. By keeping in memory the matrix of completion times $[C_{i,j}]_{i,j \in N}$ and updating it only for jobs at or after the first different position in the two solutions, the computation time can be significantly reduced.

The representation we use for the different data structures is rather straightforward. A solution is stored as an array of integers $[J_i]_{i \in N}$ indicating that the job J_i is scheduled at position i. The trail is a two-dimensional array $[\tau_{i,j}]_{i,j \in N}$ of size $n \times n$ indicating the willingness of an ant to schedule job i at position j in a solution. The completion time matrix $[C_{i,j}]_{i \in N, j \in M}$ for a given solution is a two-dimensional array of size $n \times m$ indicating the time required for job i to complete its execution up to the machine j.

4 Experiment

The experimental setup is composed of three phases: parameters optimization, algorithms performance assessment, and local search performance assessment. We detail in turn each of these phases in the following subsections.

4.1 Parameters optimization

The three algorithms each have a different set of parameters that can be tuned to maximize their performance. These are:

RB-AS

- number of ants $K \in \mathbb{N}$
- number of top ants selected in ranking $\omega \in N$
- trail persistence $\rho \in [0, 1]$

MM-AS

- number of ants $K \in \mathbb{N}$
- exploitation probability $p_0 \in [0, 1]$
- ratio between trail extrema $a \in \mathbb{R}^+$
- number of iteration before trail reset $L \in \mathbb{N}$
- trail persistence $\rho \in [0, 1]$

IG-RLS

- number of constructed-destructed jobs $d \in N$
- weighting of the temperature formula (true or false)

Since we did not have enough computation time to evaluate a significant portion of the search space, we restricted the search to a set of 6 different configurations for each algorithm. The configurations have been chosen as to explore more on parameters that we believe to have a greater impact on the solution quality. Also, when multiple parameters influence the exploration/exploitation trade-off, the value of these parameters are modified as to influence the trade-off in the same direction. The 6 configurations are shown in section 5, in figures 1 to 3. For MM-AS, in all configuration we set $a = 5$ and $L = 50$. The performance of each configuration is evaluated by running each algorithm once on each of the 20 problem instances. The configuration being most often the best is retained.

4.2 Algorithms performance

The best parameter configurations is selected according to the protocol presented above, and each of the three algorithms is run 10 times on each of the 20 problem instances. Wilcoxon rank-sum tests are then conducted to determine the best algorithm for each problem instance. Bonferroni correction is applied by

multiplying the p-values by 3, since the p-values given for the three algorithms comparisons are dependent, for given problem instance. The p-values between two different instances are however independent. Also, the convergence of each three algorithms on the problem instance `DD_Ta060` is evaluated by comparing the current solution quality against execution time and the number of calls to the evaluation function. This instance is chosen because its size is a compromise between small and large instances. Note that these results are evaluated on the same data set as for the parameter optimization phase. We have therefore no guarantee of generalization of these results, as the best parameters are biased towards the specific problem instances we have.

4.3 Local search assessment

The local search used for in IG-RLS algorithm is added to the best performing of the two Ant System approaches, and the resulting algorithm is run 10 times on each of the 20 problem instances. The same statistical test as presented in the previous subsection is conducted to evaluate the statistical significance of the difference in performance with and without local search.

5 Results

Results for the parameter optimization are given in figures 1 to 3. A box-plot representing the distribution of the total weighted tardiness of the solution for all instances is represented. We also added a dot for each value, with some jitter on the vertical axis to better distinguish each dot. It can be seen that all runs of the RB-AS approach give the same numerical result, independently of the chosen parameters. For the MM-AS, the number of ants did not produce a significant difference, but the two parameters p_0 and ρ influencing the exploitation-exploration trade-off where better adjusted when favoring exploitation. The results for IG-RLS are not as clear-cut as for MM-AS, but choosing $d = 2$ and no job weighting in the temperature formula gave the best results. A small value of d favors exploitation, since this allows to evaluate new solutions closer to the current one in the local search phase. The fact that the weighted temperature formula does not give better results can be explained by the fact that the weights have not been normalized, the weighted temperature has thus a larger value, simply increasing the exploration rate.

The convergence of the three algorithms on the instance `DD_Ta060` is shown in figure 4. One can observe that the Rank-based Ant System fails to improve its solution altogether, and is stuck with the initial solution provided by the NEH-edd heuristic as the best solution. The MaxMin Ant System do not converge either, and even worson over time. The spikes are caused by the reset of the trail

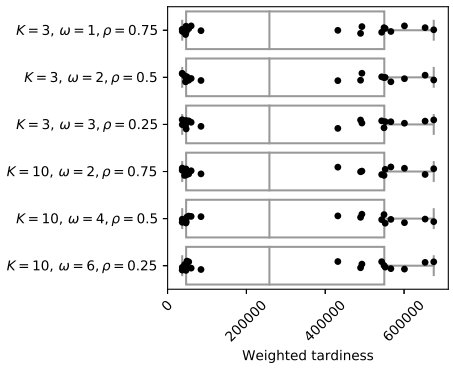

Fig. 1. Evaluation of different parameters configurations for the Rank-based Ant System algorithm. All configurations gave identical solutions, for all problem instances.

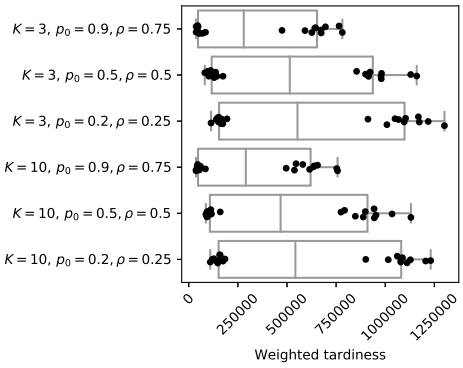

Fig. 2. Evaluation of different parameters configurations for the Max-Min Ant System algorithm. The best performing configuration is using 3 or 10 ants, with an exploitation probability of $p_0 = 0.9$, and a trail persistence factor $\rho = 0.75$.

every 50 iterations due to the absence of solution improvement. On the other hand, the iterated search approach converges slowly to a better solution.

The results of the performance assessment are given in figures 5 and 6. The numerical results are also summarized in table 2. For the graphics, we separated the results into two plots, since the magnitude of the numerical results is different between the first 10 instances and the last 10 due to the different number of jobs and machines. The IG-RLS performs consistently better than the two AS approaches. RB-AS is better than MM-AS on large problem instances, but show equivalent performances on small problem instances.

Finally, the results of the use of local search for the Rank-based Ant System are presented in figures 8 to 10. As it can be seen on figures 8 and 8, the local search alternative clearly outperforms its standard counterpart. The Wilcoxon rank sum tests all give a p-value less than 0.01, we therefore do not report the p-values. It is worth mentioning however that the algorithm do not converge to the optimal solution, as shown in figure 10, and actually ends up in solution worse than without the local search. The final solution is better because it is found in the first iterations, by searching locally around the initial guess provided by the NEH-edd heuristic.

The RB-AS is more sensitive to parameter choices with local search than without, as it can be seen by comparing figures 1 and 9. The best performing configuration for the local search variant consists in using $K = 10$ ants, selecting the best $\omega = 2$ for updating the trail, and setting the persistence parameter ρ to 0.75.

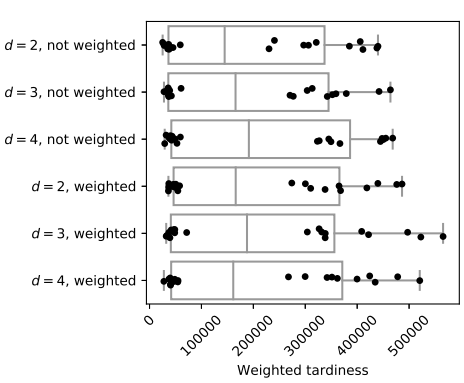

Fig. 3. Evaluation of different parameters configurations for the IG-RLS algorithm. The best performing configuration is to destruct-construct $d = 2$ jobs in the local search, with no weighting in the temperature calculation.

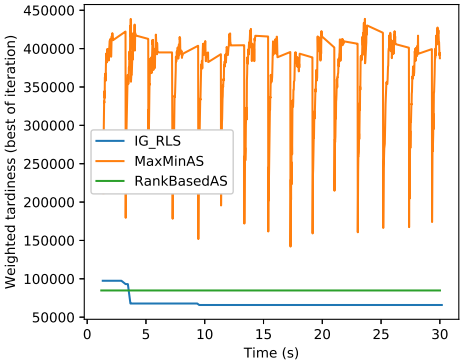

Fig. 4. Convergence of the three algorithms on the instance `DD_Ta060`. RB-AS fails to improve upon the first solution found with NEH-edd, the solutions MM-AS degenerate until the trail is reset every 50 iterations, while IG-RLS converges slowly.

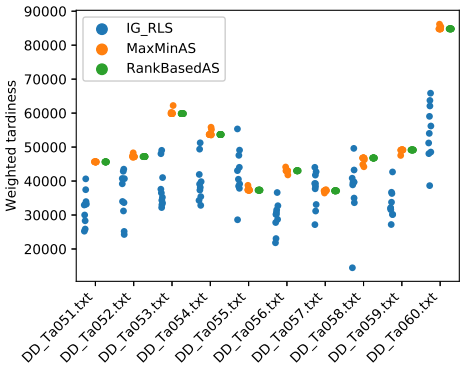

Fig. 5. Comparison of all three algorithms with best parameters. Only small instances are shown. The two ant system approaches show similar performances and are outperformed by the iterated search approach.

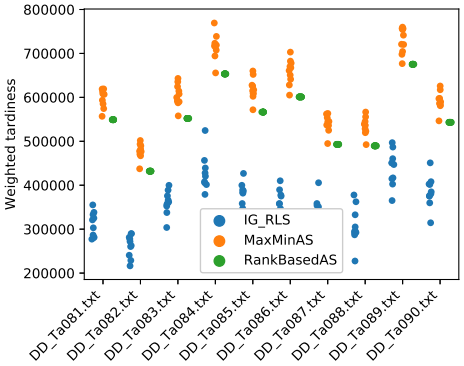

Fig. 6. Comparison of all three algorithms with best parameters. Only large instances are shown. The Rank-based approach performs as well as the best run of the Max-Min ant system. The iterated search clearly outperforms the two ant systems.

Instance	RB-AS vs MM-AS	RB-AS vs IG-RLS	MM-AS vs IG-RLS
DD_Ta051	0.95	< 0.01	< 0.01
DD_Ta052	1	< 0.01	< 0.01
DD_Ta053	0.20	< 0.01	< 0.01
DD_Ta054	0.84	< 0.01	< 0.01
DD_Ta055	0.09	< 0.01	0.01
DD_Ta056	1	< 0.01	< 0.01
DD_Ta057	1	0.32	0.36
DD_Ta058	0.20	< 0.01	< 0.01
DD_Ta059	0.20	< 0.01	< 0.01
DD_Ta060	1	< 0.01	< 0.01
DD_Ta081	< 0.01	< 0.01	< 0.01
DD_Ta082	< 0.01	< 0.01	< 0.01
DD_Ta083	< 0.01	< 0.01	< 0.01
DD_Ta084	< 0.01	< 0.01	< 0.01
DD_Ta085	< 0.01	< 0.01	< 0.01
DD_Ta086	< 0.01	< 0.01	< 0.01
DD_Ta087	< 0.01	< 0.01	< 0.01
DD_Ta088	< 0.01	< 0.01	< 0.01
DD_Ta089	< 0.01	< 0.01	< 0.01
DD_Ta090	< 0.01	< 0.01	< 0.01

Table 1. Double-sided p-values for the Wilcoxon rank-sum test on the results of each algorithms using Bonferroni correction. The IG-RLS outperforms clearly the two other algorithms, and RB-AS is better than MM-AS only on large problem instances.

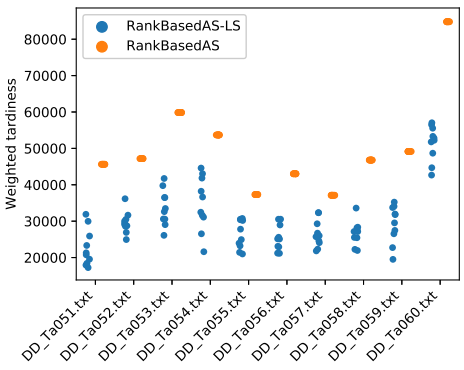

Fig. 7.

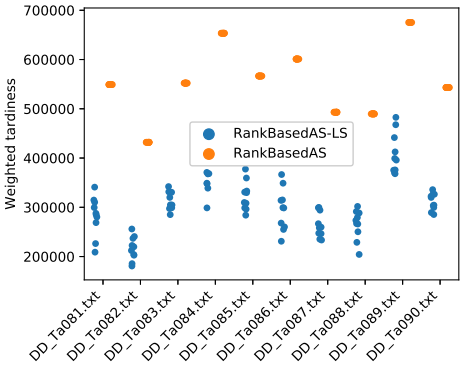

Fig. 8.

	RB-AS				MM-AS				IG-RLS			
Instance	B	W	M	SD	B	W	M	SD	B	W	M	SD
DD_Ta051	45634	45634	45634	0	45634	45705	45641	22	25237	40644	32076	4864
DD_Ta052	47185	47185	47185	0	46973	48311	47293	375	24267	43496	35527	7020
DD_Ta053	59848	59848	59848	0	59848	62234	60144	744	32190	49043	38049	6093
DD_Ta054	53690	53690	53690	0	53540	55845	54044	788	32839	51259	39951	6100
DD_Ta055	37317	37317	37317	0	37317	38744	37527	456	28601	55341	42376	7322
DD_Ta056	43023	43023	43023	0	41799	44134	43012	551	21826	36621	29344	4364
DD_Ta057	37118	37118	37118	0	36469	37465	37102	249	27168	44039	37461	5389
DD_Ta058	46788	46788	46788	0	44206	46788	46291	947	14469	49618	37462	9181
DD_Ta059	49154	49154	49154	0	47539	49154	48984	508	27201	42682	33265	4373
DD_Ta060	84800	84800	84800	0	84788	86175	85003	462	38649	65859	54743	8394
DD_Ta081	549124	549124	549124	0	556878	619199	598403	20630	277249	355178	310094	28114
DD_Ta082	431968	431968	431968	0	437385	501821	477416	17598	216440	289807	261920	25592
DD_Ta083	552112	552112	552112	0	557897	643112	605201	25396	303774	399811	360203	26728
DD_Ta084	653370	653370	653370	0	655672	769261	716022	29076	331515	524537	419392	50438
DD_Ta085	566521	566521	566521	0	571788	660010	618935	25028	322640	426856	373880	31053
DD_Ta086	600992	600992	600992	0	605099	702999	658722	28452	283597	410129	357167	35063
DD_Ta087	493053	493053	493053	0	494855	563338	539841	19915	273615	405637	331052	39023
DD_Ta088	489618	489618	489618	0	492433	566568	534348	20492	227488	377792	306674	42421
DD_Ta089	675281	675281	675281	0	676791	759833	728938	29275	365077	496922	439191	39891
DD_Ta090	543186	543186	543186	0	546510	625876	590965	21600	314495	450943	385390	35392

Table 2. Summary of the results in numerical form, with best (B), worst (W), mean (M), and standard deviation (SD) of the 10 runs of each algorithms on each of the 20 problem instances.

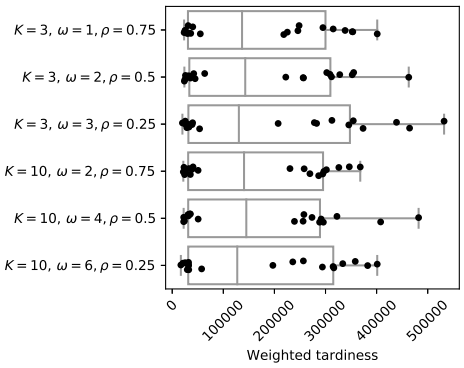

Fig. 9. Performance of the 6 parameter configurations for the Rank-based Ant System with local search. The parameters have a bigger impact than when not using local search, and the best configuration is the fourth one.

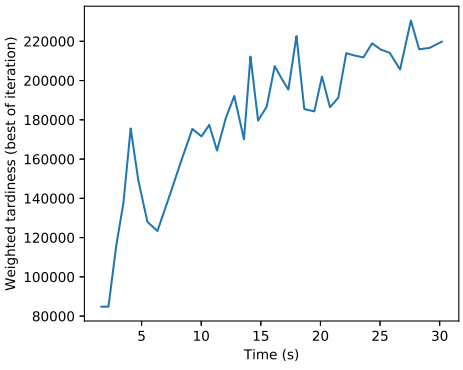

Fig. 10. Convergence plot of the Rank-based Ant System with local search. The algorithm does not converge anymore, but better solution are however found than without local search because of the better exploration.

6 Conclusion

We implemented three different algorithms for solving the permutation flow shop problem with weighted tardiness, optimized their parameters, and evaluated their performances. The parameters had no impact on the solution for the rank-based ant system, but for the other two algorithms a parameter set favoring exploitation over exploration seems to perform best. The iterated search approach gave the best results by a clear margin on the 20 problem instances, except for the instance `DD_Ta057`. The rank-based ant system outperformed the max-min variant on large problem instance. The addition of local search improved significantly the solution given by the rank-based ant system, although at the expense of convergence.

Note to the reader

I apologize for the lengthy report, which should be 6 pages long maximum. I did my best to be concise and at the same time explain everything I did and report all of my results. This report length is partially due to the implementation of a third algorithm, which is not an ant system. Also, the code of the project does not follow the style of the course exercises (which is compiled C++ program for the algorithm and bash scripts for the experiments). Instead, I did everything in Python, which is efficient for both computations and scripting. This enables a more consistent coding style for the project.

References

1. Karabulut, K.: A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. Computers & Industrial Engineering **98**, 300–307 (2016)
2. Kim, Y.D.: A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. Computers & Operations Research **20**(4), 391–401 (1993)
3. Li, X., Wang, Q., Wu, C.: Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega **37**(1), 155–164 (2009)
4. Nawaz, M., Enscore Jr, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983)
5. Stützle, T., et al.: An ant approach to the flow shop problem. In: Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing. vol. 3, pp. 1560–1564 (1998)
6. Taillard, E.: Benchmarks for basic scheduling problems. european journal of operational research **64**(2), 278–285 (1993)