Flexible Job Shop Optimization with Simulated Annealing

M.C. Bak
Supervisors: K.C. van den Houten, M.M. 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

Abstract

In this paper, a Simulated Annealing (SA) implementation for a Flexible Job Shop Problem (FJSP), with change-over time, is presented. This implementation is compared to a Mixed Integer Linear Programming (MILP) optimization, to compare performances. The SA algorithm starts with creating the first schedule with Global Selection. The neighbourhood is created with an application of k-insertion. Annealing is implemented with exponential cooling. The SA implementation does not consistently outperform the provided MILP implementation. However, the run-time of the simulated annealing is shorter than the MILP. The algorithm is then also used to discover bottlenecks in the production line presented with the FJSP instances.

1 Introduction

In process industries, industries where the production is continuous or occurs on batches of materials, have many factories and production lines to produce their products. To make sure these factories and production lines work as efficient as possible, a lot of effort is put into scheduling the orders for production.

This research looks at one production line of DSM, a biotechnology company. This production line can be represented as a so-called Flexible Job Shop Scheduling Problem (FJSP). An FJSP consists of n jobs, all consisting of some operations. These operations can run on a set of m machines. These machines differ for each operation [1]. All the machines for one operation have the same processing time. The machines in the production line need cleaning between certain jobs.

The goal of this research is to find a suitable optimization method for this scheduling problem so that all process industries can schedule faster and work more efficiently. The second goal is to find bottlenecks in the DSM production line.

To test the effectiveness and efficiency of the implementation, a Mixed Integer Linear Programming (MILP) implementation is provided. The MILP is a suitable approach to solve an FJSP [2, 3].

Besides MILP, meta-heuristic approaches are also suitable optimization methods for the FJSP [4–6]. Meta-heuristic algorithms solve a problem by creating local search spaces and applying a heuristic to select a better solution than before. Simulated Annealing (SA) is a meta-heuristic approach derived from the annealing of metal. This method has been applied to multiple different Job shop problems before, making it a suitable algorithm to use for this extended FJSP [4,5,7–9]

This paper investigates in what way SA is better suited to solve the FJSP than the MILP implementation. This is done by first establishing a baseline of the performance of the MILP. The baseline was established in cooperation with a group of other students, who worked on implementing different optimizations for this FJSP. After the baseline is established, the application of SA to the FSJP with change-over times is reviewed. Then the performance of the SA is compared to the performance of the MILP. For the last part of the research, bottlenecks in the DSM production line will be examined, from where the recommendations for improving the production line with one machine will be given.

The results of the computer experiments show that the SA implementation and the MILP are useful for the FJSP in their specific ways.

This paper consists of six parts, starting with an extended background of the FJSP and the SA optimization, followed by an explanation of the SA approach for this project. Then the Experimental setup and results will be discussed. After that, there will be a short part on responsible research, followed by a discussion on this research. Lastly, there will be a conclusion and insights for future improvements that can be made.

2 Background

To make sure that the process industries can run in the most efficient way possible, every production line has a scheduling department that attempts to tackle the complex scheduling question for their production line.

A company that is part of the process industry is DSM. For this research, a production line of DSM is used as a case study. In this specific plant, the last three operations of the downstream process are the part where the most gain of advanced optimization can be achieved.

These operations are the preparation, filtering and reception of the enzymes. The batches should be scheduled in such a way that the make-span of the enzymes is minimized.

This optimization problem can be described as an extended flexible job shop scheduling problem (FJSP). This research aims to find an optimization for this problem that finds a good balance in run time and quality of results so that DSM can decrease the makespan of the last phase of this production plant.

2.1 Flexible Job Shop Problem

The FJSP is an extension of the classical Job Shop Scheduling Problem (JSP). The classical job shop problem consists of several jobs that are to be processed by specified machines. All jobs have to follow specific operations. In the JSP, every operation has a specified machine that can process the operation. The JSP is NP-hard [10], meaning that it is not feasible to solve a JSP optimally in polynomial time. The difference between the JSP and the FJSP is that in a FJSP operations can be colpleted by multiple machines. To optimize the FJSP, it means that for every possible sequence of operations, all possible machine assignments also should be checked to find the optimal. This makes the FJSP more complex than the JSP, meaning that the FJSP is also NP-Hard. [6]. This special FJSP has change-over times between some jobs, meaning that between some jobs, the machines have to be prepared for the new job. During this preparation, the machines are not available for production.

2.2 Simulated Annealing

The optimization method used to optimize the FJSP is Simulated Annealing (SA) with multi-start. Multi-start is a technique where an algorithm runs more than once, after which

the best solution is selected [11–13]. For SA this is a favourable addition because of the short run-time and the randomness embedded in the algorithm. Here, the literature on existing techniques for SA applied to an FSJP will be discussed and explained shortly.

SA is a meta-heuristic approach used for global search optimization. The meta-heuristic is based on the annealing of metals in nature. This is used to avoid getting trapped in bad local minima [14–16]. SA is used for optimizing FJSP, JSP [7, 8], and other variants of the JSP like the Flow-Shop Problem [9, 10, 17] in the past. The meta-heuristic of SA makes that every schedule worse than the current optimal schedule has a probabilistic chance of still being chosen. This enables the optimization to jump from one optimum to another, which may be the global optimum or a better local optimum. The algorithm has a starting temperature, that cools more the longer the algorithm runs. This cooling decreases the probabilistic chance of a lesser schedule being chosen. SA consist of three components that can be implemented in various ways, initialization, neighbourhoods and the annealing schedules.

Initialization

The initialization method used for this implementation of SA is a method called Global Selection (GS). GS is a method that is used for Genetic Algorithm (GA) optimizations and Particle Swarm Optimizations (PSO) for the FJSP on multiple occasions [18–20]. GS generates a schedule by randomly choosing a job. The operations of that job are scheduled on the machine that has the shortest processing time. This repeats until all jobs are scheduled. The resulting schedule then is used to generate a neighbourhood.

Neighbourhood

To create the neighbourhood, a graph is used to make small changes to the schedules. A graph representation is used to represent FSJP on multiple occasions [21–23]. For this algorithm, the graph is used to create a neighbourhood from a schedule. To get new schedules, operations on the longest path are inserted into a different machine sequence using k-insertion [21, 22].

Annealing schedules

The meta-heuristic of annealing is based on a cooling scheme in the algorithm. This cooling presents a probabilistic chance to select a less optimal schedule to escape unfavourable local optima. This chance is based on the following formula: $e^{-\delta/T}$ [24] where δ is the difference in make-span between two schedules and T is the so-called temperature of the algorithm. The start temperature is experimentally determined. This temperature decreases while the algorithm runs. Due to this cooling, the change of a less optimal solution being chosen decreases the longer the algorithm runs [14, 16, 25].

3 Simulated Annealing algorithm

The SA implementation for this optimization problem follows the flowchart presented in Figure 1.

Figure 1 shows the three most important aspects of the implementation identified. First, the initialization of the first schedule is discussed. Then the creation of the neighbourhood, which presents a new search space, is explained. Lastly, different cooling methods will be compared.

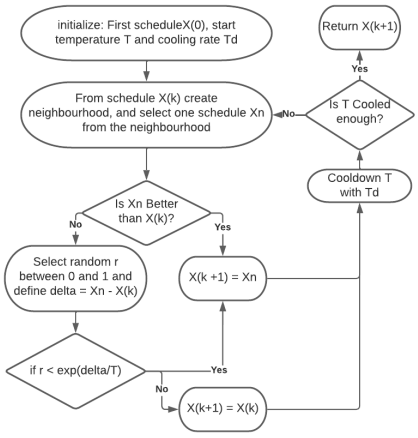

Figure 1: Flow-chart representing the simulated annealing algorithm

3.1 Initialization with Global Selection

For the initialization of the first schedule, Global Selection (GS) is used. GS creates a schedule by planning jobs and all corresponding operations sequentially. The order in which the jobs are scheduled is determined at random. When a job is selected, all operations are scheduled on the machine with the lowest completion time. To keep track of the lowest completion time, a time array is created, where for each machine, the processing times of the assigned operations are added to. The machine with the lowest value in the time array is finished the earliest, and thus the operation is scheduled on that machine. Figure 2 illustrates a step by step explanation of GS [26]

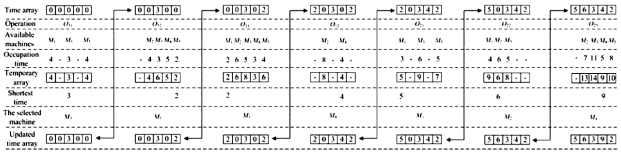

Figure 2: An illustrated execution of Global selection [26].

The pseudocode for the GS depicted in Figure 2 is shown in Algorithm 1

For the representation of the created schedule, vectors $v1$ and $v2$ are used. The first vector, $v1$ is a vector where each index represents an operation. The value assigned to the operation in the vector is the machine on which the operation is executed. An example vector $v1$ is shown in Figure 3

Position: r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Operation Indicated	$o_{1,1}$	$o_{1,2}$	$o_{1,3}$	$o_{1,4}$	$o_{2,1}$	$o_{2,2}$	$o_{2,3}$	$o_{2,4}$	$o_{3,1}$	$o_{3,2}$	$o_{3,3}$	$o_{3,4}$	$o_{4,1}$	$o_{4,2}$	$o_{4,3}$	$o_{4,4}$
Machine Assignment: $v_1(r)$	4	3	3	1	2	4	1	4	3	1	2	1	2	4	4	3

Figure 3: Machine assignment vector [21].

Algorithm 1 Pseudocode for global selection

time ← array with length M and values 0
v_1 ← array with length of all operations
jobs ← array with all jobs
while $jobs \neq empty$ **do**
  job ← random job from jobs
  remove job from jobs
  for all operations op of job **do**
    temp ← time
    machines ← machines that can process op
    for machine m in machines **do**
      temp[m] ← processing- and changeover time of op on m
    end for
    min_t ← min value of temp
    $I_{j,o}$ ← index of min_t
    $v_1[I_{j,o}]$ += min_t
  end for
end while

The indices of vector $v1$ are numbered from zero to the number of operation of operations. Making it more difficult to keep track of which job and operation a certain index points to. To solve that vector $v2$ is created.

$v2$ contains the numbering of operations for all jobs. This means that if there is an instance with two jobs with both three operations $v2$ looks like this: $[0, 1, 2, 0, 1, 2]$ With this representation, the job and operation of index i are found with Algorithm 2

Algorithm 2 Get job and operation from $v2$

v_2 ← array with length of all operation
i ← index of which job and operation are to be found
$job \leftarrow -1$
for x in range of length of $v2$ **do**
  if $v2[x]$ equals 0 **then** job += 1
  end if
  if x equals i **then** break out of the loop
  end if
end for
$op \leftarrow v_2[i]$
return job, op

In this algorithm, the job is initialized as -1, because the first job always is job zero. A job always starts with operation zero. Due to this, if $v_2[x]$ equals zero, it is certainly the first operation of a new job.

The following section, explains how vectors v_1 and v_2 are used to generate a graph representation of a schedule.

3.2 Neighbourhood generation

The schedule, represented as vectors, will be rewritten to a graph representation for creating the neighbourhood. This graph represents the schedule. A useful property of these graphs is that the length of the longest path is equal to the makespan of the schedule. The neighbourhood will consist of schedules where operations of this critical path are moved to another machine. Neighbourhood creation through graphs exists of the creation of the graph, moving operations of the graph, creating a neighbourhood from moving operations and finding the critical path and makespan. These parts will be explained in the next sections.

Graph initialization

The vectors that are returned by the initialization method can be converted as a directed graph $G = (N, E)$ where $E = (E_d \cup E_j \cup E_m)$, these sets are defined later in this section. In the graph, N are the nodes containing a start- and end node, together with a node for each operation of each job $n_{j,o}$ where j is the job and o is the operation. The graph representation uses weighted nodes, where the weight is the processing time of operation $n_{j,o}$ on the assigned machine. E_d are the so-called dummy edges, which go from the start node to all first operations of a job, and from the last operation of a job to the end node. Figure 4 represents these edges.

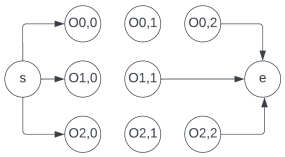

Figure 4: Graph with dummy edges

The edges in set E_j show the precedence relations between operations in a job. These edges span between nodes $N_{j,i}$ and $N_{j,i+1}$. A representation is shown in Figure 5.

Figure 5: Graph with job edges

E_m edges represent the precedence relations of the machine order, where they connect the operations that use the same machine in the same fashion as E_j does for the jobs. For these edges, the changeover times are added as weight. Figure 6 shows the machine edges.

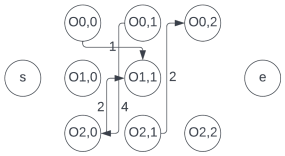

Figure 6: Graph with machine edges

Figure 7 displays a graph with all edges. As is shown in this figure, there are no cycles in this graph. For every feasible

graph, there are no cycles. This is because of the strict order of the operations.

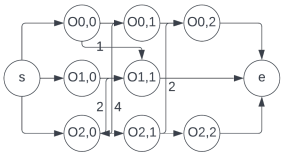

Figure 7: Graph with all edges

Determine makespan and critical path

The graph would be traversed to get the critical path, needed for the neighbourhood creation. Since the critical path is the path that takes the longest to complete, operations on this path are moved to create the neighbourhood, as described in the coming sections. The critical path is found by performing a Breath First Search (BFS) on the graph. Because BFS traverses the complete graph, the time complexity to find the critical path is polynomial, because, for every node N, there are E edges to be visited, resulting in a $\mathcal{O}(N * E)$ time complexity. This makes determining the critical path one of the most expensive operations of the algorithm. The length of this critical path is equal to the makespan of the schedule represented by the graph.

Moving operations

To be able to create a neighbourhood from the graph, an operation of the critical path is inserted on another given machine for that operation. Such an insertion of an operation $v \in \mathcal{O}$ on machine $k \in M_v$ is called a k-insertion. When performing a k-insertion on graph G, the machine edges from- and to the node of the operation should be removed from the graph, resulting in a new graph G^-. After that, v is inserted in the processing order of k. For such an insertion to be feasible, the resulting graph should be acyclic. By definition G is acyclic, meaning that G^- is also acyclic, because only edges are removed, and the removal of edges cannot create any cycles within a graph. If new edges are in the correct processing order, again there are no cycles, because every operation can only be visited once.

For the insertion of v in the operation sequence of k the following process is used to ensure feasible insertions. First, Q_k is defined as the set containing the operations processed by machine k, ordered at starting time. From Q_k two more sets are derived, R_k and L_k.

Here $R_k = (x \in Q_k | s_x + p_x > s_v^-)$ and $L_k = (x \in Q_k | p_x + t_x > t_v^-)$.

Figure 8 and 9 visualize R_k and, L_k respectively.

Figure 8 shows that R_k is the set of operations where there is no path from any $X_{j,o} \in R_k$ to v, and also there is no path from v to $Q_k \backslash R_k$.

As depicted in Figure 9, L_k is the set of operations where there is no path from v to any $X_{j,o} \in L_k$ and also there is no path from any $Q_k \backslash L_k$ to v

A neighbourhood F_{vk} where the solutions are obtained by inserting v after the operations of $L_k \backslash R_k$ and before every operation of $R_k \backslash L_k$ is a set of feasible solutions [22].

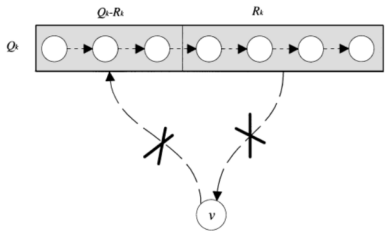

Figure 8: Properties of R_k [22]

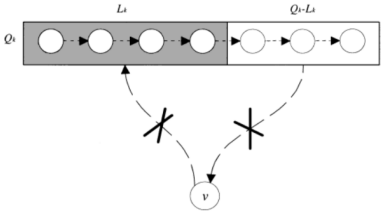

Figure 9: Properties of L_k [22]

Derive neighbourhood

The critical path is used to get a neighbourhood from a single k-insertion. For every operation o on the critical path, a k-insertion is done of all machines of o. This results in a neighbourhood where all schedules differ at most one operation from the parenting schedule. From this neighbourhood, one schedule is selected at random. This schedule is compared to the parenting schedule, to determine whether it is selected. This is done in combination with the cooling method described next.

3.3 Cooling method

The choice of annealing method is important for getting good results [27]. Different ways of cooling are used in other applications of SA. Such annealing schedules include linear cooling, exponential cooling and logarithmic cooling.

Linear cooling is of the form $T(t) = T_0 - \alpha t$ where t is step-size and α is a constant number that is experimentally found.

For the exponential cooling $T(t) = T_0 * \alpha^t$ where t is step-size and α is a constant factor $(0 < \alpha < 1)$.

Lastly the logarithmic cooling with the equation $T(t) = c/(log(t + d))$ with c being the largest energy barrier, and d equal to one most of the time [27]

For this implementation, an exponential cooling system is used. The main advantage of exponential cooling is that at the start of the algorithm, the system cools faster than at the end. As the temperature decreases, it starts to cool slower. This makes that the more the optimum is reached, the more iterations are done to try and improve the result.

The cooling method makes the makespan converge to a global minimum the more the temperature cools, as shown in Figure 10. In this figure, the mean makespan during the iteration is shown for multiple runs. This results in a converging graph to a minimum makespan.

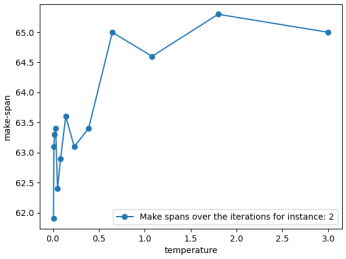

Figure 10: Convergence of makespan plotted against temperature

4 Experimental Setup and Results

The SA algorithm was implemented and tested on an HP ZBook Studio G5 with an Intel® Core™ i7-8750H CPU and 16 GB RAM. The SA algorithm is implemented in Python. The MILP runs on the same computer, with the help of Gurobi optimizer. The instances that are used on the algorithm are on the project's GitHub repository [1]

4.1 Experimental hyperparameter tuning

To get the best results from the SA, some hyperparameters are experimentally tuned. The tuned parameters are the start temperature, how often the algorithm runs for the multi-start and the variable α of the cooling system. To justify the chosen parameters, multiple tests were run. For these experiments, a higher temperature and a higher multi-start are expected to give better results. A high starting temperature means more iterations, meaning more schedules are compared. The hypothesis about multi-start mainly concerns the randomness implemented in the algorithm, making it not deterministic. Due to that, a single run might have an unfavourable first schedule, whereas a second run can have a close to the optimal starting point. Having more starts means that high outliers will become relatively rare, thus improving the results.

The first experiments are conducted to determine the best starting temperature. For the first run, temperatures 1, 10, 100, and 1000, are used. Figure 11 shows the results of this test.

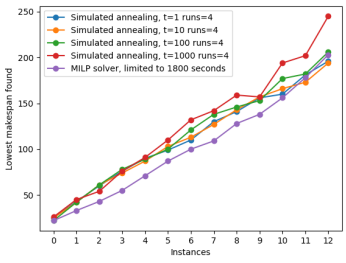

Figure 11: Simulated annealing algorithm run for temperatures 1, 10, 100, 1000

From this graph, the differences between temperatures 1, 10 and 100 are not visible. Because of this, a new test is created with temperatures 10, 25, 50, and 75, displayed in Figure 12

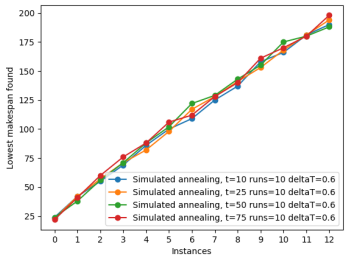

Figure 12: Simulated annealing algorithm run for temperatures 10, 25, 50, 75

The results from this experiment are almost identical to each other, meaning that the performance of $T_0 = 75$ is not significantly better than $T_0 = 25$. A T_0 of 25 is preferable, since a lower starting temperature means a lower amount of iterations, resulting in a shorter runtime.

For the amount of runs for the multi-start the first experiment was run for 1, 5, 10 runs, shown in figure13

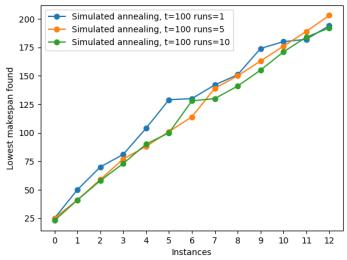

Figure 13: Simulated annealing algorithm run for 1, 5 and 10 runs

The main difference between the results of this experiment is the stability results. Since the lowest make-span of all runs is selected, the more runs are done, the less chance there is for an accidental outlier with a high makespan. To see the results of more runs, an experiment with 5, 10 and 25 runs is conducted and shown in Figure 14

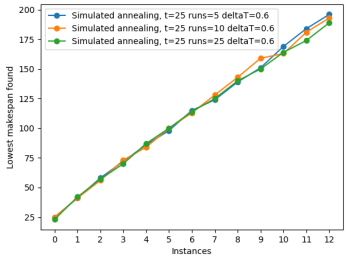

Figure 14: Simulated annealing algorithm run for 5, 10 and 25 runs

From this graph, the same conclusions can be drawn. In this graph, the later instances also have a slightly lower

[1] https://github.com/mcbak/rp_dsm

makespan for the highest amount of runs. This however comes at the cost of a higher run-time.

There are a few tests run to find the best α for the cooling of the algorithm. The results can be found in Figure 15

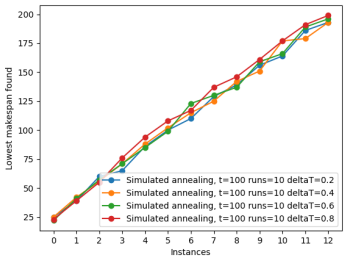

Figure 15: Performance of different cooling rates α

The results are so close to each other that there is no significant difference in resulting makespans. Here also, the same is true for the runtime of a lower α for the annealing schedule.

The following hyperparameters are used for the rest of the testing: A starting temperature of 25, multi-start with 10 runs and an α of 0.4.

4.2 Simulated Annealing compared to MILP

For the comparison of the SA against the MILP, 20 test instances are used. These test cases are ordered from smallest to biggest. Since the MILP finds the global optimum, given enough time, it is limited to 1800 seconds. This means that every instance runs for this time, even if the optimum is already found. However, if no feasible solution is found in that time, no makespan is returned. The SA, as explained, converges to a minimum temperature, after which it stops and returns the last schedule. Because of that, the runtime of the SA varies per instance.

The results of running the MILP and the SA implementation to 20 test instances are found in Figure 16.

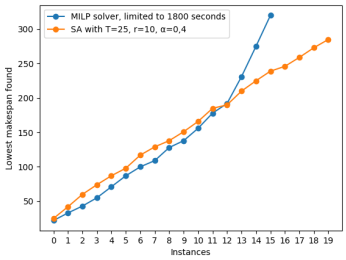

Figure 16: Make-spans of SA compared to MILP for 20 instances.

The first thing concluded from this graph is that from instance 15 on, the MILP was not even able to find a schedule in 1800 seconds. When comparing the MILP with the SA, the graph shows that from instance 0 to instance 12, the MILP performs better than the SA. From instances 13 to 19, the SA outperforms the MILP.

Results SA with T=25, r=10, α=0,4 and MILP limited to 1800s			
	Make-span		
Instance	Results MILP	Results SA	Difference
0	22	25	-14%
1	33	42	-27%
2	43	60	-40%
3	55	74	-35%
4	71	87	-23%
5	87	98	-13%
6	100	117	-17%
7	109	129	-18%
8	128	138	-8%
9	138	151	-9%
10	156	166	-6%
11	178	185	-4%
12	192	190	1%
13	231	210	9%
14	275	225	18%
15	320	239	25%
16	Not Found	246	100%
17	Not Found	259	100%
18	Not Found	273	100%
19	Not Found	285	100%

Figure 17: Difference between the results of the MILP and the SA.

The percentage difference between makespans of the MILP and SA is shown in Figure 17.

When looking at the differences, it becomes clear that for the first instances, the MILP performs better, to a point where the makespan of the MILP is 40% lower than the one of the SA. After instance 12, the SA starts to perform better, while the MILP cannot find a feasible solution in the given time.

In Figure 18, the exact makespans and runtimes of the MILP and SA are listed.

Results SA with T = 25, r = 10, α = 0,4 and MILP limited to 1800s				
	Results MILP		Results SA	
Instance	Make-span	Run-time (s)	Make-span	Run-time (s)
0	22	1800	25	3,8
1	33	1800	42	5,7
2	43	1800	60	8,9
3	55	1800	74	11,1
4	71	1800	87	14,9
5	87	1800	98	20
6	100	1800	117	23,8
7	109	1800	129	29,4
8	128	1800	138	36,7
9	138	1800	151	44,1
10	156	1800	166	52,3
11	178	1800	185	63,2
12	192	1800	190	72,5
13	231	1800	210	89,2
14	275	1800	225	109,1
15	320	1800	239	124,3
16	Not Found	1800	246	139,5
17	Not Found	1800	259	162,3
18	Not Found	1800	273	191,4
19	Not Found	1800	285	224,3

Figure 18: Results MILP and SA, with run-time.

When comparing the runtimes of the SA against the runtimes of the MILP, the SA performs significantly better. Again, the bigger instances show a big difference in performance. The last four instances could not be solved by the MILP, where the SA completed them within 225 seconds.

4.3 Bottleneck testing

Since the last aim of the paper was to identify possible bottlenecks in the production line, a few tests are conducted to find out how much an extra machine could decrease the makespan for the production of big batches. To identify potential bottlenecks, three instance sets were created. Each set has an operation with an extra machine compared to the original instance

set. This allows examining for which operation an extra machine makes the most impact on the makespan. When adding a machine, the change-over times were set equal to the worst changeover of the existing machines.

When looking at the current division of machines over the operations, it can be expected that an extra machine for operation 3, would give the best improvements. This is because operations one and two have more machines that can be used. Next to this, of the six different enzymes that can be produced, five enzymes need operation 3, whereas only four need operation 1. Operation 2 is needed by all enzymes. Knowing this, it is likely that after operation 3, the best improvement comes from adding a machine for operation 2 and that adding a machine for operation 1 leads to the smallest improvement.

The results of the experiments can be found in Figures 19, 20 and 21. In these tables, the results of the MILP and SA are compared to each other. The decrease in makespan is displayed as percentages, to make it equal for all instances. To make sure that the

Results extra machine on operation 3					
Original	extra machine		Original	extra machine	
MILP-900	MILP-900	Improved	SA	SA	Improved
22	18	18%	25	19	24%
33	24	27%	42	36	14%
43	32	26%	57	50	12%
56	43	23%	71	60	15%
72	53	26%	86	77	10%
87	67	23%	99	86	13%
100	76	24%	112	97	13%
109	91	17%	129	108	16%
128	102	20%	141	115	18%
138	116	16%	156	127	19%
156	135	13%	169	139	18%
178	156	12%	176	155	12%
200	180	10%	191	159	17%
Average Improvement		20%	Average improvement		16%

Figure 19: Comparison of the original results against the same instances with an extra machine for operation 3

The results of adding a machine for operation 3 are displayed in Figure 19. These results show an average decrease in makespan of 20% for the MILP and a decrease of 16% for the SA. As expected, the addition of a machine to operation 3 gives a good improvement for the makespan.

Results extra machine on operation 2					
Original	extra machine		Original	extra machine	
MILP-900	MILP-900	Improved	SA	SA	Improved
22	22	0%	25	25	0%
33	32	3%	42	40	5%
43	43	0%	57	52	9%
56	55	2%	71	70	1%
72	68	6%	86	83	3%
87	85	2%	99	98	1%
100	97	3%	112	111	1%
109	110	-1%	129	123	5%
128	126	2%	141	137	3%
138	139	-1%	156	151	3%
156	178	-14%	169	163	4%
178	186	-4%	176	176	0%
200	209	-5%	191	196	-3%
Average Improvement		-1%	Average improvement		2%

Figure 20: Comparison of the original results against the same instances with an extra machine for operation 2

Figures 20 and 21 show us the results of adding a machine to operation 1 and 2. These results hardly show a decrease in makespan, where the MILP with an extra machine for operation 2 performs worse than the original. Since these improvements are less than 5% on average, it is not viable to speak of

Results extra machine on operation 1					
Original	extra machine		Original	extra machine	
MILP-900	MILP-900	Improved	SA	SA	Improved
22	22	0%	25	24	4%
33	33	0%	42	38	10%
43	43	0%	57	56	2%
56	54	4%	71	67	6%
72	68	6%	86	86	0%
87	81	7%	99	95	4%
100	96	4%	112	109	3%
109	113	-4%	129	125	3%
128	128	0%	141	136	4%
138	141	-2%	156	144	8%
156	154	1%	169	161	5%
178	208	-17%	176	172	2%
200	193	4%	191	185	3%
Average Improvement		0%	Average improvement		4%

Figure 21: Comparison of the original results against the same instances with an extra machine for operation 1

improvement.

5 Responsible Research

To ensure that this research is up to the standards of modern research, the next sections are a reflection on research data processing and reproducibility of the research.

5.1 Research data processing

To ensure that there was no misconduct in the data gathered in this research, all the results of the results are published on a public GitHub repository[2]. The data presented in this paper is selected in such a way that all measured variables and outcomes are represented in the results. The results that are not in the paper, but are in the repository, are results that underline the shown results but present no new information. Graphs that may not be included are for example small test sets which have big deviations, or sets with older results.

5.2 Reproducibility

To ensure the reproducibility of this project, this paper includes extensive explanations of the developed algorithm and tests concerning the hyperparameter tuning. Some of these parts also include pseudocode, which can easily be implemented. For tuning the hyperparameters and getting the results, all the variables are defined in the paper. This ensures that all experiments can be redone precisely. For the MILP, the used optimizer is mentioned, together with the used time constraints. The used MILP representation is also present on the GitHub repository. Also, the PC and processor used for testing are specified. In this way, potential differences due to the computational power of the machine can also be identified more easily.

6 Discussion

As seen in section 4, the MILP outperforms the SA for the lower instances. For bigger instances, the SA outperforms the MILP in terms of make-span and runtime. This is explained because the MILP tries to find the global optimum, without optimizing for good intermediate results. For smaller instances, the MILP can find the optimum or gets close to the optimum within the allocated 1800 seconds, while for the bigger instances, it has to do too many operations to get near

[2] `https://github.com/mcbak/rp_dsm`

the optimum. The SA does not aim to find the global optimum but to get good results from exploring some parts of the whole search space. From the explored search spaces, the best result is selected most of the time.

By improving the initialization and k-insertion, it is likely to improve the performance of the SA.

Possible improvements in global selection are that the random selection of jobs in the GS can be replaced by an educated selection based on heuristics such as expected runtime, start- and completion times and machine availability [28].

The neighbourhood function also has room for improvement. To determine where to insert the operation in the new machine's sequence, the algorithm compares the start and completion times of all the jobs on that machine to the old start and completion times of the inserted operation. These start and completion times are now based on estimation. This could be improved slightly by finding the exact start and completion times of the operations, giving fewer insertions and thus a smaller and better neighbourhood.

Lastly, the results are achieved on a relatively small test set of 20 instances. To get more results, and better insights into the performance, bigger test sets should be used.

The improvement of adding a machine to operation three was expected. This is because operation thee had the least machines and was the second most used operation. The lack of improvement for adding a machine to either operation 1 or 2 was unexpected. The main reason probably is that the most delay is in operation 3, which in hindsight makes sense, seeing the improvement achieved there. For the MILP the falling performance might be because the extra complexity of the instance, with an extra machine, is bigger than the delay caused by that operation. This means that more time is spent trying all the different machines meaning that lesser schedules can be explored.

These results show that if there is a possibility to add one machine to any operation, it should be added to operation 3 to achieve the best improvement.

7 Conclusions and Future Work

In this paper, a Simulated Annealing (SA) optimization is developed for a Flexible Job Shop Scheduling Problem (FJSP), extended with change-over times between different jobs. The main goal of this research was to compare SA to Mixed Integer Linear Programming (MILP) optimization, which already existed. The idea behind the different part of the implementation is thoroughly discussed, and explained.

The performance of the SA is worse than the MILP implementation for small test instances. For the larger test instances, the SA has a slightly better performance.

It can be concluded that, for smaller instances, the MILP outperforms the SA, but for the larger instances, the SA has a better makespan. Next, the MILP had a run-time of 1800 seconds, while the SA finished within 225 seconds for the largest instance.

When improving the current production line used for the test set, adding a machine for the third operation gives the best results.

There are also some improvements that can be implemented to improve the performance of the SA. The first improvement is using heuristics for the initialization method instead of the random selection used now. In the k-insertion part, start and completion times of nodes are now estimated. Improvements can be made by implementing a better method to find the start and completion times.

References

[1] Y. Demir and S. Kürşat İşleyen, "Evaluation of mathematical models for flexible job-shop scheduling problems," *Applied Mathematical Modelling*, vol. 37, no. 3, pp. 977–988, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0307904X12001837

[2] C. Özgüven, L. Özbakır, and Y. Yavuz, "Mathematical models for job-shop scheduling problems with routing and process plan flexibility," *Applied Mathematical Modelling*, vol. 34, no. 6, pp. 1539–1548, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0307904X09002819

[3] H. M. Wagner, "An integer linear-programming model for machine scheduling," *Naval Research Logistics Quarterly*, vol. 6, no. 2, pp. 131–140, 1959. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800060205

[4] V. Roshanaei, A. Azab, and H. ElMaraghy, "Mathematical modelling and a meta-heuristic for flexible job shop scheduling," *International Journal of Production Research*, vol. 51, no. 20, pp. 6247–6274, 2013. [Online]. Available: https://doi.org/10.1080/00207543.2013.827806

[5] M. Yazdani, M. Zandieh, R. Tavakkoli-Moghaddam, and F. Jolai, "Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem," *Scientia Iranica*, vol. 22, no. 3, pp. 1242–1257, 6 2015. [Online]. Available: http://scientiairanica.sharif.edu/article_3715.html

[6] P. Fattahi, M. Saidi-Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 18, pp. 331–342, 4 2007.

[7] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job Shop Scheduling by Simulated Annealing," *Operations Research*, vol. 40, no. 1, pp. 113–125, 1992. [Online]. Available: https://doi.org/10.1287/opre.40.1.113

[8] S. G. Ponnambalam, N. Jawahar, and P. Aravindan, "A simulated annealing algorithm for job shop scheduling," *Production Planning & Control*, vol. 10, no. 8, pp. 767–777, 1999. [Online]. Available: https://doi.org/10.1080/095372899232597

[9] C. Low, "Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines," *Computers & Operations Research*, vol. 32, no. 8, pp. 2013–2025, 8 2005.

[10] T. Gonzalez and S. Sahni, "Flowshop and Jobshop Schedules: Complexity and Approximation," *Operations Research*, vol. 26, no. 1, pp. 36–52, 1978. [Online]. Available: https://doi.org/10.1287/opre.26.1.36

[11] S. W. Lin and K. C. Ying, "Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm," *Computers & Operations Research*, vol. 40, no. 6, pp. 1625–1647, 6 2013.

[12] V. F. Yu and S. W. Lin, "Multi-start simulated annealing heuristic for the location routing problem with simultaneous pickup and delivery," *Applied Soft Computing*, vol. 24, pp. 284–290, 11 2014.

[13] S. W. Lin, "Solving the team orienteering problem using effective multi-start simulated annealing," *Applied Soft Computing*, vol. 13, no. 2, pp. 1064–1073, 2 2013.

[14] N. M. Najid, S. Dauzere-Peres, and A. Zaidat, "A modified simulated annealing method for flexible job shop scheduling problem," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, 10 2002, pp. 6 pp. vol.5–.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671

[16] S. Delahaye Daniel
}and Chaimatanan and M. Marcel, "Simulated Annealing: From Basics to Applications," in *Handbook of Metaheuristics*, J.-Y. Gendreau Michel
}and Potvin, Ed. Cham: Springer International Publishing, 2019, pp. 1–35. [Online]. Available: https://doi.org/10.1007/978-3-319-91086-4_1

[17] I. Osman and C. Potts, "Simulated annealing for permutation flow-shop scheduling," *Omega*, vol. 17, no. 6, pp. 551–557, 1 1989.

[18] X.-L. Gu, M. Huang, and X. Liang, "A Discrete Particle Swarm Optimization Algorithm With Adaptive Inertia Weight for Solving Multiobjective Flexible Job-shop Scheduling Problem," *IEEE Access*, vol. 8, pp. 33 125–33 136, 5 2020.

[19] A. M. Moghadam, K. Y. Wong, and H. Piroozfard, "An efficient genetic algorithm for flexible job-shop scheduling problem," in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, 12 2014, pp. 1409–1413.

[20] G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3563–3573, 4 2011.

[21] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers & Operations Research*, vol. 35, pp. 2892–2907, 4 2008.

[22] M. Mastrolilli and L. M. Gambardella, "Effective neighbourhood functions for the flexible job shop problem," *Journal of Scheduling*, vol. 3, no. 1, pp. 3–20, 1 2000. [Online]. Available: https://doi.org/10.1002/(SICI)1099-1425(200001/02)3:1⟨3::AID-JOS32⟩3.0.CO2-Y

[23] M. A. Cruz-Chávez, "Neighbourhood generation mechanism applied in simulated annealing to job shop scheduling problems," *International Journal of Systems Science*, vol. 46, no. 15, pp. 2673–2685, 2015. [Online]. Available: https://doi.org/10.1080/00207721.2013.876679

[24] T. van den Boom and B. de Schutter, "Lecture notes Optimization in systems and control," pp. 76–78, 2021.

[25] T. Witkowski, P. Antczak, and A. Antczak, "The application of simulated annealing procedure for the flexible job shop scheduling problem," in *Proc. 11th Int. Conf. Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU*, 2006, pp. 21–26.

[26] S. Yang, Z. Guohui, G. Liang, and Y. Kun, "A novel initialization method for solving Flexible Jobshop Scheduling Problem," in *2009 International Conference on Computers Industrial Engineering*, 7 2009, pp. 68–73.

[27] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, pp. 8373–8385, 10 1998. [Online]. Available: https://doi.org/10.1088/0305-4470/31/41/011

[28] M. Ziaee, "A heuristic algorithm for solving flexible job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 71, no. 1, pp. 519–528, 2014. [Online]. Available: https://doi.org/10.1007/s00170-013-5510-z