Applying hierarchical tabu search to an adapted version of the flexible job shop problem

Robin Jansen
Supervisor(s): Kim van den Houten, Mathijs de Weerdt
EEMCS, Delft University of Technology, The Netherlands

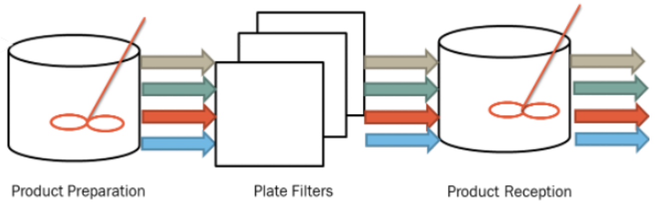
A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

The scheduling departments of batch manufacturing plants have to repeatedly solve a complex scheduling problem for the operation of their production lines. This problem can be modeled as a flexible job shop problem (FJSP) in which a set of operations has to be assigned to a set of machines and then the order of operations on each machine has to be determined. The main difference to the general FJSP is that there are changeover times that appear between two sequential operations on the same machine. To solve this extended problem, a hierarchical tabu search method has been chosen. This algorithm makes use of a global selection initialization procedure as well as two neighborhood functions for the assignment and sequencing subproblems. The objective of this project is to show the effectiveness of tabu search on this version of FJSP compared to a mathematical model serving as a baseline. The initialization procedure performs well compared to the baseline on larger instances while the neighborhood functions worsen the initially found result. This is also the case for a random initialization method which leads to believe that these neighborhood functions are non-optimal and should be replaced which could not have been achieved due to time constraints.

1 Introduction

The scheduling departments of batch manufacturing systems have to repeatedly solve a complex scheduling problem for the operation of their production lines [11]. In this problem, the example of a DSM enzyme plant is used. The enzymes that are produced in this plant each have to follow a specific recipe to be manufactured. These recipes each have a set of operations that have to be performed in order for the final product to be produced correctly and each of the operations can be performed on a certain type of machine. One additional caveat is that machines have to be cleaned in between operations, which complicates the scheduling. These cleaning times depend on which products are produced on the machine and in what order. The cleaning time might be different between two operations on the same machine if they are being swapped in order. The image below shows an illustration of a three step process for some enzyme to be produced [11].



The goal of scheduling these operations on machines is to optimize some objective criterion. This can be the makespan of the overall schedule, the idle time of all the machines in the plant or the tardiness of production related to due dates of incoming orders. This paper will focus solely on the makespan of the produced schedules.

This scheduling problem of a DSM plant enzyme production line can be modeled as an adapted version of a flexible job shop problem, which is a combinatorial NP-Hard problem in computer science [4]. The mentioned adaptation is the inclusion of changeover times that might occur between operations which depends on the order in which operations for certain products are performed on a machine. Although this problem can be used in more occasions, the project starts off with the specific example of a DSM plant. During this project, an algorithm to solve the general flexible job shop problem is altered to fit the adapted version of the problem. The knowledge gained during the project comes from applying this adapted algorithm to provided problem instances and compare the obtained results with a baseline. Due to the complexity of the problem and the size of the problem instances, finding an exact solution in reasonable time seems almost impossible. To this end a heuristic method, which creates a trade-off between the optimality of the solution and the running time of the algorithm, is a good alternative. This paper will therefore concentrate its efforts towards hierarchical tabu search [2] [5] and its performance. Since this version of FJSP with the added complexity of changeover times has not been looked at in other research, there is no direct implementation that can be found. Instead, an adapted version of the algorithm has to be produced. The performance of an algorithm is measured in the optimality of its solution to the set of test instances. To accurately measure its performance, a mixed integer linear programming approach [7] is used to create a baseline of results that other results will be compared to. Mixed integer linear programming is an exact method that makes use of a mathematical model using a set of constraints to solve each problem instance.

Following this introduction, some background information on both algorithms used is discussed in section 2. This section also contains a formal description of the flexible job shop problem. Section 3 explains in detail how hierarchical tabu search works and highlights the most important steps. Section 4 covers the experimental setup and the results obtained from those experiments. The aspect of responsible research is talked about in section 5. Finally, the conclusion and possible future work are presented in section 6.

2 Background information

2.1 Problem description

This research paper focuses on a simulated production line of a DSM plant in which multiple enzymes are being produced. The production of each of these enzymes follows the corresponding recipe which consists of multiple steps that have to be executed in that specific order and each of these steps takes a certain time to complete. All of the steps can be performed on a certain subset of the total set of machines available in the plant. The objective is to minimize the overall makespan of the plant, but other goals could also be considered, like deadlines for each of the orders that come in. This problem can be efficiently modeled as a flexible job shop problem.

The flexible job shop problem (FJSP) [4] can be explained as follows: The goal is to schedule n jobs on m machines. A specific job is denoted with j and a specific machine with i. Each of these n jobs consists of a set of n_j operations denoted as $O_{j,1}$, ..., O_{j,n_j}. These operations have to be executed in order one after the other. Each of the operations of a job can be performed on a predetermined subset of machines and takes $p_{i,j}$ time.

In this version of the flexible job problem, there exists an additional cost which has to be taken into account, namely the changeover time. This can occur between 2 distinct operations that are being performed consecutively on the same machine. This changeover cost is dependent on the type of enzymes that are being processed on a certain machine and the order they are being processed in but not on the choice of the machine. In the specific case in which the scheduling of a DSM production line for chemical enzymes is simulated, these changeover times can be regarded as the cleaning times of the machines in between operations.

One additional change to the general flexible job shop problem is that each machine can only support one type of operation, like filtering or mixing, for each enzyme that is being produced. This reduces the flexibility of assigning an operation to a machine and makes the problem slightly less challenging.

Finally, the time to perform an operation in this version of FJSP is not dependent on the choice of machine. The running time is identical for each machine on which that particular operation can be run.

The flexible job shop problem can be split into two sub-problems. The first one is an assignment problem in which each operation of each job is going to be assigned to a certain machine which can perform that operation. The second is a sequencing problem, in which the already assigned operations will be given a sequence in which the machine will perform these operations. This fact gives us the opportunity to split the algorithm into two main parts, each covering one of the sub-problems.

For each schedule that is presented as a feasible result to the FJSP, a certain set of conditions has to fulfilled:

- Each operation is only assigned to one machine
- Two jobs cannot be performed on one machine at the same time, there must be a precedence relation between jobs
- All operations of one job have to be performed in order
- The finishing time of a job is the completion time of its final operation
- The difference between the start and the completion time of an operation is its processing time on a machine
- The makespan is at least the latest completion time of any operation

2.2 Mixed integer linear programming

Before starting the implementation of the main algorithm, namely hierarchical tabu search, an initial algorithm using mixed integer linear programming provides a baseline for obtained results. Mixed integer linear programming (MILP) [7] is a mathematical optimization model that tries to optimize an objective function while adhering to a specific set of constraints. The objective function in question is the makespan of the produced schedule which the MILP algorithm tries to minimize. There are multiple different models that use different inequalities and constraints to create the mathematical model, each of which has their own specific advantages. More details on this can be found in [13], [12] and [14]
The mathematical model used in this research was provided by the supervisor and its set of constraints has to hold for any feasible schedule, not just for the ones produced by the MILP solver. Since this method uses exact values to find a feasible solution, the constraints have been translated into mathematical equalities and inequalities which look as follows [11]:

$$\min C_{max}$$

$$\sum_{k\in M_j} X_{ijk} = 1 \qquad \forall i \in J \quad \forall j \in O_i \quad (1)$$

$$S_{i,j,k} + C_{ijk} \leq X_{i,j,k} \cdot L \qquad \forall i \in J \quad \forall j \in O_i \quad \forall k \in M_j \quad (2)$$

$$C_{i,j,k} \geq S_{i,j,k} + p_{i,j,k} - (1 - X_{i,j,k}) \cdot L \qquad \forall i \in J \quad \forall j \in O_i \quad \forall k \in M_j \quad (3)$$

$$S_{i,j,k} \geq C_{g,h,k} + co_{g,i,k} - (2 - X_{i,j,k} - X_{g,h,k} + Y_{i,j,g,h,k}) \cdot L \qquad \forall i < g \; \forall j \in O_i \; \forall h \in O_g \forall k \in M_j \cap M_h \quad (4)$$

$$S_{g,h,k} \geq C_{i,j,k} + co_{i,g,k} - (3 - X_{i,j,k} - X_{g,h,k} - Y_{i,j,g,h,k}) \cdot L \qquad \forall i < g \; \forall j \in O_i \; \forall h \in O_g \forall k \in M_j \cap M_h \quad (5)$$

$$\sum_{k\in M_j} S_{i,j,k} \geq \sum_{k\in M_j} C_{i,j-1,k} \qquad \forall i \in J \quad \forall j \in O_i - \{O_{i,l(i)}\} \quad (6)$$

$$F_i \geq \sum_{k\in M_j} C_{i,O_{i,l(i)},k} \qquad \forall i \in J \quad (7)$$

$$C_{max} \geq C_i \qquad \forall i \in J \quad (8)$$

Without going into detail for each of the inequalities, these formulae are the mathematical representation of the constraints listed in the problem description.

2.3 Hierarchical tabu search

Hierarchical tabu search [9] is the combination of two more general search algorithms. Tabu search is a general metaheuristic search algorithm that can be applied to a number of optimization problems. It employs a local search method, meaning that it moves from one solution to another by applying changes to that solution until one considered to be optimal is found. At the start of the algorithm, tabu search creates an empty list with moves that are considered tabu. A move in this case is a schedule out of the neighborhood which is a set of schedules that are similar to the current schedule and are obtained by applying small changes to the current schedule. The algorithm always tries to take the best possible move out of this neighborhood based on some criterion, in this case the makespan of the schedule. If however that schedule is present in the list of tabu moves, the algorithm tries to find the next best solution, until one which is not considered tabu is found and picked. At the end of each iteration of the algorithm, the current schedule is being placed into the tabu list. The main goal of this approach is to avoid getting stuck in a local minimum of the makespan in which the algorithm would go back and forth between two or more solutions without gaining an improvement in the result.

Hierarchical algorithms are specific to the flexible job shop problem [8]. An FJSP can be split into two sub-problems, namely assignment and sequencing. When the two sub-problems are handled simultaneously, the algorithm is considered integrated. A hierarchical approach splits the two sub-problems and creates a hierarchy between them. In this case, the assignment problem is handled first, and based on the assignment of operations on machines, sequencing is performed.

The main advantages of hierarchical tabu search are the computational simplicity of the algorithm itself and the simple implementation. As will be explained in the next section, the algorithm consists of two main parts, each of them covering one of the sub-problems. These parts are very similar in nature and implementation. Based on the stopping conditions that are chosen, the algorithm runs rather fast, especially compared to the mixed integer linear programming method.

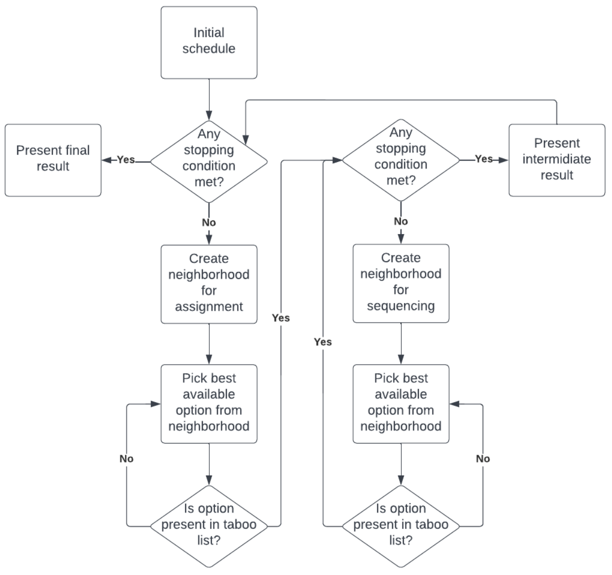
3 Implementation

The methodology used in this paper is to use an existing algorithm that solves the flexible job shop problem and see how an adapted version of that algorithm would perform on the adapted version of FJSP detailed in section 2. To accurately compare the final results, an algorithm is needed to provide initial results that will act as a baseline. This baseline algorithm makes use of mixed integer linear programming. The algorithm that will be repurposed for this specific flexible job problem is hierarchical tabu search.

This section explains in detail how the overall algorithm functions, what the most important parts of it are, and how each of them functions. These parts include the initialization procedure and two neighborhood functions. This section also contains information on the graph representation that is being used throughout the entire algorithm to represent a feasible schedule.

Note that this section will only cover the specific implementation and will not contain any results obtained after running the algorithms. These will presented in section 4.

3.1 Hierarchical tabu search


Initial schedule
Present final result
Yes
Any stopping condition met?
No
Create neighborhood for assignment
Pick best available option from neighborhood
No
Is option present in taboo list?
Yes
Yes
Any stopping condition met?
Yes
Present intermidiate result
No
Create neighborhood for sequencing
Pick best available option from neighborhood
No
Is option present in taboo list?

In the image above, a block representation of the algorithm can be found. There is an inherent symmetry found in this graph. This is due to the hierarchical nature of the algorithm and the fact that both the assignment and sequencing subproblems of the flexible job shop problem are solved using tabu search.

Both parts of the algorithm start off with an initial solution and initialize an empty tabu list. Some stopping conditions will also be defined. These can take multiple forms like the number of overall iterations the loop will go through or the number of iterations without improvement in the makespan of the schedule. The loop starts by creating a neighborhood to the current schedule. In the assignment part of the algorithm, the changes applied to the current schedule will only swap operations around between machines and will not consider the sequencing of these operations. In the sequencing step of the algorithm the neighborhood set will only be made up of schedules in which some sequence of operations on a machine is changed without swapping operations between machines. From this neighborhood, the schedule with the lowest makespan will be chosen for the next iteration as long as it is not in the list of tabu moves. If this is the case, the second best will be considered and so on. During the assignment step of the algorithm, this schedule will be sent to the sequencing step for further improvement in the makespan and serves as the initial schedule for that part of the algorithm. At the end, the current schedule will be added to the list of tabu moves for a certain number of iterations. This number is determined beforehand. During the whole looping process, the algorithm keeps track of the best schedule it has encountered up until that point and at the end, when one of the stopping conditions has been reached, will return this schedule as a result.

The code block below shows pseudo code of the general algorithm. The symmetry between the two parts of the algorithm can clearly be seen here. After a move has been chosen from the assignment neighborhood, it will then be processed in the sequencing part of the algorithm for further optimization.

```
def assignment(n):
  init = getInitial()
  tabu = []

  current = init
  optimalResult = getMakeSpan(init)
  optimal = init
  stoppingCondition = false

  while (not stoppingCondition):
    neighbours = getNeighborhoodAssignment()
    move = selectBestNonTabuMove(neighbours)
    sequencedMove = sequencing(move)

    moveResult = getMakeSpan(sequencedMove)
    if (moveResult < optimalResult):
      optimalResult = moveResult
      optimal = move

    tabuAdd(current)
    if (len(tabu) == n)
      remove the last value of tabu
    current = move
    update(stoppingCondition)
  return optimal

def sequencing(n, init):
  tabu = []

  current = init
  optimalResult = getMakeSpan(init)
```

```
optimal = init
stoppingCondition = false

while (not stoppingCondition):
  neighbours = getNeighborhoodSequencing()
  move = selectBestNonTabuMove(neighbours)

  moveResult = getMakeSpan(move)
  if (moveResult < optimalResult):
    optimalResult = moveResult
    optimal = move

  tabuAdd(current)
  if (len(tabu) == n)
    remove the last value of tabu
  current = move
  update(stoppingCondition)
return optimal
```

3.2 Graph representation

To illustrate and manipulate schedules in the implementation of the algorithm, a disjunctive graph representation is used [1]. This representation proved to be useful mainly for the neighborhood functions. The disjunctive graph can be explained as follows:

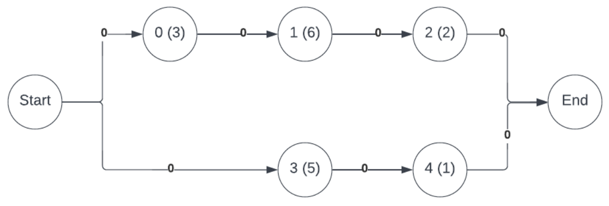
Each operation of each job is represented with its own node with a certain cost attached to it, which is the cost of performing this operation on an available machine.

Two dummy nodes, called the starting and ending nodes, are constructed to represent the beginning and ending of the schedule. These two nodes do not carry any cost and are used only for simplification.

An edge with zero cost originating from the starting node will lead to the first operation of each job. Similarly, an edge with zero cost will go out from the last node of each job and lead to the ending node.

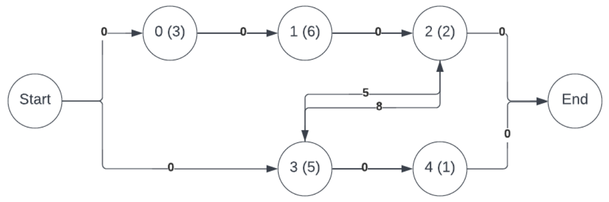
Operations belonging to the same job have to be performed consecutively and without overlap. To this end each operation in a certain job, with the exception of the last operation, has an outgoing edge with zero cost towards the next operation in the same job.

The following graphic shows a simple example of an FJSP with two jobs, the first with operations 0-2 and the second with operations 3 and 4. Each node and edge is accompanied by their cost.



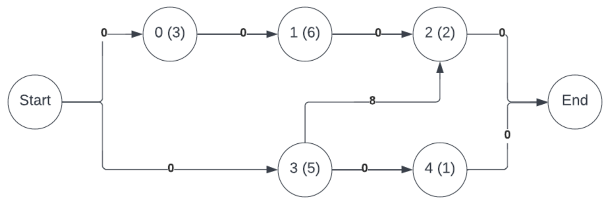
During the assignment step of the algorithm, each operation is assigned to a certain machine. There is also the addition that changeover times might occur between these operations. This will be represented by edges going from each operation to each of the other operations that are being performed on the same machine. The cost of these edges is determined by the changeover times between operations. Note that the changeover times might differ following the order in which two operations are performed on a machine. As such the two edges going in opposite directions between two nodes might have different costs associated to them.

The next image shows the same instance as before, but operations 2 and 3 are now assigned to the same machine.



During the sequencing step of the algorithm, all operations assigned to the same machine will be given an order in which they are performed on that machine. To represent this, the edges between these operations will be used. With the exception of the last operation, each operation will retain an edge going from itself to the next operation in the sequence. All other edges from the previous step will be removed.

After sequencing the operations, it is shown in the next image that operation 3 precedes operation 2.

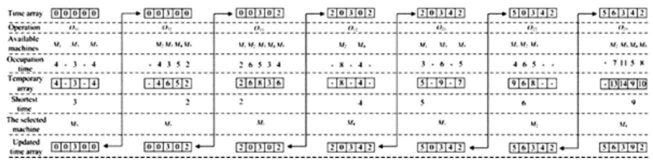


For this graph representation, it is easy to implement a method to find the makespan and the critical path of the proposed solution [1]. The critical path is the set of nodes that make up the longest path in the graph from the starting node to the ending node. It plays an important role in determining effective neighborhoods for the solution, which will be discussed in more detail in the neighborhood functions section. The makespan is obtained by finding the latest completion time of any scheduled operation. The method to come by the critical path is a depth-first search of the graph that takes into account the makespan and halts as soon as a viable path with that length is found.

A second advantage of the graph representation is the ease of swapping operations and thus creating neighborhoods. This holds for both the assignment and sequencing neighborhoods. No nodes have to be exchanged at any time and swapping edges is a simple matter of looking up the corresponding changeover times. The base graph, represented by the first image in this section, will never change. The precedence constraints and running times of operations stay the same for each schedule.

3.3 initialization method

To create the initial schedule for the algorithm, a global selection procedure is used. This method allows for assigning operations on machines by taking into account the maximal occupation times of each machine. In the image below, the procedure for a general flexible job shop problem is shown [10].



Global selection works as follows:

An array, which will be called the time array, is initialized. This array has the same amount of elements as there are machines to keep track of their total occupation times. All elements are initialized to zero.

Jobs are picked one after the other in some order. In the original version of global selection the job is chosen at random, but in this version the jobs are ordered in descending order by the combined running time of all their operations. This way, jobs that take up the most time will be scheduled first and jobs with smaller operations can be scheduled around it. After having chosen a job, the next step is to loop over each operation of that job in order. For each operation a new array, called the temporary array, is created. For each machine, if the operation can be run on it an entry is added to the temporary array. This entry adds up the corresponding element in the time array, the cost of the operation and a changeover time if necessary. If the machine cannot run the operation, a null value is added to the list.

The machine with the smallest non-null value in the temporary array will be chosen to perform the operation in the final schedule and the time array will be updated accordingly for that machine. This process will be applied to each operation of each job in the chosen order until a complete schedule is created.

The initialization procedure is an important step in finding an optimal solution for the flexible job shop problem. It provides a starting point for the rest of the algorithm to improve upon the original schedule produced by this global selection method. According to [10], this approach shortens the computational time used and generates better results compared to data obtained from other research sources.

3.4 Neighborhood functions

Hierarchical tabu search makes use of two distinct neighborhood functions, one that is used during the assignment step of the algorithm and the other one that is used during the sequencing step.

The first neighborhood function concerns itself with the sequencing step of the algorithm. Given an initial schedule it creates a new set of schedules similar to the initial one, each having some small changes. These changes only concern the sequence of the operations on a certain machine so no operation will be assigned to a different machine during this process.

The sequencing neighborhood function makes use of the critical path of the graph representation of the schedule. This is the path with the longest cost when going from the starting node to the ending node. The cost of this path is the makespan of the schedule. Since the goal of the overall algorithm is to reduce the makespan of the schedule as much as possible, it only makes sense to apply changes in the operation sequences that are part of the critical path. These changes are the only ones that might possibly reduce the makespan of the given schedule.

The critical path is represented as a list of nodes. Each pair of consecutive nodes in this list is connected by an edge going from the first node to the second with a given cost. The change that is applied to create a new schedule is to swap the order of a certain pair in the critical path. If the two nodes in the pair represent two consecutive operations of the same job, they cannot be exchanged due to the precedence constrains imposed by the operation order. If it is possible to swap the operations, then all edges going in or out of any of the two nodes will be changed, as long as they come from or go into nodes from different jobs. The edges going into the first node will now go into the second one, the edge going from the first to the second node will be reversed and all edges going out of the second node will now start at the first node.

The second neighborhood function focuses solely on the assignment of operations. Starting off with an initial solution, it creates its set of neighboring schedules by switching two operations that are performed on a different machine. The sequences on the two machines will stay the same. Since two consecutive operations in a job cannot be performed on the same job, they are impossible to be chosen to swap between machines and so none of the precedence constraints can be violated during swapping.

Same as the sequencing neighborhood function, the assignment neighborhood function makes use of the critical path. For each entry on this path that is not one of the two dummy nodes, the machine that operation is run on is found. For each other machine that that operation can possibly run on a random operation on it is chosen to be swapped. When the operations are being swapped, both the list of possible edges and the list of chosen edges have to be adapted accordingly. All edges coming out of the current node from the critical path will now originate from the chosen node and the other way around. All edges going into the current node from the critical path will now go into the chosen node and the other way around.

An advantage of the both of these neighborhood functions is their computational simplicity. The longest part of the algorithm is looking for the correct critical path. Since this is done in a depth-first approach, this method has a squared time complexity. All the rest is done in linear time which makes the algorithm very time-efficient compared to other methods and especially the MILP.

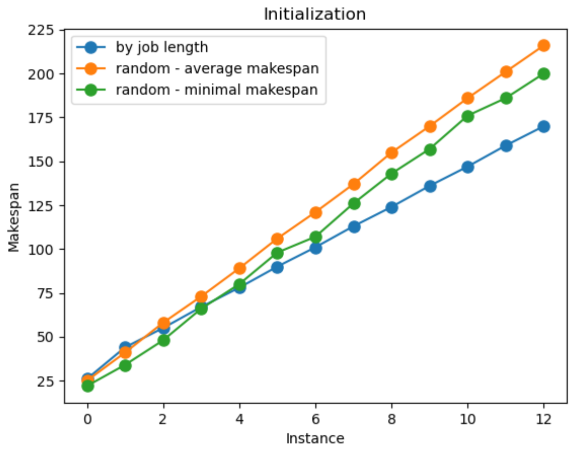
Another advantage is the fact that the graph representation does not have to be changed drastically to produce new feasible schedules. The edges that are used for swapping two edges on the same machine during scheduling are already stored during assignment. The only step is to change the edges accordingly, which is a simple matter. During assignment, new edges have to be created between nodes and the costs of these edges can be found in the base information provided by the problem instance. No precedence relations or node costs have to be changed at any time during the whole process of creating neighborhoods.

4 Experimental Setup and Results

To evaluate the performance of the hierarchical tabu search algorithm, a set of thirteen instances was provided at the start of the project. Each of these instances contains the same amount of machines for each task and possible enzymes to be produced. The difference lies in the amount of jobs that have to be completed which in turn changes the complexity of the problem instance. The evaluation will be done based on the makespan of the obtained result, which is the completion time of the last operation on any machine.

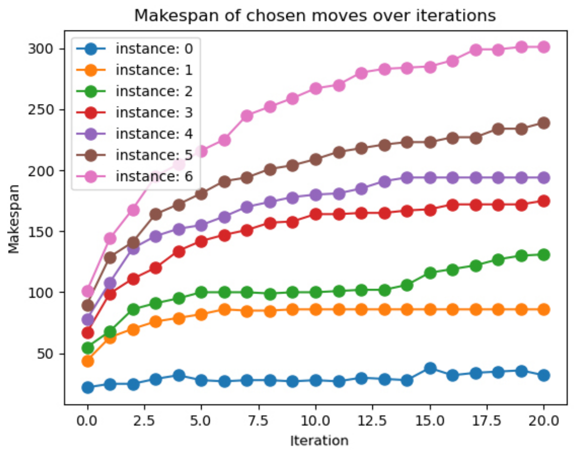
All of the experiments were run on an HP ZBook laptop with an Intel(R) Core(TM) i7-8750Hz CPU @ 2.20 GHz and the implementation is written in Python. Although most of the experiments only focus on the makespan results of the algorithm, some of them are time-sensitive and depend on the hardware used.

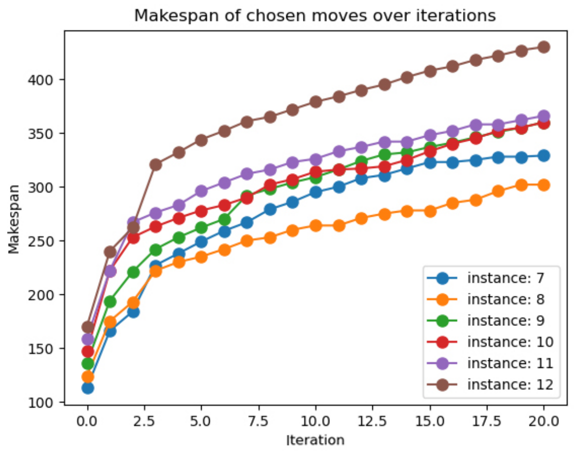
The first graph contains results obtained after only initialising the instances and creating the first schedules based on the aforementioned heuristic. All of the tests make use of global selection, but the order in which the jobs are processed differs. For the random order, the initialization process has been run fifty times and the shown values are the average and minimal makespans obtained from these runs. It is expected that the job length order approach creates superior results than a random choice approach. It can be seen that sorting the jobs by the summed length of their operations is a better method to find an initial schedule than a random order for bigger instances. For the first few, small instances, a random order initialization might produce superior results. However, due to the nondeterministic nature of a random approach, these results are not guaranteed and become worse compared to the ordered approach when the size of the problem instance increases.



In the next two graphs, the evolution of the makespan of the chosen moves is shown. These moves are the schedules with the lowest makespan that are non-tabu resulting from the neighborhood functions. The algorithm went through twenty iterations for each of the instances with a tabu length of five, meaning that after five iterations, a previously picked move may be chosen again. The algorithm should improve the result after a few iterations at most. Even if the initially created schedule is a local minimum, the algorithm tries to get out of it by temporarily choosing worse moves.

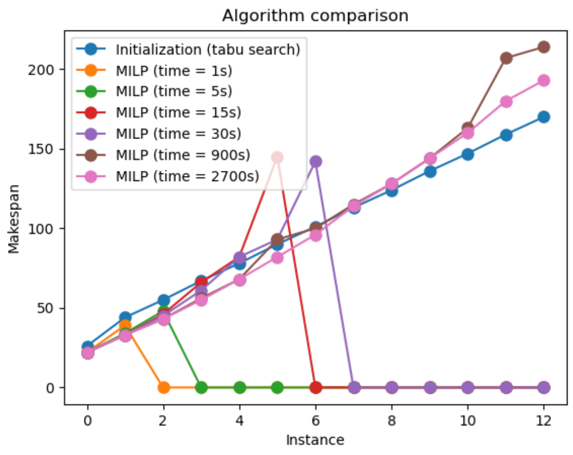
It is rather apparent that the makespan of the initialization procedure found at iteration 0 produces the best results. After the initial values, the makespans of the chosen moves keeps climbing and no new optimum is reached. This climbing process keeps going until some critical value is reached which is different for each instance. The values for larger, more complex instances keep going up for more iterations of the algorithm. When the critical point is reached, the values stabilize. The cause of this is most likely the tabu length used in this experiment. A set of schedules is sequentially picked to be the next move and on the fifth iteration, when the first schedule disappears from the set of tabu moves, that schedule is chosen again and the loop starts anew. It takes longer to find a set of looping schedules for more complex instances due to the amount of possible changes made and the slight randomness used in the assignment neighborhood step of the algorithm. It becomes nearly impossible to find a sequence of five schedules that will make the algorithm loop.



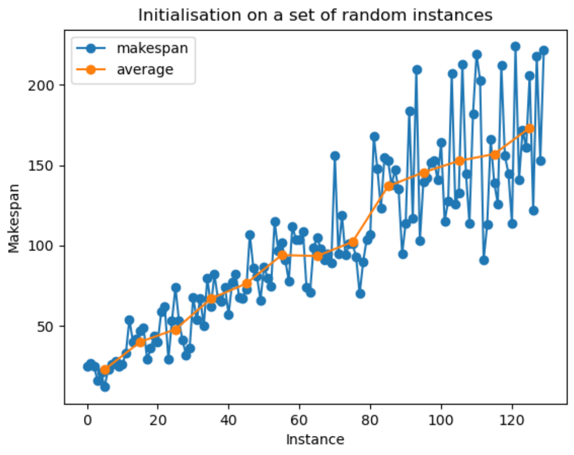


This next graph shows the comparison between the MILP and hierarchical tabu search. If the makespan is zero for a certain instance, this means that the MILP did not have enough time to produce a result. This is why, for shorter times, the results seem to plummet at certain instances that were too large to be processed in time to create a feasible schedule. The results for hierarchical tabu search are the ones obtained from the initialization procedure since those were the best in terms of makespan. Even if the overall algorithm does not work as expected, the initial results might still be superior to the ones found by MILP.

For smaller instances, the MILP performs slightly better than tabu search. As soon as the instance size becomes larger however, the global selection procedure outperforms even the longest running MILP, which takes forty-five minutes. For real-life use cases, which are often times more complex in nature, global selection is more efficient. The set of problem instances is rather small. Nonetheless, it provides enough information to conclude that the global selection procedure improves upon the MILP in terms of results for larger instances. Creating new instances will only confirm what is already known from the initial set of results. The MILP can find better results for smaller instances but the global selection procedure overtakes it at some point when the problem instances become more complex.

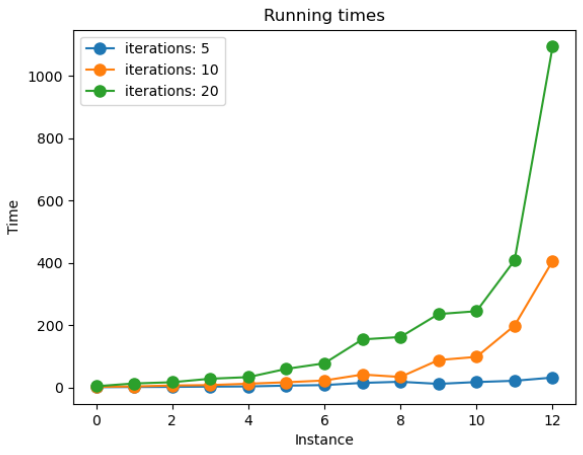


The set of problem instances is rather small. Nonetheless, it provides enough information to conclude that the global selection procedure improves upon the MILP in terms of results for larger instances. The MILP can find better results for smaller instances but the global selection procedure overtakes it at some point when the problem instances become more complex. For the next experiment, a set of 130 instances has been created in sets of ten. The complexity for the instances for each of the sets is comparable to the original instances since the same instance constructor was used. The changes lie in the processing and changeover times. The amount of machines, jobs and possible products stays the same which in turn means that the complexity of the problem instances stays the same as well. In the graph below, the makespan of each of the instances can be found as well as the average makespan for each of the thirteen sets. The individual makespans can differ wildly in one set due to the randomness in the creation. The average makespan for each set is comparable to the makespan of the corresponding original provided instance.



This last graph shows the running times for different iteration amounts that hierarchical tabu search goes through for each instance. The time investment scales with the amount of iterations and the complexity of the instance itself. The initialization procedure, which is not shown on this graph, never takes more than two seconds for any instance.

With a duration of just over 18 minutes for the biggest instance, it is clear that, based on time alone, hierarchical tabu search is more efficient than the mathematical model. With the incredibly short running time of the global section initialization and the improved results from a previous graph, one can conclude that this method outperforms the mathematical model in both speed and result.



5 Responsible Research

All of the obtained results discussed in the previous section can be reproduced by someone else if they follow the same implementation as described in section 3 and use the same hardware as mentioned in section 4. The precise code can also be found in the github repository[1]. The results might differ in some places due to the randomness used in the assignment neighborhood function for the makespan. The time

[1]https://github.com/whodatbo1/research_project_dsm_enzymes

investment might also differ slightly based on the computational power of the hardware on which the MILP and hierarchical tabu search algorithms are run on. The conclusions from those experiments, no matter the hardware choices, should stay the same as the relative change in time and results should be identical to the ones obtained during this project.

All of the obtained results are presented in this paper and no data is left out to prove some hypothesis. Each of the makespans in the graphs originates from a schedule. These schedules are checked for correctness using a feasibility method that looks over the schedule and sees that each of the constraints mentioned in section 2.3 is met.

6 Conclusions and Future Work

In this paper, a scheduling problem of the enzyme production line of a DSM plant is translated into a variation of the flexible job shop problem (FJSP). The difference to the general flexible job shop problem arises from the fact that changeover times can occur between sequential operations performed on the same machine. This problem is solved by applying two different algorithms. The first is a mathematical model making use of mixed integer linear programming (MILP), whose results serve as a baseline to compare other results to. The second is a heuristic approach, namely hierarchical tabu search (HTS). A hierarchical method splits the FJSP into two sub-problems. These are the assignment and sequencing problems, the second of which corresponds to a normal job shop problem. Both of these sub-problems are solved using a tabu search method.

These algorithms are being compared based on time investment and optimality of the final schedule. A schedule is, in this case, superior to another one if its makespan, which is the completion time of the last operation on any machine, is lower.

The initialization method, which is a global selection algorithm, creates the best schedules for each instance in the case of HTS. For medium-sized and larger instances, this method creates more optimal schedules than the MILP at its longest running time. Since the time invested is only a few seconds at most, even for the largest instance, this initialization method can be considered superior to the MILP. As mentioned in section 3.3, this initialization method is superior in time investment and produced results compared to other initialization methods. According to the obtained results, this also holds for a mixed integer linear programming approach.

After the initialization procedure, the algorithm only produces schedules that are worse than the initial one. This issue can be traced back to the neighborhood functions used in the HTS algorithm. They fail to find similar schedules that improve on the makespan of the original one. A possible reason for this are the added constraints of this version of FJSP, namely the changeover times and the fact that only certain operation types can be performed on certain machines.

The running times of HTS increase with the complexity of the problem instance and the amount of iterations that the algorithm goes through. Even so, the running times are below the time the MILP method necessitates to achieve adequate results or even produce a result for larger instances.

In future work, it is recommended to look at alternative neighborhood functions which create schedules that actually improve the makespan of the solution if possible. One possible proposal is to look at an integrated approach that combines both the assignment and scheduling sub-problems as presented in [6] or [3]. Additional future research could look into the difference of integrated and hierarchical approaches specifically for a tabu search method and their effectiveness in solving the flexible job shop problem.

References

[1] Balas E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, 1968.

[2] Hurink J, Jurisch B, and Thole M. Tabu search for the job-shop scheduling problem with multi-purpose machines, 1993.

[3] Li J, Pan Q, and Suganthan N. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, 2010.

[4] Xie J, Gao L, Peng K, Li X, and Li H. Review on flexible job shop scheduling, 2019.

[5] Dell'Amico M and Trubian M. Applying tabu search to the job-shop scheduling problem, 1993.

[6] Mastrolilli M and Gambardella M. Effective neighbourhood functions for the flexible job shop problem, 2000.

[7] Wagner M. An integer linear-programming model for machine scheduling, 1959.

[8] Zribi N, Kacem I, Kamel E, and Borne P. Assignment and scheduling in flexible job-shops by hierarchical optimization, 2007.

[9] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search, 1993.

[10] Yang S, Guohui Z, Liang G, and Kun Y. A novel initialization method for solving flexible job-shop scheduling problem, 2009.

[11] van den Houten K. Algorithm for smart scheduling of a dsm enzyme production line, 2022.

[12] Demir Y and Isleyen K. Evaluation of mathematical models for flexible job-shop scheduling problems, 2012.

[13] Unlu Y and Mason J. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems, 2010.

[14] Özgüven C, Özbakır L, and Yavuz Y. Mathematical models for job-shop scheduling problems with routing and process plan flexibility, 2009.