Ant Colony Optimization for DSM's Flexible Job Shop Problem

Tim Numan
Supervisor(s): Kim van den Houten, Dr. Mathijs de Weerdt
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 19, 2022

Abstract

This papers examines an ant colony optimization approach for solving a specific variant of the Flexible Job Shop Problem faced by the Dutch chemistry company DSM. Jobs consisting of operations on a specific enzyme need to be scheduled as efficiently as possible on groups of available machines. The most interesting requirement upon the general FJSP are the sequence-dependent cleaning times a machine needs when it processes two different enzyme types consecutively. This can all be intuitively represented using a weighted disjunctive graph, which ACO uses as its input. The algorithm consists of a number of epochs for which multiple ants create a feasible schedule one operation at the time. Scheduling choices are made using pheromones and an heuristic visibility function based on earliest starting times. Pheromone amounts are updated using both a negative local updating rule and a positive global update for the best ant per epoch. Two hyperparameters, the initial pheromone amount τ_0 and the cutting exploration parameter q_0 are experimentally evaluated. Varying τ_0 does not consistently impact the solution quality nor runtime, while the higher the value of q_0, the better both measures. Performance evaluation of ACO is done using a provided MILP solver as baseline and the makespan as objective function for a range of different time limits. For the tested instances, ACO shows to significantly outperform MILP, finding good solutions exceptionally fast. Based on this, ACO is an efficient method to solve the production scheduling problem of DSM.

1 Introduction

Production scheduling is one of the most important matters in manufacturing systems [1]. A concrete example is the challenge the Dutch chemistry company DSM faces. They repeatedly have to solve a complex scheduling question for their enzyme production line. In their plants batches of enzymes are produced according to a product-specific recipe that consists of different unit operations, which need to be scheduled as efficiently as possible on the available machines. The complexity of the problem raises by the fact that specialized cleaning is required if one machine successively processes two different products. This makes it a special form of the flexible job shop problem (FJSP).

This example from DSM does not stand on its own. There are many significant real-world applications of the FJSP [1], but being NP-hard [2], it is very challenging to solve. Therefore the problem has been extensively studied in literature, the first time being in 1990 [3]. An established graphical method for the JSP was adapted so it could be used for it's flexible variant, but this approach turned out not to be very efficient for three or more jobs. A better way to solve the FJSP was found in the form of (mixed) integer (linear) programming (MILP) [4]. This mathematical technique encompasses most of the exact algorithms that are presented in literature. While an exact approach like MILP guarantees the optimal solution, as a consequence of the problem's complexity, the runtime grows quickly. Therefore this method is often not viable for real-world instances with larger problem sizes.

Hence, much research has been done on heuristic approaches and the use of metaheuristics for the FJSP [1, 5]. An example for this last category is ant colony optimization (ACO), a population-based probabilistic algorithm inspired by the foraging strategies of ants. Initially developed for the travelling salesman problem [6], it was first applied to the JSP in 1994 [7]. Rossi showed it could also be extended to the FJSP [8, 9]. In his flexible job-shop environment there were sequence-dependent setup times, which can be compared to the extra cleaning requirement in DSM's problem. Subsequently, several improvements to the ACO strategy have been suggested [10, 11, 12], showing good results for many different problems [5].

Most of the previous work has been focused on pure research, developing new/improving existing algorithms which are tested on standard benchmarks, instead of applying it to a concrete, practical problem [5]. Especially the personal needs of real-world production environments, which are becoming more and more complex, should be investigated [1]. This research focuses on such a specific problem of a manufacturer: DSM. ACO, chosen because of the good previous results with sequence-dependent setup times, is applied to DSM's specific variant of the FJSP, followed by analyzing the performance.

Formally, this expressed in the following research question: **Can ant colony optimization be used to efficiently solve the production scheduling problem of DSM?** In order to determine whether the ACO approach is efficient, the more conventional MILP method is used to establish a baseline. To be able to tackle the research in a step-by-step manner, the main research question can be broken down into the following subquestions:

1. What is the performance of solvers for the mixed integer linear programming formulation of this problem?
2. How can ant colony optimization best be applied to this problem?
3. What is the performance of the ant colony optimization method implemented for this problem?
4. How does ACO compare to MILP for tackling this problem?

For the above questions, minimizing the makespan is the objective function, which is also the case for most existing research on the FJSP [5]. However, for DSM it is also important to meet certain due dates for customers and minimize the idle time of the machines. This leads to the following extra subquestion:

5. How can be dealt with conflicting objectives in the optimization problem?

The rest of the paper is organized in the following manner. A formal description of DSM's variant of the FJSP is given in Section 2. Section 3 presents the ACO algorithm implemented to solve this problem, followed by a description of the experimental setup and results in Section 4. Section 5 reflects on the ethical aspects of the research. A discussion of the results can be read in Section 6 and finally Section 7 provides the conclusions of this study.

2 Formal description of DSM's Flexible Job Shop Problem

To be able to give a more formal description of the DSM problem introduced in the previous section, it is helpful to first define its generalizations. The general job scheduling problem [13] can be defined as follows: there are **N** jobs having different processing times, and **M** machines having different processing power. The jobs need to be scheduled on the machines in such a way that the total length of the schedule, also called makespan, is minimized. A variant of this is the job **shop** problem (JSP), where each job consists of a set of operations that need to be processed in order and each on a specific machine. Finally, the **flexible** job shop problem (FJSP) is a relaxation hereof, allowing each operation to be processed on a set of identical machines.

DSM's variant of the FJSP has an extra complication: the required cleaning times when two operations of jobs of different enzyme type are consecutively processed on the same machine. A generalization of this which can be found in literature [8, 9] is the FJSP with **sequence-dependent setup times**. However, DSM's problem becomes even more difficult, because the setup times can differ per machine within the sets of identical machines. Hence the machines in each set are actually not completely identical, as is generally the case.

There are also some simplifications compared to the general FJSP for DSM's case. There are namely only three different operation types: preparation, filtering and reception. Also, there is one set of available machines for each of these unit operations, and these sets are disjoint, meaning no machine can support multiple operation types. The machines within each set are identical in the sense that can all perform the same operation type, with the same processing time. The processing times for each unit operation differ per enzyme type, which is associated with each job, but thus not per machine. However, as mentioned in the above paragraph, the setup times do differ per machine within each set. The two other variables that determine how long cleaning takes are the enzyme types of the jobs including the two successive operations that are carried out on the machine, where the relative order of the two operations is important. Therefore, if **E** is the total number of different enzyme types over all jobs, this results in **M ExE** change-over matrices.

The main objective for DSM, as for most FJSPs referenced in literature [5], is the minimization of the makespan. However, each job also has a due date and meeting these can be enforced by minimizing the tardiness of the schedule. Finally, DSM also prefers to keep the idle times of the machines as low as possible. Optimizing these three objectives at the same time results in a **multi-objective** FJSP.

3 Ant Colony Optimization algorithm for solving the DSM problem

Before diving into the specific ACO algorithm implemented for this research, some general background explanation on the method is given. As mentioned in the introduction, ACO is a metaheuristic approach. This means it is a higher-level algorithmic framework that can be applied to many different problems to find feasible solutions. The fact that it does not necessarily return the optimal solution makes it heuristic.

Of course the goal is to find as good as possible solutions, and ACO does this by mimicing the foraging behaviour of ants. The algorithm is population-based, creating multiple artifical ants for different generations. While exploring the search space, these agents lay down pheromones, to direct ants in later generations to better solutions. Being probabilistic, ants will not always exploit the path with the most pheromones, but still explore other routes too.

ACO was initially invented [6] for the travelling salesman problem (TSP), for which a set of cities needs to be visited using the shortest route possible. An instance for the TSP is intuitively represented as a graph, with the nodes being the cities, and weighted edges representing the distances between every pair of these. Using ACO for this problem would mean that each individual ant walks a path through the graph, visiting each node exactly once. The better the route of the ant turns out to be, the more pheromones will put on the corresponding edges. The idea is that later generations will find better and better paths, and in the end the best route found is chosen as the solution.

To be able to apply ACO to DSM's problem, instances of this problem also need to be modeled as a graph. This is possible using the disjunctive graph representation [8, 9], which can be seen in Figure 1.

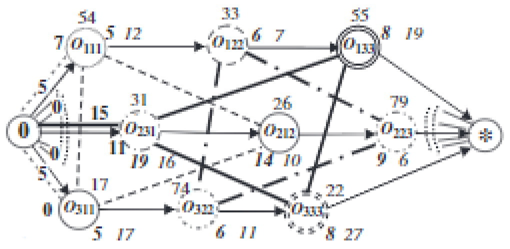


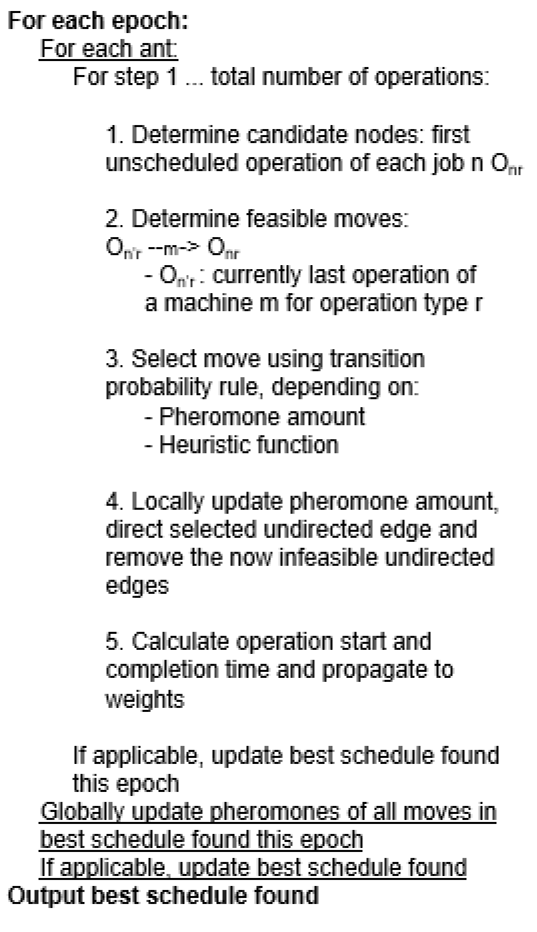
Figure 1: An example of a weighted disjunctive graph for the FJSP, taken from [8]. The specific variant of the problem investigated in the cited paper has some differences compared to one researched in this paper. For the DSM problem the different node outlines can be ignored. Also, there is only one weight per node, instead of four, and one weight per undirected edge, instead of zero in this graph.

In the graph there is a node for each operation plus a dummy start and end node. The picture shows three indices per operation node representing the job, operation number and machine pool. For the DSM problem, this can actually be simplified to two indices, since the machine pools are defined to be disjoint sets per operation type.

The nodes are connected with two types of edges: directed and undirected, both serving a different purpose. All consecutive operation nodes of the same job have a directed edge between them to specify the operation order within this job. All operation nodes that can be scheduled on the same pool of machines are connected with an undirected edge, one for each individual machine in this pool. During the ACO algorithm, these are either directed or removed, indicating that they respectively have been included in the schedule or can not be included in the schedule anymore.

Both the nodes and edges have weights. At the start each operation node has only one weight: the corresponding processing time, which is the same for each machine in the same machine pool. On runtime, it is convenient to also store the start and completion time of each operation on the node, once it has been scheduled. Since the cleaning times vary for the different machines, these cannot be stored on the nodes as done in [8, 9]. A solution is to instead put them on the undirected edges, one value for both directions specifying the order in which the connected operations may be included in the loading sequence of the respective machine.

Using the just described weighted disjunctive graph as input, ACO can be applied. Figure 2 shows conceptual pseudocode for this. A few parts of the algorithm require some more detailed explanation, which is given below.



```
For each epoch:
    For each ant:
        For step 1 ... total number of operations:

            1. Determine candidate nodes: first
            unscheduled operation of each job n $O_{nr}$

            2. Determine feasible moves:
            $O_{n'r}$ --m-> $O_{nr}$
                - $O_{n'r}$: currently last operation of
                a machine m for operation type r

            3. Select move using transition
            probability rule, depending on:
                - Pheromone amount
                - Heuristic function

            4. Locally update pheromone amount,
            direct selected undirected edge and
            remove the now infeasible undirected
            edges

            5. Calculate operation start and
            completion time and propagate to
            weights

        If applicable, update best schedule found
        this epoch
    Globally update pheromones of all moves in
    best schedule found this epoch
    If applicable, update best schedule found
Output best schedule found
```

Figure 2: The pseudocode for the implemented ACO algorithm.

The first component to discuss is the transition probability rule, which is used to select a move S from the set of feasible moves FM. A move M of ant is defined as walking over a specific an undirected edges in one of the two directions. This is based on both the pheromone amount τ of this move, indicating the previous experience with including it in the schedule, and a visibility function η, a heuristic predicting the quality of this move. The latter is especially important for the first generations of ants, which moves will this way not be completely random. This would be the case when only using the pheromones, which are initialized equally at the start of the ACO algorithm with a small positive constant τ_0.

Move selection is done using the following equation:

$$S = \begin{cases} \text{argmax}_{M \in FM}\{\tau(M)^{\alpha} * \eta(M)^{\beta}\} & \text{if } q \leq q_0 \\ J\colon P(J) = \frac{\tau(J)^{\alpha} * \eta(J)^{\beta}}{\sum_{M \in FM} \tau(M)^{\alpha} * \eta(M)^{\beta}} & \text{if } q > q_0 \end{cases} \quad (1)$$

Here q is a number which is each time randomly generated between 0 and 1. q_0 is the cutting exploration parameter from [14], also between 0 and 1, with the purpose of creating a balance between exploitation and exploration. α and β determine the influence of respectively the pheromones and the heuristic visibility function.

For the visibility function the Earliest Starting Time (EST) dispatching rule is chosen. This turned out to be the best heuristic for ACO among several dispatching rules [15]. In this reference the authors investigated the general problem without sequence-dependent setup times, but to make it suitable for DSM's variant of the FJSP, the cleaning times are simply incorporated in the starting time calculation:

$$st(O_{nr}) = \max\{ct(O_{n'r}) + clt(M), ct(O_{nr-1})\} \quad (2)$$

$$ct(O_{nr}) = st(O_{nr}) + pt(O_{nr}) \quad (3)$$

st, ct, clt and pt are respectively the start, completion, cleaning and processing times. $r-1$ refers to the previous operation of job n and M is the move $O_{n'r} \xrightarrow{m} O_{nr}$.

After selecting a move using the transition probability rule, the ant decreases the corresponding amount of pheromones using the following local updating rule:

$$\tau(M) = (1-\rho) * \tau(M) + \rho * \tau_0 \quad (4)$$

Where ρ is the evaporation rate, meant to avoid convergence to a local optimum.

Only at the end of an epoch, a positive global update is applied for all the moves of the best ant B in this generation:

$$\tau(M) = (1-\rho) * \tau(M) + \rho * \frac{1}{makespan(S_B)} \quad (5)$$

The size of the update is proportional to the makespan of the schedule S_B produced by the ant, which is the maximum completion time among all the operations. The better the solution, the more pheromones will be laid down on the edges.

4 Computational experiments and results

After discussing the practical setup and the evaluation of different hyperparameter values, the performance of the developed ACO algorithm is compared to that of the MILP solver.

4.1 Experimental Settings

All experiments were ran on a Intel Core i7-8750H with 16 GB RAM. All code was written in Python 3.8 and can be found on the Github repository used for this research. The implementation of the ACO algorithm was done individually, while for general code such as that for performance evaluation efforts were combined within the peer group. The MILP model was provided by the supervisor of the project, as well as the instances which can also be found on the repo. External software from Gurobi was used for solving the MILP model, specifically the deterministic concurrent simplex LP optimizer.

Hyperparameters for ACO were chosen in the following manner:

- β = 0.4, ρ = 0.12: best values selected according to results from [8].
- α = 1: although it is possible for this parameter to have a value different from 1, in practice it is easier to only vary beta to balance the influence of the pheromone amounts and visibility function. This is in line with the way the values for the above parameters were obtained.
- τ_0 = 0.1, q_0 = 0.9: resulting from experimental evaluation of different values as discussed directly below.

After initially playing around with the parameters to see which values work reasonably well, τ_0 = 0.1 and q_0 = 0.3 were chosen as a starting point. Subsequently for both these hyperparameters independently, five different values were tried:

- τ_0: 0.01, 0.05, 0.1, 0.5 and 1 (q_0 = 0.3)
- q_0: 0.1, 0.3, 0.5, 0.7 and 0.9 (τ_0 = 0.1)

The values of the other parameters were kept constant and as specified above. Both the solution quality, defined by the makespan of the returned schedule, and the runtime were measured, with the stopping condition for each run being three epochs without improvement of the makespan. For all values three runs were done and the average and standard deviation for both performance measures were calculated. The results can be seen in Table 1 for τ_0 and Table 2 for q_0. The top values in each cell are for the makespan, while the bottom ones represent the runtime in seconds.

Table 1 shows that the five tested values for τ_0 do not result in significantly different performance. This holds for both the makespan and runtime. Since τ_0 = 0.1 was also used for tuning q_0, it was decided to use this value for the numerical results in the next subsection.

Looking at Table 2, varying the value of q_0, does make a notable difference for the performance. The larger q_0, the lower the makespans that were found. Also, for the two highest values, the runtime significantly decreases. Overall, q_0 = 0.9 gives the best performance, and was chosen to be used for the coming experiments.

Instances	τ_0				
	0.01	0.05	0.1	0.5	1
0	23.0 += 0.0 0.1 += 0.0	24.0 += 1.0 0.1 += 0.0	23.0 += 0.0 0.1 += 0.0	23.7 += 1.2 0.1 += 0.0	23.7 += 1.2 0.1 += 0.0
1	37.0 += 1.0 0.6 += 0.1	37.0 += 1.0 0.5 += 0.0	36.7 += 0.6 0.5 += 0.1	36.7 += 0.6 0.5 += 0.1	36.7 += 1.5 0.7 += 0.1
2	51.3 += 2.1 1.5 += 0.4	50.0 += 2.0 2.9 += 1.2	50.0 += 1.7 2.6 += 0.9	50.3 += 0.6 2.2 += 0.3	49.7 += 0.6 2.0 += 0.5
3	64.3 += 2.5 4.5 += 0.3	65.0 += 2.6 4.3 += 0.9	63.0 += 2.0 3.6 += 0.7	65.3 += 1.2 5.0 += 0.6	65.3 += 0.6 3.9 += 1.1
4	79.3 += 0.6 6.5 += 0.7	79.3 += 2.1 7.4 += 1.7	79.0 += 1.7 7.8 += 2.2	78.3 += 1.2 9.5 += 3.5	79.0 += 1.0 10.2 += 4.1
5	93.3 += 1.5 14.4 += 5.2	92.0 += 1.0 10.1 += 1.4	95.3 += 2.9 14.1 += 0.6	95.3 += 1.5 15.5 += 7.1	96.0 += 1.7 14.1 += 4.7
6	107.7 += 3.8 24.1 += 6.2	108.3 += 1.2 25.7 += 6.0	108.0 += 1.0 21.3 += 7.6	109.0 += 2.0 17.1 += 4.0	108.7 += 0.6 18.9 += 3.8
7	122.0 += 2.0 33.0 += 9.6	122.3 += 4.0 25.0 += 3.3	121.3 += 2.5 23.7 += 3.0	122.7 += 4.2 29.0 += 3.4	122.7 += 1.2 51.1 += 27.0
8	133.0 += 5.3 57.3 += 12.6	139.7 += 1.5 51.4 += 16.7	135.3 += 2.5 50.9 += 10.6	139.7 += 1.2 35.5 += 4.0	138.7 += 1.2 43.9 += 4.6
9	152.7 += 3.8 62.5 += 36.8	151.3 += 4.5 62.8 += 21.5	154.3 += 0.6 47.0 += 5.7	150.3 += 2.5 54.9 += 5.7	150.3 += 2.9 66.6 += 33.5
10	165.3 += 4.5 124.1 += 35.5	167.7 += 2.1 77.6 += 15.0	165.3 += 1.5 91.3 += 8.0	163.7 += 4.0 95.7 += 41.1	167.3 += 4.0 76.4 += 20.8
11	181.7 += 0.6 140.0 += 46.9	180.0 += 2.0 112.4 += 20.8	181.3 += 1.5 121.8 += 57.7	181.0 += 3.6 94.8 += 40.7	184.3 += 0.6 116.5 += 20.2
12	196.3 += 6.4 181.7 += 140.6	194.0 += 2.0 145.2 += 28.0	193.7 += 6.4 129.0 += 36.6	196.3 += 4.0 156.7 += 45.1	194.0 += 5.6 145.7 += 13.4

Table 1: Experimental evaluation of τ_0 using β = 0.4, ρ = 0.12, α = 1 and q_0 = 0.3. The top entry in each cell is the average makespan += standard deviation, with below it the same for the runtime in seconds. Results were calculated over three independent runs, each having three epochs without improvement as stopping condition.

Instances	q_0				
	0.1	0.3	0.5	0.7	0.9
0	23.7 += 1.2 0.1 += 0.1	23.0 += 0.0, 0.1 += 0.0	23.0 += 0.0 0.1 += 0.0	23.0 += 0.0 0.1 += 0.0	23.0 += 0.0 0.1 += 0.0
1	40.0 += 3.5 0.6 += 0.2	37.3 += 1.5 0.7 += 0.2	35.3 += 0.6 0.4 += 0.1	35.0 += 0.0 0.6 += 0.2	34.7 += 1.2 0.4 += 0.1
2	58.7 += 1.5 2.1 += 1.0	51.3 += 1.5 2.0 += 0.7	47.7 += 1.2 1.6 += 0.3	44.3 += 0.6 2.1 += 0.2	44.7 += 1.2 1.3 += 0.2
3	72.7 += 3.2 4.1 += 1.0	66.0 += 1.0 5.2 += 3.2	60.7 += 1.5 3.2 += 0.4	57.0 += 1.0 3.6 += 0.3	56.0 += 1.0 4.3 += 0.0
4	86.0 += 1.0 8.4 += 3.0	79.3 += 2.3 8.8 += 1.9	73.0 += 1.0 6.9 += 3.0	67.3 += 1.2 9.9 += 2.9	65.0 += 0.0 6.9 += 2.0
5	106.7 += 3.5 13.3 += 2.9	94.7 += 0.6 16.7 += 4.1	82.7 += 0.6 13.3 += 3.9	81.0 += 1.0 10.0 += 2.4	78.0 += 0.0 9.0 += 1.1
6	123.7 += 3.2 25.1 += 8.2	109.0 += 1.7 25.7 += 3.9	98.3 += 1.2 20.8 += 0.1	92.7 += 0.6 15.3 += 1.8	90.3 += 0.6 17.7 += 4.8
7	143.0 += 2.6 28.7 += 0.4	121.3 += 4.0 35.2 += 5.8	111.3 += 2.1 25.5 += 5.0	104.7 += 2.5 22.5 += 3.0	100.7 += 1.5 26.8 += 5.3
8	158.0 += 3.6 44.1 += 12.1	141.0 += 2.6 31.2 += 0.4	124.3 += 1.2 36.3 += 12.5	116.7 += 1.5 32.0 += 8.0	112.3 += 1.2 29.8 += 7.5
9	175.3 += 3.2 73.4 += 17.1	149.3 += 2.5 66.3 += 12.2	137.7 += 1.2 59.3 += 9.4	128.3 += 0.6 40.8 += 5.3	123.3 += 2.1 51.6 += 29.9
10	192.7 += 4.0 73.2 += 14.4	164.0 += 6.1 92.7 += 32.5	146.3 += 3.2 86.7 += 8.0	138.3 += 2.1 76.8 += 6.6	135.3 += 1.2 71.4 += 2.9
11	208.7 += 4.0 133.0 += 19.1	182.3 += 2.1 78.0 += 10.2	162.0 += 1.7 109.5 += 28.0	152.3 += 0.6 103.0 += 19.9	147.0 += 0.0 71.0 += 17.4
12	234.3 += 1.2 119.8 += 24.0	198.0 += 4.6 113.5 += 39.4	175.7 += 2.3 135.5 += 69.1	165.7 += 2.3 95.5 += 11.7	159.3 += 0.6 113.9 += 31.1

Table 2: Experimental evaluation of q_0 using β = 0.4, ρ = 0.12, α = 1 and τ_0 = 0.1. The top entry in each cell is the average makespan += standard deviation, with below it the same for the runtime in seconds. Results were calculated over three independent runs, each having three epochs without improvement as stopping condition.

4.2 Performance comparison of ACO to MILP

Figure 3 and 4 show the performance of the ACO algorithm and MILP solver for respectively small and large time limits. No other stopping conditions are enabled to allow for an as fair comparison as possible, so for ACO as many epochs are executed as there is time for. Values of the hyperparameters are as defined in the above subsection. The makespan of the returned schedules is used as the performance measure and because of the randomness in the ACO algorithm, again averages over three runs were taken.

It can be seen in Figure 3 that with the small time windows, MILP can only keep up for the first few instances, being the simplest ones. As soon as the complexity rises, the method starts to return low quality solutions, followed by nothing at

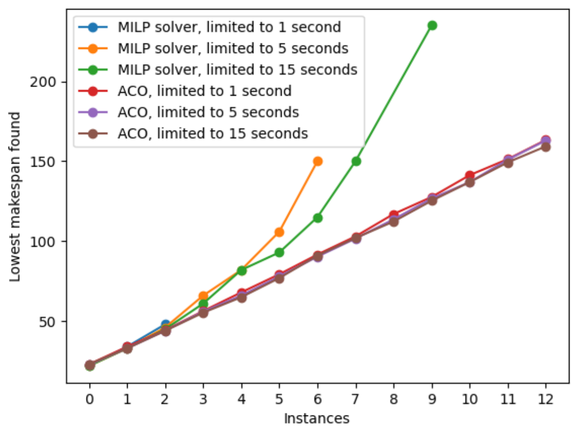


Figure 3: Comparison of the performance of the ACO algorithm and MILP solver for small time limits. For ACO, averages over three runs were taken, using $\beta = 0.4$, $\rho = 0.12$, $\alpha = 1$, $\tau_0 = 0.1$ and $q_0 = 0.9$.

all. ACO can solve all instances, also finding good solutions with low makespans. When there is a time for more epochs, a slight improvement can be seen in the solution quality, especially for the most complex instances.

Looking at Figure 4, only for the largest time limit MILP can finally find a solution for all instances. The makespans found by ACO are still significantly lower, but the gap between the methods decreases the higher the time limits. Compared to the small time ranges, the solutions ACO finds after more epochs are not much better, possibly because they are already at or near the global optimum.

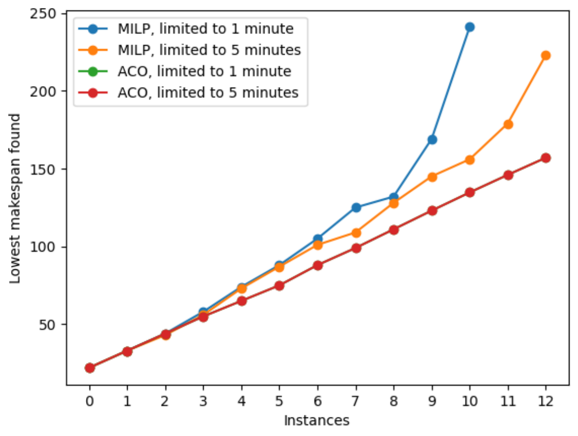


Figure 4: Comparison of the performance of the ACO algorithm and MILP solver for large time limits. For ACO, averages over three runs were taken, using $\beta = 0.4$, $\rho = 0.12$, $\alpha = 1$, $\tau_0 = 0.1$ and $q_0 = 0.9$.

5 Responsible Research

Investigating a concrete problem for the Dutch chemistry company DSM, the research could have some moral implications. With the firm potentially benefiting from the results, it's ethics were reviewed. DSM has a clear point of view, as can be read in their code of conduct: "we want to help solve some of the most pressing issues of today's world, like climate change and malnutrition. We use our science and expertise to develop innovative solutions that matter" [16, p. 3]. This research makes it possible to contribute towards this moral goal.

Another important aspect of responsible research is reproducibility. This is achieved by sharing the full experimental setup in Subsection 4.1, including hyperparameter values, hardware and external software utilized. Also, a link to the Github repository with the implemented code, including documentation, and all instances used for the experiments is provided. If someone would want to not only replicate the numerical results, but the complete research, all theoretical background behind the implemented algorithm is also provided in Section 3, with references where applicable. With the problem being formally defined in the section before that and all research questions and general background information discussed in the introduction, full reproducibility is made possible.

6 Discussion

The presented work is not of as high quality and quantity as it could be, at least not as personally wished and used to. With only ten weeks available for the project the time was already very limited, but personal circumstances also impacted the research significantly.

Most important, the shown results are only based on thirteen instances. No time was found to create more and do experiments with these. This makes it hard to validate drawing general conclusions about the performance of ACO for the investigated problem. Ideally, the ACO algorithm and MILP solver would have been tested on a large number of instances of high complexity, comparable to real-world scenarios. Even better, there would have been different groups of instances with similar properties, such as many machines and/or many jobs. This way more interesting and general conclusions could have been made about the ACO method: in what cases it works well, but also when it does not.

Another crucial part of DSM's challenge are the multiple conflicting objectives explained at the end of Section 2. It was not achieved to implement these and do experiments using a multi-objective function; all results only take the makespan of the schedules into account.

Additionally, the tables for the hyperparameter evaluation show there is still quite a bit of variance when doing three independent runs to base the results on. Averages over more measurements would increase reliability and possibly also show some consistent differences in performance when varying τ_0.

Despite all this, there are some valuable insights this research gives. Most important, it is clear that on the tested instances, the ACO algorithm significantly outperforms the

MILP solver when optimizing for the makespan. ACO is able to find good solutions exceptionally fast. Even when limited to one second, it finds better solutions for all instances than MILP in 5 minutes, especially for the most complex ones.

Also, the results of the hyperparameter evaluation evidently show that for all used instances, high values of q_0 give better performance than lower ones. For τ_0, the experiments might indicate that, next to averaging over more runs, it would be good to try a broader range of values. Potentially these would show impact of the parameter on the solution quality or runtime of the algorithm.

7 Conclusions

This paper examines an ant colony optimization approach for solving a practical problem the Dutch chemistry company DSM faces. It can be characterized as a specific case of the multi-objective FJSP with sequence-dependent setup times. There are N jobs, each of one of E enzyme types and consisting of a subset of three possible unit operations. There are M machines, divided of three disjoint sets for the different operation types. All operations of all jobs need to be processed on a machine, which takes a certain amount of time defined by operation type and enzyme type of the overarching job. Also, specialized cleaning is required when a machine consecutively handles different enzyme types. The duration of this setup time depends on the respective machine and enzyme types. Finally, the conflicting objectives are minimizing the makespan of a schedule, meeting due dates per job, and keeping the idle time of the machines as low as possible.

To be able to apply ACO to the problem, instances need to be represented as a weighted disjunctive graph. There is a node for every operation, with directed edges to specify the order within each of the jobs. Undirected edges connect operations of the same type that can be scheduled on the same pool of machines. The processing times are specified as weights on the nodes, the cleaning times as weights on the undirected edges.

The ACO algorithm consists of a number of epochs, each consisting of multiple ants creating a feasible schedule. This is created one operation at the time. What scheduling choice is made at each step is defined by a transition probability rule depending on pheromones and a visibility function. The first indicate previous experience of ants in prior generations, the second is an heuristic based on the earliest starting time possible for a certain operation. Pheromone amounts are updated using both a negative local updating rule after every scheduled operation of each ant, and a positive global update at the end of a whole epoch, only favoring the best schedule generated by an ant in that generation. Finally, the best schedule found over all epochs is returned.

Experimental evaluation of two hyperparameters is performed. First τ_0, a small positive constant defining the initial amount of pheromones for each possible scheduling move. The different values tried did not consistently impact the makespan nor runtime for the used instances. q_0, the cutting exploration parameter determining the balance between exploitation and exploration when scheduling an operation, did have a lot of influence on both performance measures. For all tested instances, the higher the value of the parameter, the better.

To determine whether the implemented ACO approach is efficient, it is compared to a provided MILP solver for a number of different time limits. The makespan of the returned schedules is used as performance measure. On the used instances, ACO significantly outperforms MILP. As soon as the complexity rises, MILP quickly struggles to find feasible solutions. First bad solutions are returned, followed by no solutions at all. In contrast, ACO is able to find good solutions exceptionally fast. Even when limited to one second, it finds better solutions for all instances than MILP in 5 minutes, especially for the most complex ones. Based on the tested instances, ACO is an efficient method to solve the production scheduling problem of DSM.

The research is carried out responsibly, having positive moral implications by potentially contributing towards DSM's goals. Reproducibility is made possible by sharing the full experimental setup, including code and instances, and all necessary background information.

Future work entails evaluating the performance of ACO on a larger number of and more complex instances, to be able to make more interesting and general conclusions about the method. Secondly, the multiple conflicting objectives can be implemented and experimented with, which was not achieved within this research.

References

[1] Jin Xie et al. "Review on flexible job shop scheduling". In: *IET Collaborative Intelligent Manufacturing* 1.3 (Sept. 2019), pp. 67–77. ISSN: 2516-8398. DOI: 10.1049/iet-cim.2018.0009.

[2] F. Pezzella, G. Morganti, and G. Ciaschetti. "A genetic algorithm for the Flexible Job-shop Scheduling Problem". In: *Computers and Operations Research* 35.10 (2008). ISSN: 03050548. DOI: 10.1016/j.cor.2007.02.014.

[3] P Brucker and R Schlie. *Computing Job-Shop Scheduling with Multi-Purpose Machines*. Tech. rep. 1990, pp. 369–375.

[4] Christoph S. Thomalla. "Job shop scheduling with alternative process plans". In: *International Journal of Production Economics* 74.1-3 (Dec. 2001), pp. 125–134. ISSN: 09255273. DOI: 10.1016/S0925-5273(01)00119-0.

[5] Imran Ali Chaudhry and Abid Ali Khan. "A research survey: review of flexible job shop scheduling techniques". In: *Intl. Trans. in Op. Res* 23 (2016), pp. 551–591. DOI: 10.1111/itor.12199.

[6] Alberto Colorni et al. "Distributed Optimization by Ant Colonies". In: 1992.

[7] Alberto Colorni et al. *Ant system for Job-shop scheduling*. Tech. rep. 1. 1994.

[8] Andrea Rossi and Gino Dini. “Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method”. In: *Robotics and Computer-Integrated Manufacturing* 23.5 (Oct. 2007), pp. 503–516. ISSN: 07365845. DOI: 10.1016/j.rcim.2006.06.004.

[9] Andrea Rossi. “Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships”. In: *International Journal of Production Economics* 153 (July 2014), pp. 253–267. ISSN: 0925-5273. DOI: 10.1016/J.IJPE.2014.03.006.

[10] Li-Ning Xing, Ying-Wu Chen, and Wei Yang. *Double Layer ACO Algorithm for the Multi-Objective FJSSP*. Tech. rep. 2008, pp. 313–327.

[11] Rong Hwa Huang, Chang Lin Yang, and Wei Che Cheng. “Flexible job shop scheduling with due window - A two-pheromone ant colony approach”. In: *International Journal of Production Economics* 141.2 (Feb. 2013), pp. 685–697. ISSN: 09255273. DOI: 10.1016/j.ijpe.2012.10.011.

[12] Lei Wang et al. “Flexible Job Shop Scheduling Problem Using an Improved Ant Colony Optimization”. In: *Scientific Programming* 2017 (2017). ISSN: 10589244. DOI: 10.1155/2017/9016303.

[13] *Job-shop scheduling*. URL: https://en.wikipedia.org/wiki/Job-shop_scheduling.

[14] R Kumar, M K Tiwari, and R Shankar. *Scheduling of ¯exible manufacturing systems: an ant colony optimization approach*. Tech. rep. 2003.

[15] Christian Blum and Michael Sampels. “An ant colony optimization algorithm for shop scheduling problems”. In: *Journal of Mathematical Modelling and Algorithms* 3.3 (2004), pp. 285–308. ISSN: 15701166. DOI: 10.1023/B:JMMA.0000038614.39977.6f.

[16] *One DSM - Code of Business Conduct*. URL: https://www.dsm.com/content/dam/dsm/suppliers/en/documents/code-of-business-conduct-en.pdf.