A Heuristic Algorithm for the Flexible Job Shop Problem with Changeover Times

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

24-6-2022

Abstract

In this paper a heuristic algorithm is described that can constructively produce solutions to a variant of the Flexible Job Shop Problem (FJSP) that introduces changeover times between each pair of two operations consecutively performed on a machine. The performance of the heuristic algorithm is compared to the performance of an exact solver. Seven heuristics are compared for the FJSP variant with changeovers. The main objective function used is the makespan of the created schedules. The difference that occurs in the quality of the seven heuristics is examined also when instead the total lateness across all jobs is chosen as objective function. It concludes that a heuristic algorithm allows for the creation of good feasible solutions to complex problem instances for the FJSP variation with changeover times in under 30 seconds and that it outperforms an exact algorithm for the FJSP with changeover for use cases where efficiency is important and runtime is limited.

1 Introduction

The schedule for a production line in a manufacturing plant has a direct impact on its efficiency. An inefficient schedule will lead to an inefficient production line. One problem often used to describe the scheduling process for a manufacturing plant is the strongly NP-hard Job-Shop Problem (JSP) [1]. In the JSP, the goal is to create a schedule for a set of jobs given a set of machines. Each job consists of multiple operations, which have to be performed on a specific machine out of the set of machines. The time needed for each operation is known.

The Flexible Job-Shop Problem (FJSP) is a more complex generalisation of the Job-Shop Problem. In the FJSP every operation can be performed on a subset of all existing machines, rather than on only one specific machine. This adds complexity, since now rather than only the problem of sequencing all operations, there is also the subproblem of assigning every operation to a machine from the set of machines on which this operation can be performed. It was first introduced by Brucker and Schlie [2] in 1990.

Mapping the reality of a complex production plant to a problem instance for the FJSP often requires simplifications or abstractions to be made. Many real life use cases that could be translated to the FJSP have transport times, cleaning times or setup times for machines. The inability to easily express these in the regular FJSP means that algorithms that can solve the FJSP cannot be used at all in certain fields.

A variation on the FJSP can be made however that can include these cases and thus be useful for many real life scenarios which the regular FJSP can not sufficiently map. We call this variation the changeover variation. In this variation we introduce changeover times. For each machine and combination of products of two sequentially scheduled operations on the same machine a specific time is reserved that has to be scheduled after the first of the two operations finishes and before the next operation can start.

The scheduling problem of the enzyme manufacturing line of a DSM plant is an example of this changeover variation of the FJSP. In this plant machines have to be cleaned between every two operations with the time needed for cleaning dependent on both the machine and the products used in the two operations. In this paper the situation of this DSM enzyme manufacturing line will be used as an example of the FJSP variation with changeovers.

An exact algorithm exists that can solve the changeover variation of the FJSP. It aims to find the optimal solution, and works on any problem instance for this problem. Due to this algorithm aiming to find the optimal solution, as the size of the instances increases, the runtime increases rapidly. This makes it infeasible for many real-life use cases where efficiency is important and only a limited runtime is available to create schedules.

A heuristic algorithm could be a good solution in such cases, where both the quality of the created schedules and the time taken to create the schedules are important. Heuristic algorithms have been shown to allow for the creation of reasonably good and feasible solutions to the regular FJSP in almost non-zero time [3] and the hypothesis is that this will also be the case for the changeover variation of the FJSP

Creating a heuristic algorithm for this variation of the FJSP with changeover times will be useful in the first place to determine whether heuristic approaches to this variant on the FJSP have any merit, and whether further research is worthwhile.

A successful heuristic algorithm is also an essential contribution to more complex algorithms. Most of the state of the art approaches to solving the FJSP are meta-heuristic algorithms, for which a good heuristic is necessary. A successful meta-heuristic algorithm could thus potentially be developed using the heuristic algorithm or some of its underlying heuristics. The resulting algorithm would likely be less time-efficient but with a higher quality of resulting schedules.

The main goal of this paper is to propose a heuristic algorithm that can solve the changeover variation on the FJSP and to determine how this heuristic algorithm performs compared to an exact algorithm. A secondary goal is to compare the use of different heuristics and determine which are the most useful for effectively creating schedules for this problem. Another secondary goal is to determine how a change in objective function will impact the usefulness of heuristics.

The rest of the paper is structured as follows: Section 2 explains in-depth the FJSP with changeover times and gives a background of other research done into the FJSP and variations on it. Section 3 explains the heuristic algorithm for the FJSP with changeover times. Section 4 describes the experiments performed and the results of these experiments. Section 5 discusses and reflects on the way in which the research has been performed and how responsibly this has been done. Section 6 discusses and reflects on the results taken from the experiments. Section 7 describes the conclusions that this research has led to and potential future work that can be done.

2 The Flexible Job-Shop Problem with Changeover Times

In this section first the problem will be defined in further detail. Afterwards, related literature and the gap in this literature that this paper aims to fill will be discussed.

2.1 Formal Problem Description

A formal description of the FJSP variant with changeover times can be given as:

1. There is a set of jobs $J = \{J_1, J_2, ..., J_n\}$ with size n that has to be performed on a set of machines $M = \{M_1, M_2, ..., M_m\}$ with size m
2. Each job J_i consists of a sequence of operations $(O_{i,1}, O_{i,2}, ..., O_{i,l_i})$ of length l_i
3. Each operation j of a job i has to be completed on one machine out of a set $M_{i,j}$ of available machine for this specific operation.
4. Completing operation $O_{i,j}$ on machine M_k takes $P_{i,j,k}$
5. Between the completion time of operation $O_{i,j}$ on machine M_k and the starting time of the next scheduled operation $O_{x,y}$ on machine M_k has to be a gap of at least the cleaning time for machine k and jobs i and x given by $C_{i,x,k}$

Throughout this paper the following assumptions are also made:

1. All machines and jobs are available at time $t = 0$
2. Operation $O_{i,j}$ can only start being processed when all operations $O_{i,x}$ where $x \leq j$ have been completed.
3. A machine can only process one operation at a time.
4. Once started the processing of an operation cannot be paused.
5. Transportation times are assumed to be included in the processing time, such that if operation $O_{i,j}$ finishes at time t, it is possible for the processing of operation $O_{i,j+1}$ to start at time t.

An exact algorithm to solve the scheduling problem for the enzyme manufacturing in a DSM plant, which is an example of the FJSP variation with changeover times, has already been created. For this the formal definition of the FJSP with changeover variation has been turned into an exact mathematical formulation [4]. This mathematical formulation can be given to a Mixed-Integer Linear Programming (MILP) solver and this MILP solver can thus be used to then solve the scheduling problem for the DSM plant. The MILP solver is an exact solver, that takes as input a mathematical formulation of the problem and a problem instance and then uses a linear-programming based Branch & Bound algorithm to find the optimal solution [5].

2.2 Related Literature

Much research has been done into the FJSP and variations on the FJSP [6]. The algorithms created to solve the FJSP can mainly be divided into three categories: exact algorithms, heuristic algorithms and meta-heuristic algorithms [7].

Research has been done into meta-heuristic approaches to the FJSP by Fattahi *et al.* [8] who created both hierarchical and integrated algorithms for solving the FJSP that use all possible combinations of simulated annealing and tabu search. As well as by Gao *et al.* [9] who created a hybrid genetic algorithm with a variable neighborhood descent strategy for the FJSP that uses two vectors as a representation. Buddala and Mahaptra [10] proposed an integrated algorithm using a teaching-learning based optimization method to solve the FJSP.

A heuristic algorithm was created by Ziaee [3] that managed to obtain effective solutions to the FJSP with a near zero runtime and performance comparable to certain meta-heuristic approaches. Sotskov and Gholami [11] created five heuristic algorithms for the FJSP using a mixed-graph representation. Their proposed heuristic algorithms use two different objective functions: the makespan of a schedule and the sum of completion times of all operations. A heuristic dispatching algorithm for the FJSP has been created and applied to a real use-case to show its practical applicability by Ortíz *et al.* [12]. Research into multi-objective flexible job-shop problems has been done by Pérez and Raupp [13], who proposed a hierarchical heuristic algorithm based on the Newton's method for optimization problems with multiple objectives.

The general FJSP is well researched and many algorithms for it have been proposed, mainly consisting of exact, heuristic and meta-heuristic approaches. Not much research however has been done into the changeover variation on the FJSP, although an exact algorithm for it has been created. The changeover variation on the FSJP allows real life use-cases that are not expressible as a regular flexible job shop problem to now be expressed with this variant of the FJSP. Creating a heuristic algorithm for the changeover variation on the FJSP and testing its efficiency compared to that of an existing exact algorithm for this variation is important to develop an understanding of the potential for heuristic approaches to this variation on the FJSP, as well as to provide information necessary for the creation of successful meta-heuristic approaches for this variant.

3 The Heuristic Algorithm

The heuristic algorithm uses a weighted combination of several heuristics. It is based on the heuristic algorithm for solving the regular FJSP proposed by Ziaee [3], but with changes meant for the changeover variation on this FJSP.

An algorithm that uses a weighted combination of heuristics allows for the use of multiple heuristics to constructively create a schedule. Assuming the weights used for combining the heuristics are chosen properly it will perform better then any of the used heuristics would independently.

Further, an algorithm using a weighted combination of heuristics allows for easy comparison between these heuristics. By setting the weight of a heuristic to zero, the performance of the algorithm without the impact of this particular heuristic can be tested. This, combined with the information of which weights result in the best solutions, is helpful for determining how important and essential each heuristic is for the creation of high quality schedules.

The main objective function used to judge the quality of the schedules created by the algorithms is the total makespan. The reason for using this objective function is that it is the most widely used objective function in research on the FJSP [6]. The total makespan is defined as the maximum completion time between all operations and the aim is to minimize this value. Another objective function that is used is that of lateness, which in real life scenarios where orders have different due dates is often of a higher importance than the total time taken for the schedule as a whole. Lateness in this case is defined as $\sum_{i=1}^{n} max(0, t_{e,i,l_i})$.

Some notation used in the heuristic algorithm:

s_x	The sum of all $P_{i,j,k}$ where $i = x$ and where $k \in M_{i,j}$
c_y	The sum of all $C_{i,x,k}$ where $y = k$
l_{max}	The maximum value of all l_i
L	A large number
w_x	The weight used for heuristic x
$h_{x,i,j,k}$	The value of heuristic x for operation $O_{i,j}$ and machine M_k
k_{max}	The maximal current completion time of all operations that are scheduled on M_k
$t_{s,i,j}$	The starting time of operation j of job i
$t_{e,i,j}$	The ending time of operation j of job i
H_{total}	The weighted sum of the heuristics
d_i	The due date of job i

Algorithm 1 Heuristic algorithm for FJSP with changeover

```
$i\_sort \leftarrow$ list of jobs sorted on increasing $s_i$
$k\_sort \leftarrow$ list of machines sorted on increasing $c_k$
for $j := 0$ to $l_{max}$ do
    while the $j_{\text{th}}$ operation of some job is not yet scheduled do
        $H^*_{total} \leftarrow L$
        for $i^* := n$ to $0$ do
            $i \leftarrow i\_sort_{i^*}$
            if $j \leq l_i$ and $O_{i,j}$ not yet scheduled then
                for $k^* := 0$ to $m$ do
                    $k \leftarrow k\_sort_{k^*}$
                    if $k \in M_{i,j}$ then
                        $H_{total} \leftarrow \sum_{x=1}^{5} w_x \cdot h_{x,i,j,k}$
                        if $H_{total} \leq H^*_{total}$ then
                            $H^*_{total} \leftarrow H_{total}$
                            $z \leftarrow i$
                            $y \leftarrow k$
        $t_{s,z,j} \leftarrow y_{max} + C_{y,z,prev}$
        $t_{e,z,j} \leftarrow t_{s,z,j} + y_{z,j}$
        Schedule operation $j$ of job $z$ on machine $y$ with starting time $t_{s,z,j}$ and ending time $t_{e,z,j}$
```

The pseudocode for the heuristic algorithm is shown in Algorithm 1. The heuristic algorithm considers combinations of operations and machines. It initially only considers combinations for the first operation of every job. The order in which operations are considered is such that operations of jobs which have a large total processing time are considered before those of jobs with a smaller total processing time. Once the firsts operation for every job has been scheduled, the algorithm then considers the operation and machine combinations for the second operation of each job which has 2 or more operations. This process continues until all operations of all jobs have been scheduled.

For each operation and machine combination that the algorithm considers, it calculates the values of several heuristics. It then calculates a final heuristic value, which is a weighted combination of those separate heuristics. Using these final heuristic value, it determines the next job and machine combination to schedule by picking the minimum. The weights used to combine the heuristics into one final value are not static, the algorithm considers multiple weight values for each heuristic and uses the values that give the best results. The weight values for each heuristic are bounded by both a lower and an upper bound, which are given to the algorithm as hyper-parameters.

The six heuristics that are used in the algorithm are:

$h_{1,i,j,k} = max(k_{max}, t_{e,i,j-1}) + P_{i,j,k}$
$h_{2,i,j,k} = max(0, (t_{e,i,j-1} - k_{max}))$
$h_{3,i,j,k} = max(0, (k_{max} - t_{e,i,j-1}))$
$h_{4,i,j,k} = P_{i,j,k}$
$h_{5,i,j,k} = s_i$
$h_{6,i,j,k} = C_{k,i,prev}$
$h_{7,i,j,k} = d_i$

The first heuristic calculates the expected completion time of operation $O_{i,j}$ if it were to be scheduled on machine M_k. In other words, it calculates an estimate for the new k_{max} of machine M_k if operation $O_{i,j}$ were to be scheduled on it. We aim to minimize the final value of k_{max} for each machine, since the total makespan is equal to the maximum k_{max}.

The second heuristic calculates the expected idle time of machine M_k if operation $O_{i,j}$ were to be scheduled on it next. The higher the idle time throughout the schedule, the longer the total completion time of all operations and thus the higher the makespan, which is why we also want to minimize this value.

The third heuristic calculates the expected waiting time between the previous operation of job j, $O_{i,j-1}$ and the current operation of job j, $O_{i,j}$ if it were to be scheduled on machine M_k, The longer the wait times, the longer we expect the makespan of the total schedule to be, so this value should also be minimized.

The fourth heuristic is the expected time taken for this machine and job combined. Taking a longer time per operation could lead to a longer total time taken across all operations which can lead to a higher makespan. A lower value for this heuristic would thus be expected to lead to lower makespan.

The fifth heuristic is the total combined expected time of all operations of job J_j. This heuristic will ensure that operations part of longer total jobs will be prioritized. This is useful when we are attempting to minimize the makespan, because we want the maximum completion time between all jobs to be as low as possible, and thus it makes sense to prioritize longer jobs. If we were to prioritize shorter tasks, we would expect the differences between the maximum completion times of jobs to be larger and the maximum value of completing any job to thus be larger as well.

The sixth heuristic is the expected changeover time that will be necessary when scheduling $O_{i,j}$ on machine M_k taking into account the previously scheduled operation on machine M_k. Minimizing the total amount of time spent on changeovers will minimize the total amount spent and with that also be helpful to lowering the makespan.

The seventh heuristic is the due date of the job for which the operation is to be scheduled. This heuristic will allow the algorithm to take into account the due dates of each job. When the objective function is the makespan we would not expect this to be helpful and only to have a random effect. When the objective function is to minimize the lateness, it would be preferential to first schedule jobs with an earlier due date, for which this heuristic would be helpful.

4 Experimental Setup and Results

4.1 Experimental Setup

The heuristic algorithm is coded in python and the MILP is also coded in python using the Gurobi [14] package. All experiments ran on an Intel I7, 2.80GHz and 16GB ram machine.

Set	N instances	Min size	Max size	Mean size
1	13	6	78	42
2	100	76	81	78
3	130	6	78	42

Table 1: An overview of the instance sets used for the experiments.

Multiple sets of instances have been used to run experiments on. An overview of the instance sets can be seen in Table 1. The size of an instance is expressed by the amount of jobs in the instance. For all instances the amount of machines is equal to nine. Set 1 is a benchmark set originally created for the MILP. It consists of thirteen instances of linearly increasing sizes. Benchmark Set 2 and Set 3 have been specifically created for this research. Set 2 consists of 100 instances of similar size. The average size of the instances of Set 2 are equal to the size of the largest instance in Benchmark Set 1. Set 3 consists of thirteen groups of ten instances with equal size, with the size linearly increasing between the groups similarly to the individual instances in Set 1. Set 1 has been used to compare the MILP with a long runtime limit, to the Heuristic Algorithm with a short runtime limit, to determine whether even with a fraction of the runtime it can still generate better results. Instance set 2 has been used to compare the different heuristics, finetune the hyper-parameters and to determine how the choice of heuristics and hyper-parameters could be impacted by the choice of bjective function.

The hyper-parameters for the heuristic algorithm are the lower and upper bounds for the weight of each heuristic and the step sizes with which the weights for each heuristic should be changed. This means that there are three hyper-parameters per heuristic, for a total of fifteen hyper-parameters. The choices for these three hyper-parameters per heuristic impact both the runtime as well as the quality of results for the algorithm and properly finetuning them is thus of utmost importance. The hyper-parameters tuning was first done with the makespan as objective function. The initial values used in this finetuning process were based on the values used by Ziaee [3]. The values for each hyper-parameter were then individually refined experimentally. For the lower bounds and upper bounds of each heuristic this was done by changing the hyper-parameter values in both the positive and negative directions and using the quality of the schedules created by the heuristic algorithm with these hyper-parameters to determine if the hyper-parameters should be changed in this direction and if further changes in this direction should be tested. The step sizes have been chosen with a balance of efficiency and schedule quality in mind. The aim for this was to find the largest possible step size which does not cause a substantial decrease in the average schedule quality. These values were found by using an iterative increase of the step sizes for each heuristic. This has led to the final values shown in Table 2.

Heuristic	Lower bound	Upper bound	Step size
1	6	8	2
2	0	3	3
3	-4	0	4
4	-3	3	3
5	-1	-1	0
6	0	5	5
7	0	0	0

Table 2: The hyper-parameters for the heuristic algorithm with the makespan as objective function that have been determined using finetuning.

Heuristic	Lower bound	Upper bound	Step size
1	4	8	4
2	0	3	3
3	-4	0	4
4	-3	3	3
5	-1	-1	0
6	10	20	10
7	0	6	2

Table 3: The hyper-parameters for the heuristic algorithm with the lateness as objective function that have been determined using finetuning.

For the heuristic algorithm in the case where the lateness is used as objective function, the hyper-parameters have been finetuned in a seperate process. For this process the initial values used were the values found in the hyper-tuning process for the heuristic algorithm with the makespan as objective function, which can be found in Table 1. The further process of the finetuning was not changed from the process for the case where the makespan was used as objective function, described in the previous paragraph. This led to the final hyper-parameter values shown in Table 3.

The hyper-parameter finetuning for the heuristic algorithm has been done on Set 2 of instances. This set is not used for any of the experiments. The choice for using a separate set of instances from the sets of instances used in the further experiments has been made to ensure that the hyper-parameters do not get tuned specifically to the instances in the comparison set, which could lead to unfairness in the experiments.

The feasibility of all the schedules created by the heuristic algorithm as well as the schedules created by the exact MILP solver during all the performed experiments was verified using the following method. The method firstly checks if all operations of all jobs have been scheduled in the proposed schedule by the algorithm. It then checks if there is enough time in between each pair of two operations on each machine for the needed cleaning time for this particular machine and pair of operations. It further checks if the time taken to schedule operation is indeed the time needed for that operation on that machine. Lastly it also checks that the operations of a certain job are correctly ordered, and that multiple operations of the same job can thus not be performed simultaneously. All of these checks combined should ensure that the solutions created by the algorithms are indeed feasible solutions to the problem instances.

4.2 Results

In order to determine whether the heuristic algorithm is more efficient than the exact algorithm we compare the performance of the heuristic algorithm to that of the exact MILP solver in two experiments.

The aim of the first experiment is to determine how well the heuristic algorithm performs compared to the MILP solver in cases where more runtime is available. For this experiment the MILP solver has been ran with a runtime limit of 2700 seconds which corresponds to 45 minutes. The heuristic algorithm has been ran with a 30 second runtime limit and using the hyper-parameter values shown in Table 2. The objective function used is the makespan.

For this experiment both of the algorithms ran on Set 1 of instances. This set contains problem instances of different sizes and the results therefore allow us to compare the performances of the two algorithms across the range of instance sizes. The reason for choosing a set which contains a lower amount of instances has been made with the feasibility of the experiment in mind, considering the relatively long runtime limit for the MILP solver.

Figure 1 graphically shows the differences in the minimal makespan found by both the MILP and the heuristic algorithm (HA). The exact numerical results of this experiment can be found in Table 3. For the lower instances with smaller sizes, the MILP manages to find an optimal solution within the time limit, and performs therefore performs or equal to the heuristic algorithm. We do see that in these cases the heuristic algorithm is still quite close to the MILP when comparing the quality of the schedules. As the sizes of the instance increase, we see that the heuristic algorithm starts to increasingly outperform the MILP solver. From instance 5 onward the heuristic algorithm outperforms the MILP and starting from instance 7 the difference in schedule quality becomes especially significant. Even though the time taken by

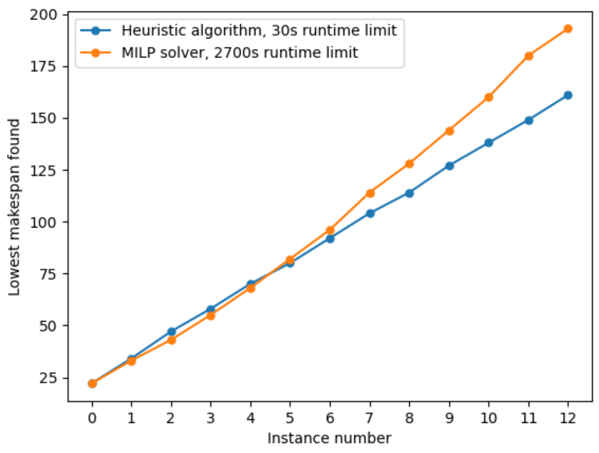

Figure 1: Results of running the heuristic algorithm with a 30 second runtime limit and the MILP solver with a 2700 second runtime limit on instance Set 1 with the makespan of created schedules used as objective function.

the heuristic algorithm is only a fraction of the time taken by the MILP solver, it still significantly outperforms the MILP solver for the larger instances.

We can conlude that the heuristic algorithm outperforms the MILP solver for larger size instances when runtime is limited to 45 minutes or less and that for small instances although not performing as well as the MILP it still manages to create good feasible results.

The aim of the second experiment is to further compare the MILP solver and the heuristic algorithm in cases where the runtime limit is much lower. For this experiment the MILP solver and heuristic algorithm ran with a runtime limit of 30 seconds. The heurstic algorithm used the hyper-parameter values shown in Table 2 and the objective function used was the makespan.

For this experiment Set 3 of instances has been used. Once again using a range of instance sizes allows for a better comparison between the two algorithms. Unlike in the first experiment, it is now feasible to use an instance set containing a larger amount of instances which will make the results of the experiment more convincing.

Figure 2 compares the average minimal makespans found by the two algorithms per group of instances of the same size in the second experiment. With this runtime the MILP solver did not manage to find a solution for each problem instance. For problem instances where no solution was found a value of 388 was taken as makespan found by the MILP solver. This is the highest makespan found by either of the algorithms across all instances.

Figure 3 shows the minimal makespans found by the two algorithms for each instance individually. In this case the instances for which no solution has been found have been given a value of 400, to ensure that these values are higher than any of the found values.

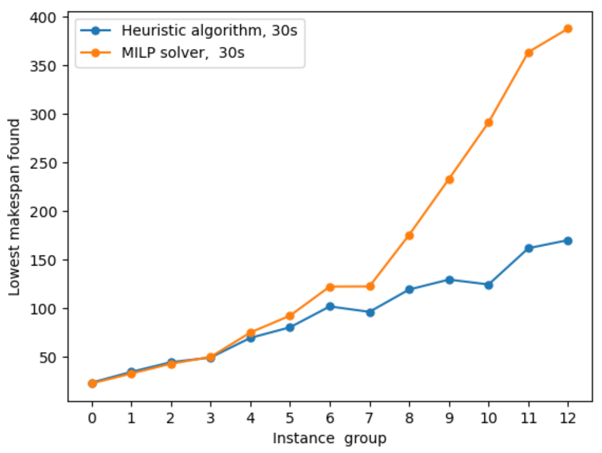

Figure 2: Average result per group of instances of the same size when running the heuristic algorithm with a 30 second runtime limit and the MILP solver with a 30 second runtime limit on instance Set 3 with the makespan of created schedules used as objective function. If no solution was found by either of the algorithms a value of 388 was used for the makespan.

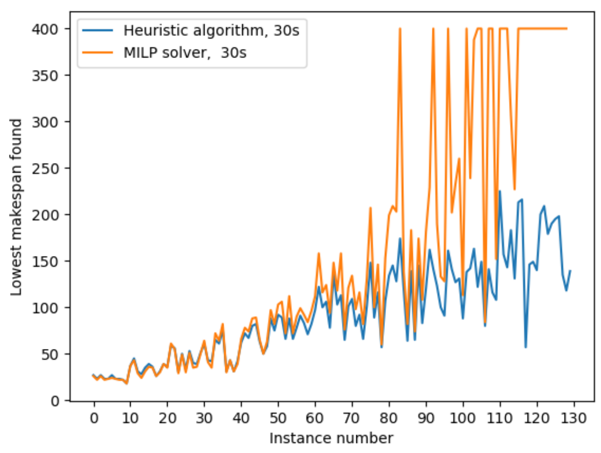

Figure 3: Results of running the heuristic algorithm with a 30 second runtime limit and the MILP solver with a 30 second runtime limit on instance Set 3 with the makespan of created schedules used as objective function. If no solution was found a value of 400 was used for the makespan.

Figure 2 shows that asin experiment one, for the first few instance sizes the MILP once again outperforms the heuristic algorithm slightly. For the larger instances we once again see the heuristic algorithm increasingly outperform the MILP. The difference in schedule quality found by the MILP and the heuristic algorithm for the larger instances are much larger than the differences in the first experiment, not only due to the cases when the heuristic algorithm could not find any solution as can be seen in Figure 3. This is to be expected due to the heuristic algorithm having the same runtime in both experiments while the MILP solver has a lower runtime in the second.

From the second experiment we can conclude that when the runtime is very limited, the heuristic algorithm can consistently outperform the MILP solver on a large number of instances across a different range of instance sizes when the instance contains 36 or more jobs. For the instances with a lower size, the MILP outperforms the heuristic algorithm, but the heuristic algorithm still manages to create schedules of a quality very close to those created by the MILP.

The aim of the third experiment is to determine the importance of each of the six heuristics used within the heuristic algorithm for creating schedules of a high quality. This has been examined by running the heuristic algorithm on Set 3 once for each heuristic and setting the weight of this heuristic to zero for this run of the algorithm. When the weight is set to zero for a certain heuristic this heuristic has no impact on the final schedule created by the algorithm. This allows a comparison between the results for all of these runs of the algorithm to determine how big the impact of certain heuristics is on the final resulting schedules. This experiment has been done using the makespan as objective function.

Figure 4 and 5 graphically show the decrease in performance caused when switching of the different heuristics. It displays the difference between the results created by the algorithm without the heuristic and the full heuristic algorithm, such that the heuristic algorithm with all heuristics turned on would get a value of zero for each instance.

In Figure 4 we see that when heuristic 1 is removed the makespan found increases majorly. Without the use of heuristic 1, the heuristic algorithm is not able to find good and feasible solutions anymore. We also see that the impact of the other five heuristics is much lower than that of the first heuristic. This shows that the first heuristic is the most essential for the heuristic algorithm.

In Figure 5 we see that out of the other heuristics heuristic five has the biggest impact on the results since removing it leads to the biggest difference for every group of instances besides the first. We also see that heuristic 2 has the lowest impact on the final results create by the algorithm. Further we can see that excluding heuristic 6 or heuristic 4 leads to a bigger decrease in schedule quality than excluding heuristic 2 and heuristic 3. We can also note that the impact of heuristic 6 for the smaller instances is larger than its impact on the bigger instances, where removing the impact of heuristic 6 leads to a much less significant decrease than for the smaller sized instances.

From this experiment we can conclude that heuristic 1 is by far the most important heuristic used in the heuristic algorithm when the makespan is used as objective function. Heuristic 5 is the second most important heuristic, although its impact is only a fraction of that of heuristic 1. The importance of heuristic 4, 6, 3 and 2 can be concluded to be in that order for this particular set of instances. It is important to note that the instance set used is based on the case in the DSM enzyme manufacturing line. Instance sets with different features could be assumed to lead to a different importance per heuristic.

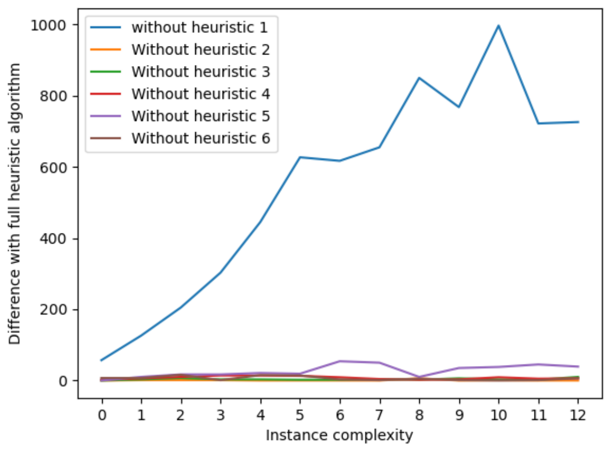

Figure 4: Differences between the lowest makespans found when running the heuristic algorithm with certain heuristics turned off. The graph includes all heuristics. The makespan found by the algorithm with all heuristics turned on is used as zero value. The objective function used is the makespan.

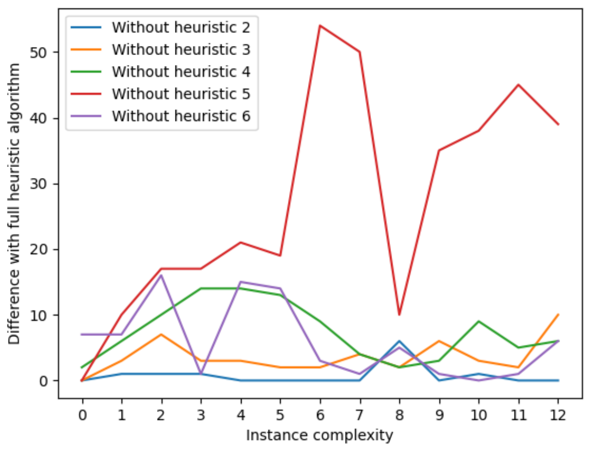

Figure 5: Differences between the lowest makespans found on average per instance group when running the heuristic algorithm with certain heuristics turned off. The graph excludes the first heuristic. The makespan found by the algorithm with all heuristics turned on is used as zero value. The objective function used is the makespan.

The aim of the fourth experiment is to examine the result of changing the objective function used on the impact of the heuristics on the quality of the created schedules by the heuristic algorithm. This experiment has, like the previous experiment, been ran on Set 3 of instances. The method used for this experiment follows the method used for the previous experiment, with the only difference being the objective function used. In this case the objective function which was used was the total lateness across all jobs.

In figure 6 we see that, as in Figure 4, the importance of heuristic 1 is still bigger than that of the other heuristics by another margin. We see that heuristic 7, which is specifically aimed to decrease lateness has a big impact in this case, which is to be expected. A surprising result is that the impact of heuristic 6 has increased drastically compared to its impact when the objective function used is the makespan. The importance of heuristic 6 for this objective function is comparable to that of heuristic 7. We can also note that the impact of heuristic 2, 3, 4 and 5 is minimal and that excluding any of these heuristics only leads to a minor decrease in schedule quality.

From this experiment we can conclude that when using lateness as an objective function heuristic 1 is the most important heuristic for the heuristic algorithm. Heuristic 6 and 7 have a roughly equal impact which is much lower than that of heuristic 1, but much higher than that of all the other heuristics. The other 4 heuristics only have a minor impact on the quality of the created schedules. We can also conclude that although the importance for most of the heuristics stayed the same when using two different objective functions, there were two heuristics which had a much higher impact when using lateness as an objective function compared to using the makespan.

5 Responsible Research

One way in which we have ensured that research was done responsibly, was by not strictly aiming for a positive result. The goal was therefore to determine whether a heuristic algorithm had any merit for this problem, rather than to create a successful heuristic algorithm for it. Even a negative result, where for example the heuristic algorithm would not perform well or be able to create good solutions to the FJSP variations with a short runtime, would have been useful in this regard.

Very important when performing responsible research is reproducibility. Research has to be reproducible in order to be able to confirm results and to be able to minimize the chance of misconduct, whether intentional or unintentional.

In order to ensure that this research is reproducible the pseudocode used for the algorithm has been shown and explained. The machines on which all experiments have been run have been described, to allow for similar experiments to be ran. For running the exact MILP solver the Gurobi package has been used, which is publicly available and thus good for reproducability.

Data about the instance sets used to run experiments on has also been shared, such that similar instances can be used to reproduce the experiments.

The inclusion of as much data and information about the methods and set-ups for the experiments has thus been given, all with the aim of allowing the experiments from this paper to be correctly replicated and the results to be correctly interpreted.

Data from all the experiments ran has been shared, without any distinguishment made based on the potential meaning or implications of these results. Even without malicious intent, leaving out data of certain experiments can lead to unexpected changes or the wrong conclusions being taken.

No outside data from other sources has been used, so po-

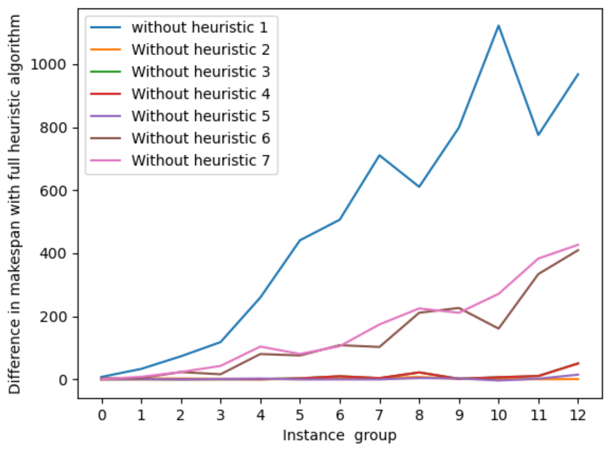

Figure 6: Differences between the lowest makespans found on average per instance group when running the heuristic algorithm with certain heuristics turned off when the objective value used is the lateness across all jobs. The makespan found by the algorithm with all heuristics turned on is used as zero value.

tential issues with secondary data are not applicable.

6 Discussion

The experiments done have shown that the proposed heuristic algorithm is capable of producing good feasible results to the FJSP variation with changeovers with a runtime much lower than that of an exact algorithm when the makespan is used as objective function.

For the instances with a smaller size, the MILP solver managed to find optimal solutions within its runtime limit, while the heuristic algorithm could not. Because of this, the MILP solver slightly outperformed the heuristic algorithm for these instance. We can explain this with the fact that the heuristic algorithm does not attempt to find an optimal solution, but rather a good feasible solution based on the chosen heuristics and it therefore makes sense that the MILP, when it does find the optimal solutions, outperforms the heuristic algorithm.

For the larger instances we see that the heuristic algorithm outperforms the MILP solver with a big margin, even with a much lower runtime. The MILP solvers main goal is to find an optimal solution to the problem, while the heuristic algorithm simply wants to find a good and feasible solution. Because of this the big difference can be explained by the MILP solver being working towards a final optimal solution, and when this cannot be reached within the runtime limit used in the experiments, it will not have solutions of a high quality, since the creation of solutions of as high quality as possible within a certain limit is not the main of this algorithm. The heuristic algorithm on the other hand has this exactly as it's aim, which could likely explain the much better performance on the larger instances.

The hyper-parameter finetuning process could be improved by changing the values for all the hyper-parameters collectively in all possible directions and experimentally determining which combination of hyper-parameters leads to the highest possible schedule quality. This was not feasible in this case, which is why instead the method of determining each hyper-parameter individually was chosen.

A shortcoming of the chosen algorithm is that it only considers the first operation of every job, until all of those first operations have been scheduled. This is something that in the instances used in this paper did not lead to issues, due to all jobs in the DSM case having between 1 and 3 operations. In cases where the differences in amount of operations per job is much larger this could however lead to issues, especially when using lateness as an objective function.

A possible extension of the algorithm could be to consider all currently possible operations. Initially this would be the first operation of every job, but once an operation is scheduled, the algorithm would then include the next operation of this job, if one is available, in the set of operations to consider for the next iteration of the loop.

7 Conclusions and Future Work

The main aim of this paper was to propose a heuristic algorithm for the FJSP variation with changeover times and determine how it performs compared to an existing exact solver for the same problem.

The algorithms have been compared on a set of instances with varying sizes, and we can conclude that with a fraction of the time, the heuristic algorithm manages to create solutions close in size to the optimal solutions created for the smaller and less complex instances, while creating solution of significantly higher quality then the exact solver for the larger, more complex, and more realistically sized instances.

Furthermore, the aim for this paper was to determine how certain heuristics perform compared to others and which heuristics had the highest impact on the result of our algorithm when using the makespan of schedules as objective function. By far the most essential heuristic is the one that computes the expected finish time of a certain operation on a certain machine before scheduling this operation machine combination. The heuristics that take into account the idle times of machines and the wait times between multiple operations of a job had only minor impacts on the final results. The heuristics calculating the changeover times and the expected total time of a job had average impacts on the decisions made by the algorithm.

Another aim for this paper was to examine how the comparative performance of the heuristics changed when changing the objective function used. We have showed that when using lateness as an objective function the heuristic computing the expected finish time of an operation and machine combination is still the most essential. Other heuristics that have a major impact on the final schedule quality for this objective function are the heuristic calculating the expected changeover time and the heuristic taking into account the due date of a job.

The heuristic algorithm created and the comparison between the heuristics should be helpful when attempting to create a meta-heuristic solution for this variation on the FJSP in the future. The heuristic algorithm as a whole could potentially be used either to initialize a schedule that can then be improved upon further using a meta-heuristic algorithm. Some or all of the heuristics described could also be used inside meta-heuristic algorithms to judge qualities of in-between solutions or rank certain options for potential changes to a schedule.

For use-cases where the aim is not simply to get a schedule of as high of a quality as possible, but where the time to create such a schedule is highly limited, a heuristic algorithm could be of great value for this variation on the FJSP and certainly has merit. Further research could prove worthwhile.

A suggestion for future work to improve on the heuristic algorithm proposed in this paper is to consider all operations currently possible, rather than only sequentially considering a certain operation of each job until all of those have been scheduled. This could especially be important in cases where there is a high variance between the amount of operations per job, which is not the case for the DSM enzyme production plant.

Another suggestion would be to examine the performance of the heuristics for a wider range of objective functions, as well as also comparing the performance of the heuristic algorithm to that of an exact algorithm for different objective functions.

References

[1] M. R. Garey, D. S. Johnson, and R. Sethi, "Complexity of flowshop and jobshop scheduling." *Mathematics of Operations Research*, vol. 1, pp. 117–129, 1976.

[2] P. Brucker and R. Schlie, "Computing job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, pp. 369–375, 1990.

[3] M. Ziaee, "A heuristic algorithm for solving flexible job shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 71, pp. 519–528, 3 2014.

[4] K. van den Houten, "Algorithms for smart scheduling of a dsm enzyme production line problem formulation," 2022. [Online]. Available: https://gitlab.ewi.tudelft.nl/kvandenhouten/cse3000-research-project-dsm-smart-scheduling/-/blob/master/documents/problem_formulation.pdf

[5] H. M. Wagner, "An integer linear-programming model for machine scheduling," *Naval Research Logistics Quarterly*, vol. 6, pp. 131–140, 6 1959.

[6] I. A. Chaudhry and A. A. Khan, "A research survey: Review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, pp. 551–591, 5 2016.

[7] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, vol. 1, pp. 67–77, 2019.

[8] P. Fattahi, M. S. Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 18, pp. 331–342, 6 2007.

[9] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers and Operations Research*, vol. 35, pp. 2892–2907, 9 2008.

[10] R. Buddala and S. S. Mahapatra, "An integrated approach for scheduling flexible job-shop using teaching-learning-based optimization method," *Journal of Industrial Engineering International*, vol. 15, 2018. [Online]. Available: https://doi.org/10.1007/s40092-018-0280-8

[11] Y. N. Sotskov and O. Gholami, "Mixed graph model and algorithms for parallel-machine job-shop scheduling problems," *International Journal of Production Research*, vol. 55, pp. 1549–1564, 2017.

[12] M. A. Ortíz, L. E. Betancourt, K. P. Negrete, F. D. Felice, and A. Petrillo, "Dispatching algorithm for production programming of flexible job-shop systems in the smart factory industry," *Ann Oper Res*, vol. 264, pp. 409–433, 2018.

[13] M. A. F. Pérez and F. M. P. Raupp, "A newton-based heuristic algorithm for multi-objective flexible job-shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 27, pp. 409–416, 2016.

[14] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022. [Online]. Available: https://www.gurobi.com