# The Hybrid Flow Shop Problem

Neil Sanjiv Punjani

## Contents

### Main Body

1. Introduction and Problem Description ........ 1
2. Literature Review ........ 2
3. Methodologies in Paper [11] ........ 3
  3.1 MILP Model ........ 3
    3.1.2 Applying this model to our problem ........ 3
  3.2 Hybrid Genetic Algorithm ........ 4
    3.2.1 Hybrid Genetic Algorithm Explanation ........ 4
    3.2.2 Applying Hybrid Genetic Algorithm to our problem ........ 5
4. Results discussion for paper [11] ........ 6
  4.1 Testing and Computational Results ........ 6
  4.2 Results in the context of our problem ........ 6
5. Simulated Annealing Approach ........ 7
  5.1 Explanation of NSA ........ 7
  5.2 Applying the Novel Simulated algorithm to our problem ........ 8
6. Results and Observations of NSA: ........ 9
  6.1 Results in the context of our problem: ........ 10
7. Final Discussion and Conclusion ........ 10

### References and Appendix

8. References ........ 12
9. Appendix ........ 14
  9.1 MILP formulation and Constraint Explanation ........ 14
  9.2 Explanation in-depth of the Genetic Algorithm ........ 15
    9.2.1 Applying the GRASP algorithm for machine allotment ........ 15
    9.2.2 Applying Selection, Crossover and Mutation for improvement in job sequences ........ 16
    9.2.3 A note on parameter tuning ........ 17
  9.3 Explanation of Simulated Annealing in Depth ........ 17
    9.3.1 The SPTCH and Machine Allotment Methodology for initial solution generation: ........ 17
    9.3.2 Local Neighbor searches and Inverse Moving Operators ........ 18
    9.3.3 The Simulated Annealing Probability Distributions and Temperature Cooling ........ 19
    9.3.4 Testing Combinations: ........ 20
    9.3.5 A note on Parameter Tuning ........ 20

## 1. Introduction and Problem Description

Following the assignment to develop a daily machining schedule that ensures the completion of all jobs while minimizing the makespan, I have analyzed our current machining processes and constraints. This section outlines our operational setup, the challenges we face, and the constraints under which we operate, aiming to provide a basis for optimizing our machining schedule.

Our company specializes in machining cast iron parts for the automobile industry, utilizing a range of machinery that includes 30 CNC Turning Machines which perform operations on the outside diameter of circular profile parts along a horizontal axis, 20 VMC machines Used for milling, tapping, or drilling operations, 5 Manual Drilling Machines which are used for specialized drilling operations and Oiling machinery that allow for parts to undergo oiling for rust prevention and lubrication before dispatch.

**We have the following Operational Constraints**

1. Sequential Machine Processing: Parts must follow a sequence of CNC machining -> VMC machining -> Specialized Drilling -> Oiling. Not all parts require every step, depending on complexity.
2. Mandatory Oiling: All parts require oiling, regardless of the machining processes they undergo.
3. Single Part Processing: Each machine can only process one part at a time, constrained by machine operation and fixture setups.
4. Variable Processing and Setup Times: Each part type has unique processing and setup times on different machines, dictated by the complexity of the required operations.
5. Efficient Setup for Repeated Part Types: If a part type has already been set up on a machine, subsequent parts of the same type incur no additional setup time.
6. Labor and Fixture Availability: We operate with sufficient labor and fixtures to utilize all machines concurrently within a 16-hour daily timeframe.
7. No Buffer Space: Limited shop floor space and wear and tear considerations prevent the use of buffer spaces between machines. Parts must wait on the previous machine if the next machine is occupied.
8. Identical Machine Capabilities: Each machine in a given stage (e.g., all CNC machines) is identical in terms of processing and setup times for a specific part.
9. Non-Interruptible Jobs: Once a machining job begins, it cannot be paused or removed, ensuring consistency and quality in the machining process.

Upon performing a literature review, papers [1], [2] and [3] helped me identify our specific instance of the problem and is called **the Hybrid Flow-shop Scheduling (HFS) problem with identical parallel machines and sequence dependent set-up times (SDT).** Paper [3] provides a visual representation of HFS

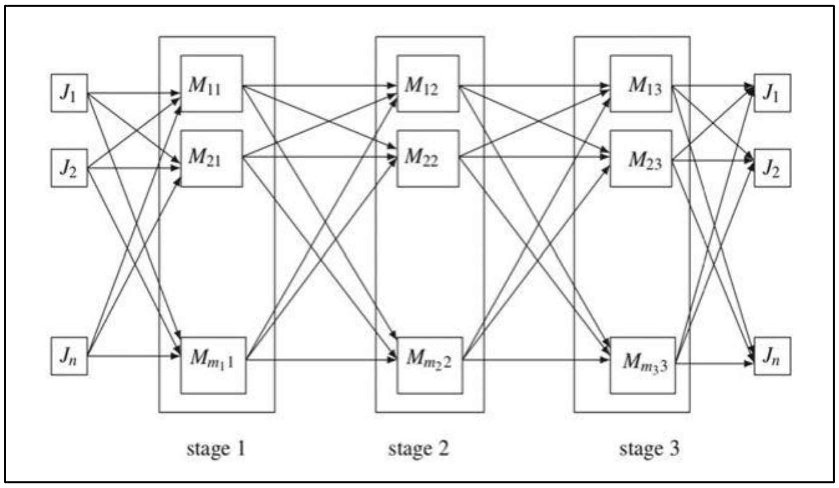


**Figure 1:** The Hybrid Flowshop Problem

Papers [4], [5], and [6] highlight the complexity of scheduling problems in machining operations. [4] demonstrates that solving for the shortest processing time in regular flow shop settings (with machines in series and no parallel operations) is NP-complete when there are more than three machines. [5] extends this complexity to two-stage HFS problems, showing they are NP-complete when any stage involves more than one machine. [6] further indicates that scheduling problems with sequence-dependent setup times, even for a single machine or stage, are as complex as the Travelling Salesman Problem, marking them among the toughest scheduling challenges.

Due to the difficulties of our problem, I conducted a literature review in order to determine efficient algorithms designed by researchers and if we can apply them to the context of our problem to get high quality solutions.

## 2. Literature Review

The literature review revealed no studies directly addressing our specific problem conditions simultaneously. Highly cited papers on the HFS problem with sequence-dependent setup times, such as [6], [7], and [8], focus on different problem structures. They do not consider stage skipping, processor blocking, buffer space, or wait times between stages, with paper [8] explicitly excluding wait times. The review of real-life industrial applications in papers [9] and [10] revealed a mismatch due to their focus on specialized issues like varying processing times, absence of blocking, multiple job processing simultaneously, and different objectives, making their solutions less applicable or completely infeasible to our problem.

After performing a more focused search into papers that closely align with our problem type, I chose papers [11] and [12] to be the focus of finding the appropriate solution to our problem. Paper [11] is titled **“A hybrid genetic algorithm for the hybrid flow shop scheduling problem with machine blocking and sequence-dependent setup times”** and published in the **Journal of Project Management by Growing Science** in **2022.** My reasons for choosing paper [11] are as follows:

- Paper [11] is extremely relevant to the conditions of our problem, as it allows for processor blocking. It also deals with identical parallel machines and sequence dependent times, which are central to our problem.
- This is one of the few papers that deal with our problem conditions. Minimal research has been done in the context of processor blocking and sequence dependent times simultaneously for HFS. Other papers that deal with this type of problem are [13] and [14]. However, [13] does not provide any results of testing on different combinations of jobs and machines while [14] deals with the problem of unrelated machines which have varying speeds for the same part which does not apply in our case.
- The paper provides a Mixed Integer Linear Programming Model (MILP), which can be a possible algorithm for use. Various commercial software is available for solving MILP models and hence this model is something I wished to bring to your consideration.
- The paper also provides an updated Hybrid Genetic Algorithm to solve our problem and has been shown to be effective. I will discuss the results in **section 4.1** and **section 4.2.**
- Paper provides extensive testing on different problem instances and combinations and compares the performance of the MILP and the HGA which will allow us to make solid decisions regarding either algorithm to solve our problem. The paper also provides information about the pseudocode for their implementation of the HGA in Python which can help us implement this algorithm for our problem.

<u>This paper has been published very recently in 2022, which aids us twofold</u>

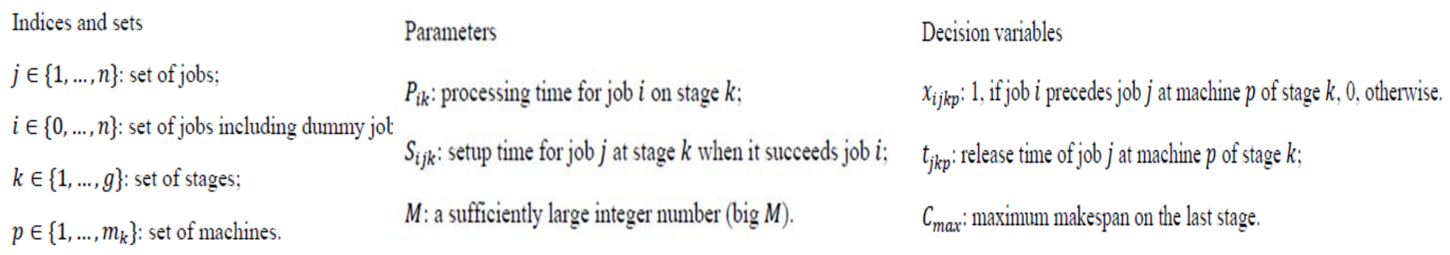
- CPLEX, the software deployed by IBM receives updates every year and improvement in performance along with it. By having information about the computational performance of CPLEX that is most recent, we can know it’s most recent performance and effectiveness in dealing with our problem
- As this paper is published recently, it builds upon earlier research in Hybrid Flow shop scheduling problems in terms of genetic algorithms and other meta heuristics as evident from their literature review, which helps us provide a more state of the art approach, potentially helping us get better performance for our case.

## 3. Methodologies in Paper [11]

### 3.1 MILP Model

#### *3.1.1 MILP Formulation Explanation*

I have included the complete formulation as presented in the paper in **Appendix Section 6.1**. I have also broken down each constraint using help from the paper's explanation of the constraints as well as my understanding so it is easier for you to understand how they relate to our problem. I am not including them in the main body of this recommendation as it is not as relevant to the understanding of whether we can solve our problem using this MILP method. The parameters in **Figure 2** will provide some intuition to you regarding how the modeling has been conducted if you do not wish to look at the entire formulation.



Indices and sets

$j \in \{1, \ldots, n\}$: set of jobs;

$i \in \{0, \ldots, n\}$: set of jobs including dummy jo

$k \in \{1, \ldots, g\}$: set of stages;

$p \in \{1, \ldots, m_k\}$: set of machines.

Parameters

$P_{ik}$: processing time for job $i$ on stage $k$;

$S_{ijk}$: setup time for job $j$ at stage $k$ when it succeeds job $i$;

$M$: a sufficiently large integer number (big $M$).

Decision variables

$x_{ijkp}$: 1, if job $i$ precedes job $j$ at machine $p$ of stage $k$, 0, otherwise.

$t_{jkp}$: release time of job $j$ at machine $p$ of stage $k$;

$C_{max}$: maximum makespan on the last stage.

**Figure 2**: Variable Notations for the MILP model.

The objective as per your request, was to minimize the time of completion of all jobs in order to increase efficiency, save resource and labor costs and increase customer satisfaction. This objective of improvement is often called makespan and is referred to as the time between the start of the first job and the end time of the last job. This is the objective function that is also focused on in this paper.

$$\boldsymbol{Objective}: \ \min C_{max}$$

This would essentially be the release time of the last job of completion at its last stage which can be represented as $t_{jkp-Max}$ where $k$ is the index of the last stage. The jobs indices specifically use a dummy job in the set. This allows for us to consider the sequence dependent set-up time for the first job for that machine in that stage begins allowing for a simpler formulation and also helps by setting a clear start time for when the scheduling process begins.

#### *3.1.2 Applying this model to our problem*

This model can be applied to our problem to generate feasible solutions to generate machining schedules.

- We have multiple jobs that are the same type, allowing for no set-up time if jobs of the same type follow the other. We can parametrize this by setting sequence dependent setup time as 0 for $S_{ijk}$ for if same types of jobs follow each other and the rest as the setup time of each independent job type itself for each stage.
- Due to our no buffer allowance between the stages, the blocking constraint (8) ensures that no product will be allowed to move into our next stage of machining till a machine is available and constraints (1), (2), (3), (4), (5), (6), (7) outlined help us follow the sequence of our flow shop and ensures that each product follows the job processing sequence without skipping the stage where it is required to be processed.
- However, the formulation provided does not explicitly account for the fact that some of our products can skip stages based on their machining requirements. The formulation will still provide us feasible solutions if we set the processing time $P_{ik}$ = 0 and $S_{ijk}$ = 0, for all jobs $i$, $j$ that do not need to be processed at stage $k$.
- A cause of concern is that our model treats non-processed jobs at those stages as 'dummy' jobs that still queue for machine assignment, causing unnecessary delays due to processor blocking as even if machines are idle in the next stage, they will not be allowed to be assigned. This could lead to sub-optimal scheduling solutions, inadequately representing the true flexibility of our process.
- Adding redundant decision variables $x_{ijkp}$ for stages that can be skipped increases the MILP model's complexity, potentially slowing computational time by complicating solver techniques like branching and cutting planes, and impacting the efficiency of solver heuristics

- In order to solve these issues, we can define a new index $K = \{ s_{1,}s_{2\dots}s_j \}$ where each $s_j$ is a sequence that each of our jobs j need to follow in our flow shop. If job 1 needs to go through CNC, VMC and oiling, then the set $S_1 = \{1,2,4\}$. We can adjust the constraints in the MILP formulation carefully to such that each constraint accurately reflects this flow process for a unique job. For example, constraint (1) can be represented as $\sum_{i=0,i\neq j}^{n} \sum_{p=1}^{m_k} x_{ijkp} = 1 \quad \forall j \in \{1,\dots.,n\}, \forall k \in s_j, s_j \subseteq K$. This constraint would allow only the jobs that need to be processed at stage k to be assigned.
- More information has been provided in paper [15] where they present a MILP model that explicitly accounts for stage skipping. However, they do not account for processor blocking, assume unlimited storage between stages and have minimal testing of their model, hence I did not wish to present this formulation to you. If at the end of the results section, you wish to use MILP models, I can make modifications to the MILP model using methodology I described previously and using paper [15] as a reference.

## 3.2 Hybrid Genetic Algorithm

### 3.2.1 Hybrid Genetic Algorithm Explanation

In order to address the widespread knowledge in literature that MILPs are not very effective for higher sized HFS problems, the authors have developed an algorithm called the Hybrid Genetic Algorithm. An in-depth explanation of their algorithm in their paper is not provided, so by complimenting the paper with the research in papers [16], [17], and [18], I have understood their methodology and created a flowchart with a higher-level overview that outlines the main steps taken in the algorithm. If you wish to receive more in-depth details regarding how this methodology works, please see **Appendix section 6.2**

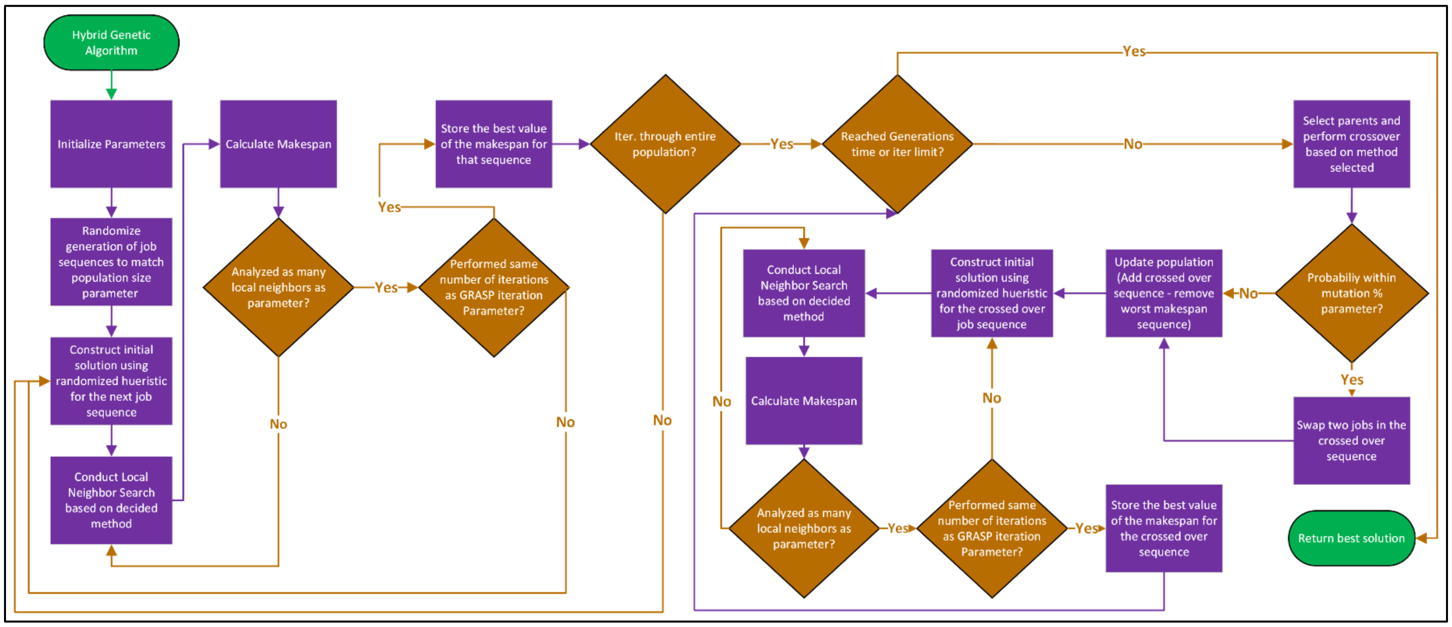

Hybrid Genetic Algorithm
Initialize Parameters
Randomize generation of job sequences to match population size parameter
Construct initial solution using randomized hueristic for the next job sequence
Conduct Local Neighbor Search based on decided method
Calculate Makespan
Analyzed as many local neighbors as parameter?
Yes
No
Performed same number of iterations as GRASP iteration Parameter?
Yes
No
Store the best value of the makespan for that sequence
Iter. through entire population?
Yes
No
Reached Generations time or iter limit?
Yes
No
Select parents and perform crossover based on method selected
Probabily within mutation % parameter?
Yes
No
Swap two jobs in the crossed over sequence
Update population (Add crossed over sequence - remove worst makespan sequence)
Construct initial solution using randomized hueristic for the crossed over job sequence
Conduct Local Neighbor Search based on decided method
Calculate Makespan
Analyzed as many local neighbors as parameter?
Yes
No
Performed same number of iterations as GRASP iteration Parameter?
Yes
No
Store the best value of the makespan for the crossed over sequence
Return best solution

**Figure 3:** Hybrid Genetic Algorithm Flowchart

The main ideas of the algorithm are as follows:

1. We start with a randomized number of job sequences based on the size of the population parameter created earlier, which remains the same across all stages in terms of priority of job assignment.

2. We used a greedy heuristic (earliest completion time of job) with a randomization parameter (GRASP) that allocates jobs to each machine with the same priority job sequence. Neighbors of the generated solution are searched to receive the best makespan for that particular sequence in a fixed number of iterations.

3. This process is repeated for the same job sequence as many times as specified in the GRASP iterations. This process is continued for every single job sequence that was initially generated.

4. Two parents are selected. This method is conducted using a probability distribution that selects jobs sequences with a better makespan value (Called Fitness in GAs). These parents are crossed over by selecting jobs in the sequence randomly from each parent and assigning them in the same position as the parent.

5. The algorithm determines whether to mutate the crossover sequence if the probability is within a certain number. If so, then two random positions of jobs in the crossed-over sequence are swapped.

6. This new sequence replaces one of the job sequences in the population that had the worst makespan. Then GRASP is applied to this new generated sequence and then parents are again selected, cross overed, mutated and replaces the worst makespan sequence in the original population.

7. This process continues for the parameter set, for example the number of these offspring (called generations) or time of running. The best makespan is then returned.

The main idea of this Genetic Algorithm (GA) is that by crossing over better sequences consistently, we allow the algorithm to explore better job sequences and machine assignments that can give us higher quality solutions. (Lower Makespan). By randomizing job allocations, local neighborhood searches and mutation, we help the algorithm not to get stuck in a local minimum solution space.

### *3.2.2 Applying Hybrid Genetic Algorithm to our problem*

This genetic algorithm can be used for our instance of solving our problem.

- The algorithm handles our special case of processor blocking, by ensuring that the GRASP algorithm does not allow for a job to be scheduled until there is a machine that is idle. It also accounts for sequence dependent set-up times by using this in the machine allocation formula.
- The GRASP equation presented in paper [11] and **appendix section 6.2.1** has a variable called machine time which accounts for the idle time of the machines for that particular job, which is atleast the time that the job was completed at the prior stage depending on the sequence. This prevents the job to be processed in the next stage till it is processed in the prior one.
- The algorithm will produce feasible job sequences for us, as it forces each job to be processed at each of the stages ensuring that it does not miss any stage required for it. Stage skipping is not directly accounted for leading to inefficiencies, but ensuring that it is allocated at all stages ensures feasibility.

There are some considerations in the algorithm which will reduce its effectiveness in our problem

1. Not exactly optimal due to considerations of fixing job sequencing. According to paper [16], it has been stated that the optimal sequence of jobs scheduling is generally among the non-permutational sequences. However, fixing job sequencing schedule is required for computational efficiency, fixing the search space and generating a feasible sequence.

2. Forced sequencing means a job that could skip a stage gets stuck waiting for an idle machine, despite not needing processing there. Even in the next stage for that job, it still won’t be allowed to be processed earlier as job sequencing is fixed for all stages. This will lead to high inefficiencies as the problem is not able to exploit the stage skipping structure of jobs and the makespan will be further away from the optimal.

3. Paper [16] offers a method of designing the formulation of sequencing in such a way that we can account for stage skipping. The sequence “breaks” at the job sequencing stage where the skipping can conduct, and then reformulates in the next stage where skipping for that job does not hold. Further investigation is required to determine how to adopt his within the framework of the presented genetic algorithm.

4. The local neighborhood search and RCL generation in GRASP are not thoroughly explained in paper [11], leaving gaps in understanding their implementation for job sequencing and machine allocation. However, I have some insights in **Appendix section 6.2.1** and with help from paper [18], I will be able to understand their methodology and implement it if you would like choose to proceed with this method.

5. The pseudocode provided in the paper is a solid point for developing this algorithm. However, their implementation code is not provided, so implementing this will require time.

## 4. Results discussion for paper [11]

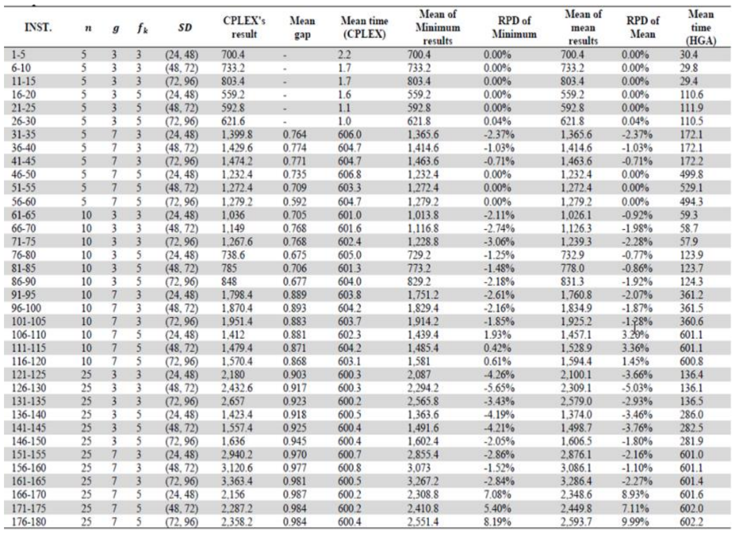
### 4.1 Testing and Computational Results

•The authors have conducted testing of these developed algorithms. The MILP model was inputted in CPLEX 12.8 while the Hybrid Genetic Algorithm was performed in python 3.8 Spyder IDE.

•Currently, the latest CPLEX model is available is 20.1 and Python's latest version available is 3.12 both of which have experienced good performance growth from the versions used in the models for use in our implementation.

•The authors have used **formula (1)**[1] to calculate the performance of MILP vs HGA:

$$RPD = \frac{HGA\ Result - MILP\ Result}{MILP\ Result} \quad \mathbf{(1)}$$



| INST. | $n$ | $g$ | $f_k$ | $SD$ | CPLEX's result | Mean gap | Mean time (CPLEX) | Mean of Minimum results | RPD of Minimum | Mean of mean results | RPD of Mean | Mean time (HGA) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1-5 | 5 | 3 | 3 | (24, 48) | 700.4 | - | 2.2 | 700.4 | 0.00% | 700.4 | 0.00% | 30.4 |
| 6-10 | 5 | 3 | 3 | (48, 72) | 733.2 | - | 1.7 | 733.2 | 0.00% | 733.2 | 0.00% | 29.8 |
| 11-15 | 5 | 3 | 3 | (72, 96) | 803.4 | - | 1.7 | 803.4 | 0.00% | 803.4 | 0.00% | 29.4 |
| 16-20 | 5 | 3 | 5 | (24, 48) | 559.2 | - | 1.6 | 559.2 | 0.00% | 559.2 | 0.00% | 110.6 |
| 21-25 | 5 | 3 | 5 | (48, 72) | 592.8 | - | 1.1 | 592.8 | 0.00% | 592.8 | 0.00% | 111.9 |
| 26-30 | 5 | 3 | 5 | (72, 96) | 621.6 | - | 1.0 | 621.8 | 0.04% | 621.8 | 0.04% | 110.5 |
| 31-35 | 5 | 7 | 3 | (24, 48) | 1,399.8 | 0.764 | 606.0 | 1,365.6 | -2.37% | 1,365.6 | -2.37% | 172.1 |
| 36-40 | 5 | 7 | 3 | (48, 72) | 1,429.6 | 0.774 | 604.7 | 1,414.6 | -1.03% | 1,414.6 | -1.03% | 172.1 |
| 41-45 | 5 | 7 | 3 | (72, 96) | 1,474.2 | 0.771 | 604.7 | 1,463.6 | -0.71% | 1,463.6 | -0.71% | 172.2 |
| 46-50 | 5 | 7 | 5 | (24, 48) | 1,232.4 | 0.735 | 606.8 | 1,232.4 | 0.00% | 1,232.4 | 0.00% | 499.8 |
| 51-55 | 5 | 7 | 5 | (48, 72) | 1,272.4 | 0.709 | 603.3 | 1,272.4 | 0.00% | 1,272.4 | 0.00% | 529.1 |
| 56-60 | 5 | 7 | 5 | (72, 96) | 1,279.2 | 0.592 | 604.7 | 1,279.2 | 0.00% | 1,279.2 | 0.00% | 494.3 |
| 61-65 | 10 | 3 | 3 | (24, 48) | 1,036 | 0.705 | 601.0 | 1,013.8 | -2.11% | 1,026.1 | -0.92% | 59.3 |
| 66-70 | 10 | 3 | 3 | (48, 72) | 1,149 | 0.768 | 601.6 | 1,116.8 | -2.74% | 1,126.3 | -1.98% | 58.7 |
| 71-75 | 10 | 3 | 3 | (72, 96) | 1,267.6 | 0.768 | 602.4 | 1,228.8 | -3.06% | 1,239.3 | -2.28% | 57.9 |
| 76-80 | 10 | 3 | 5 | (24, 48) | 738.6 | 0.675 | 605.0 | 729.2 | -1.25% | 732.9 | -0.77% | 123.9 |
| 81-85 | 10 | 3 | 5 | (48, 72) | 785 | 0.706 | 601.3 | 773.2 | -1.48% | 778.0 | -0.86% | 123.7 |
| 86-90 | 10 | 3 | 5 | (72, 96) | 848 | 0.677 | 604.0 | 829.2 | -2.18% | 831.3 | -1.92% | 124.3 |
| 91-95 | 10 | 7 | 3 | (24, 48) | 1,798.4 | 0.889 | 603.8 | 1,751.2 | -2.61% | 1,760.8 | -2.07% | 361.2 |
| 96-100 | 10 | 7 | 3 | (48, 72) | 1,870.4 | 0.893 | 604.2 | 1,829.4 | -2.16% | 1,834.9 | -1.87% | 361.5 |
| 101-105 | 10 | 7 | 3 | (72, 96) | 1,951.4 | 0.883 | 603.7 | 1,914.2 | -1.85% | 1,925.2 | -1.28% | 360.6 |
| 106-110 | 10 | 7 | 5 | (24, 48) | 1,412 | 0.881 | 602.3 | 1,439.4 | 1.93% | 1,457.1 | 3.20% | 601.1 |
| 111-115 | 10 | 7 | 5 | (48, 72) | 1,479.4 | 0.871 | 604.2 | 1,485.4 | 0.42% | 1,528.9 | 3.36% | 601.1 |
| 116-120 | 10 | 7 | 5 | (72, 96) | 1,570.4 | 0.868 | 603.1 | 1,581 | 0.61% | 1,594.4 | 1.45% | 600.8 |
| 121-125 | 25 | 3 | 3 | (24, 48) | 2,180 | 0.903 | 600.3 | 2,087 | -4.26% | 2,100.1 | -3.66% | 136.4 |
| 126-130 | 25 | 3 | 3 | (48, 72) | 2,432.6 | 0.917 | 600.3 | 2,294.2 | -5.65% | 2,309.1 | -5.03% | 136.1 |
| 131-135 | 25 | 3 | 3 | (72, 96) | 2,657 | 0.923 | 600.2 | 2,565.8 | -3.43% | 2,579.0 | -2.93% | 136.5 |
| 136-140 | 25 | 3 | 5 | (24, 48) | 1,423.4 | 0.918 | 600.5 | 1,363.6 | -4.19% | 1,374.0 | -3.46% | 286.0 |
| 141-145 | 25 | 3 | 5 | (48, 72) | 1,557.4 | 0.925 | 600.4 | 1,491.6 | -4.21% | 1,498.7 | -3.76% | 282.5 |
| 146-150 | 25 | 3 | 5 | (72, 96) | 1,636 | 0.945 | 600.4 | 1,602.4 | -2.05% | 1,606.5 | -1.80% | 281.9 |
| 151-155 | 25 | 7 | 3 | (24, 48) | 2,940.2 | 0.970 | 600.7 | 2,855.4 | -2.86% | 2,876.1 | -2.16% | 601.0 |
| 156-160 | 25 | 7 | 3 | (48, 72) | 3,120.6 | 0.977 | 600.8 | 3,073 | -1.52% | 3,086.1 | -1.10% | 601.1 |
| 161-165 | 25 | 7 | 3 | (72, 96) | 3,363.4 | 0.981 | 600.5 | 3,267.2 | -2.84% | 3,286.4 | -2.27% | 601.4 |
| 166-170 | 25 | 7 | 5 | (24, 48) | 2,156 | 0.987 | 600.2 | 2,308.8 | 7.08% | 2,348.6 | 8.93% | 601.6 |
| 171-175 | 25 | 7 | 5 | (48, 72) | 2,287.2 | 0.984 | 600.2 | 2,410.8 | 5.40% | 2,449.8 | 7.11% | 602.0 |
| 176-180 | 25 | 7 | 5 | (72, 96) | 2,358.2 | 0.984 | 600.4 | 2,551.4 | 8.19% | 2,593.7 | 9.99% | 602.2 |

**Figure 4:** Computational Results Presented in the Paper

- In **Figure 4**, $n, g, f_k$ represent the number of jobs, the number of stages and number of machines at each stage respectively. $SD$ represents the range of the uniform distribution of sequence dependent times. The maximum running time for the algorithm was 600 seconds. HGA was stopped earlier if it received solutions better than MILP model earlier to compare performance.
- MILP model and HGA are both able to find optimal solutions for smaller instances of 5 jobs, 3 stages and 5 machines. The Mean Gap, obtained from CPLEX quantifies this, which is the difference between the best bound for feasible integer solutions and the solution itself.
- As the number of instances increases, HGA performs superiorly in majority of cases than MILP. This is evident in the negative difference in RPD representing a lower makespan result than MILP in significantly lower time. MILP performs better in smaller instances with a much faster processing time. Also, for instances with 25 jobs, 7 stages and 5 machines, MILP performs better in the same amount of time with a lower makespan than the HGA. The number of runs has for each algorithm has not been highlighted by authors.
- Overall, the results are not extremely ideal, as the optimality gap as determined by the best bound of the feasible integer solution and the obtained makespan increases from around 76.4% to 98.4% for the largest instances but this is to be expected considering the low running time of 600 seconds and the complex combinatorial nature of our problem.

### 4.2 Results in the context of our problem

Based on results presented, it is not certain which algorithm is better than the other (HGA does perform better for the majority). However, for instances that are larger than this, there is strong evidence that MILP will start significantly struggling in terms of its performance.

- In paper [15], CPLEX with MILP was tested on HFS with stage skipping but unlimited buffer. With 100 jobs, and just 2 stages and 1 machine per, not even a feasible solution was found in 2 hours of computation time[2].It is extremely likely that we will face the same scenario in our case because our average number of jobs per day is 300, and the machines and the stages are comparatively much greater.
- For HGA, Due to the fixed sequence constraint and the greedy nature of our machine allotment, the HGA is significantly more likely to generate feasible job schedules. Compared to their 600 second tested times, we have a time of 8 hours before the next day's production begins. This will allow the HGA algorithm to iterate through many numbers of generations, which will be feasible sequences and provide us with good solutions.

[1] The formula provided in the paper is $\frac{CPLEX\ Result - Mean\ Result}{CPLEX\ Result}$. However, this seems to be flipped in terms of result they provided.
[2] Study was done in 2004 so CPLEXs' performance was much weaker, but situation likely to remain the same as our problem is incredibly higher in size and combinations

- Also, another factor to consider is that implementing MILP with CPLEX involves purchasing the software which can be quite expensive ($4900/year), while the python implementation for HGA will not bear any cost except the cost of the time it takes to implement this algorithm and potential hurdles faced in relating the algorithm to our instance

Testing done by paper [11] was for a maximum of 25 jobs, 7 stages and 5 machines. With permission, I would like to test each algorithm for the instances of our problem, and see if MILP or HGA performs better. It is very likely that HGA will perform better. If infeasibility persists:

- We can split each of our jobs within specific cells of production each with the same quantity of machines from each stage. We can split heuristically the jobs that are of the same type in the same cell to reduce set up time, and then select jobs that when averaged out have similar processing times.
- We can start from a lower number of divisions and rerun the algorithm to see if feasible and good solutions are provided. If not, we can split it further and run the algorithm.
- This approach can also help us determine the makespan obtained by each job split by the Genetic algorithm and CPLEX and confirm which split works best in terms of returning the lowest makespan.

In conclusion, the provided algorithms are applicable to our problem with the considerations mentioned. If given permission for testing, I can find the ideal algorithm out of the two that allows the lowest makespan and hence higher quality job schedules for our problem.

## 5. Simulated Annealing Approach

I wanted to present to you another algorithm that specifically deals with stage skipping allowed in HFS environment with SDT. Because many of our parts can skip stages, not accounting for this can yield to massive inefficiencies in makespan as parts don't need to be processed at the current stage can get blocked

Furthermore, it has also been shown that metaheuristics and algorithms can find solutions closer to optimal better, if stage skipping is accounted for. In paper [19] they tested instances with increasing number of missing operations in HFS (machines > 1 at each stage) and found increasing probability of getting a solution closer to optimal with higher number of missing operations.

After thorough research, in HFS with SDT environments where stage skipping is considered, I chose paper [12] titled **A simulated annealing algorithm approach to hybrid flow shop in the Journal of Intelligent Manufacturing** which was published in 2010. My reasons for choosing the paper are as follows:

- This is the most recent and one of the few papers that account for stage skipping and our problem structure that I could find. [9], [19] and [20] all account for stage skipping in HFSS and recent. However, [9], is industry specific, and focus is on minimizing wait times, [19] deals with unrelated parallel machines and resource planning, and [20] considers dynamic stage and also uses tardiness as the objective, making all of their algorithms infeasible for our problem.

- This paper is highly cited in the field for stage skipping and conditions of our problem. Other papers that deal with the same conditions are [15] and [7]. Paper [12], directly compares their results to both of those algorithms and has showed significant improvements over the same instances.

- They have incorporated large size of test instances upto 100 jobs with 8 stages and 10 machines, which is much closer to the conditions of our problem. Their computation times are also extremely fast in generating solutions with their algorithm, which is something I wanted to bring to your consideration. Their implementation of the algorithm is very streamlined with most steps explained thoroughly, allowing us to understand and easily implement their algorithm if we choose to do so.

- They do not account for processor blocking, and assume that there is infinite buffer, meaning that if a job is completed, next job can still begin even though there is no machine idle at the next stage.

### 5.1 Explanation of NSA

Due to the combinatorial complexity of the problem and to handle stage skipping, the authors in the paper have developed an algorithm called **Novel Simulated Annealing (NSA)**, I have been able to understand their methodology and explained in detail **in appendix section 8.3**. Below, I have provided a flowchart with a higher-level explanation about how their algorithm works:

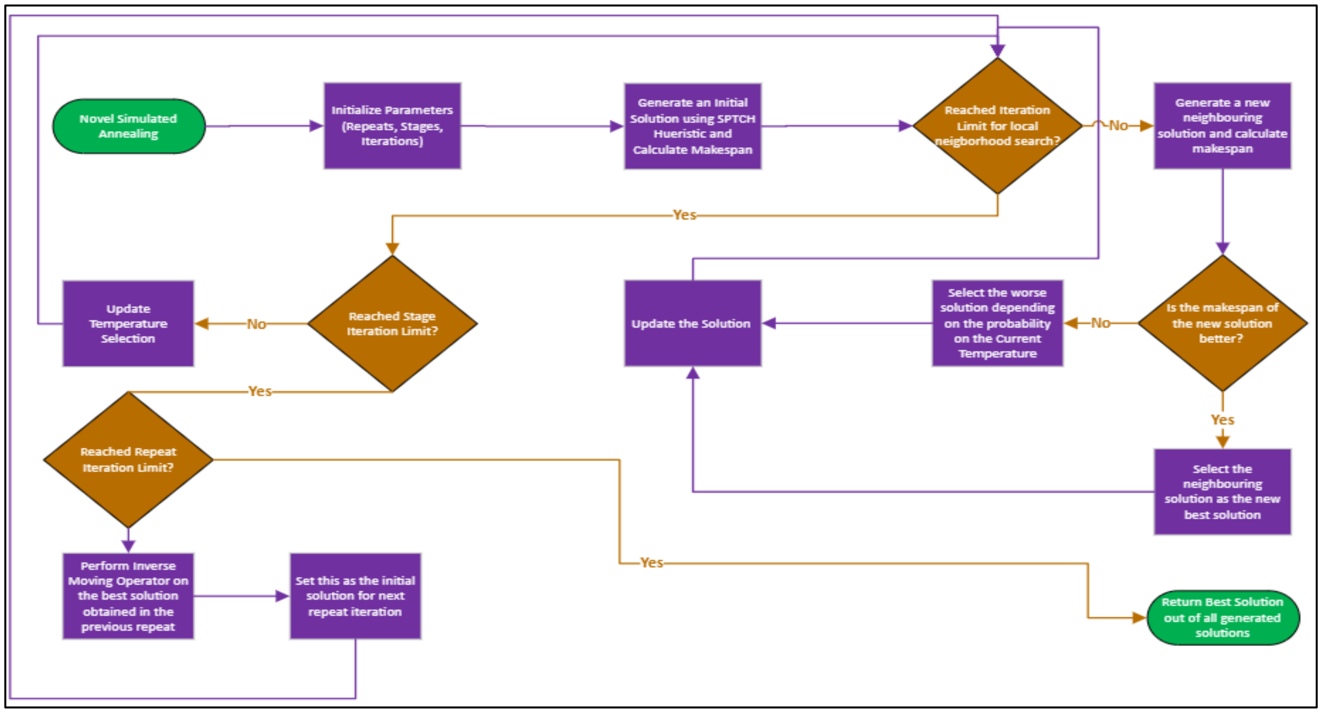

Novel Simulated Annealing
Initialize Parameters (Repeats, Stages, Iterations)
Generate an Initial Solution using SPTCH Hueristic and Calculate Makespan
Reached Iteration Limit for local neigborhood search?
No
Generate a new neighbouring solution and calculate makespan
Yes
Update Temperature Selection
No
Reached Stage Iteration Limit?
Update the Solution
Select the worse solution depending on the probability on the Current Temperature
No
Is the makespan of the new solution better?
Yes
Yes
Reached Repeat Iteration Limit?
Select the neighbouring solution as the new best solution
Yes
Perform Inverse Moving Operator on the best solution obtained in the previous repeat
Set this as the initial solution for next repeat iteration
Return Best Solution out of all generated solutions

**Figure 5:** Novel Simulated Algorithm Flowchart

1. We start with the process of generating an initial feasible sequence. NSA uses Shortest Processing Time Cyclic Heuristic (SPTCH) to order jobs in decreasing order of processing time. In subsequent stages the jobs are sequenced as per the time of completion in prior stages.

2. Machines are assigned to each job in the sequence defined by step 1 that can finish the task the earliest.

3. A local neighbor solution is constructed by swapping the sequence of two jobs at one of the stages, and recalculates the makespan (Fitness Estimator). If the new makespan is an improvement, we select the neighbor solution as our new solution.

3. If the solution is worse, we use the probability of acceptance of worse solutions (Temperature) to decide whether to select a solution or not. We repeat this process for as many local neighbors as determined by the neighbor parameter. Then we reduce temperature.

4. We repeat steps 2-3 for as many temperatures as determined by the stages of temperature parameter by authors of paper/Not to be confused with stages as the stages of the HFS.

5. We use the best solution from 2,3 and 4 for our initial solution. Inverse moving operator is applied to this sequence, which reverses the entire sequence of jobs at one stage between 2 specific jobs. Then, Steps 2-4 repeated as many temperatures and neighbors as we determined.

6. Steps 2-5 are repeated as many times as the repeat parameter. Once all the repeats are complete, we output the best solution.

As the algorithm is allowed to explore worse off solutions, with decreasing probabilities, it avoids being stuck in local optimums. Based on the lower probability of worse off solutions being selected than the better solutions, the algorithm continues moving towards better solutions while exploring more of solution space. Slowly, reducing the temperature allows for exploitation of promising regions. By selecting the best solution at the end of a repeat, and then applying an inverse operator, the algorithm aims to not only focus on a better solution space by selecting a good sequence, but by selecting the inverse operator we can avoid local optimums.

## 5.2 Applying the Novel Simulated algorithm to our problem

1. SDT and identical machines are handled efficiently, by using them in machine allotment and the generation of an initial feasible sequence. SDT is also incorporated in local neighbor search by allowing that resequencing a pair of jobs at a stage can allow for better exploitation of SDT interdependencies in jobs.

2.The key benefit is the exploitation of our stage skipping. As job sequencing is not fixed for all stages, in the initial sequence generated, the algorithm is allowed to not consider the jobs that are not part of that stage. These jobs can directly move on to the next stage with their priority determined by the times they were completed in their last stage of processing. This prevents the job from being blocked and allows processing in its next stage. During local neighbor search, this is respected as these jobs can only be switched in the sequence at stages where they need to be processed, preventing blocking in unnecessary stages.

3. The author's have provided much clearer steps regarding their implementation than the authors of HGA have, with clearer pseudocode and implementation procedures, allowing us to more closely follow the exact approaches taken by the authors for our problems and receive results similar to theirs.

Factors that block its applicability of NSA to our algorithm:

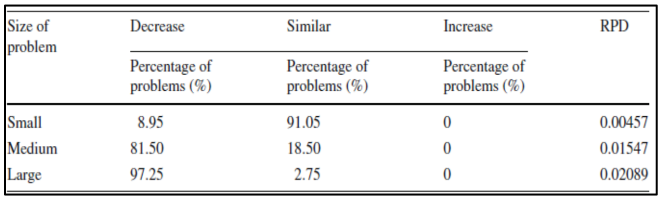
- The algorithm does not account for processor blocking because jobs are considered for processing as soon as the job before it is completed. It does not take into consideration that the jobs in the queue may get blocked and cannot move to the next stage as there is no idle machine in the next stage.
- In order to generate a sequence for the next stage, the algorithm considers the time of completion in the current stage. However now, the time of completion for a job would be dependent on idle machine availability on the next stage, which depends on the sequence of jobs in the next stage. Therefore, the machine assignment becomes circular, and hence sequences will be hard to create.
- To tackle this issue, job sequences need to be fixed from the beginning across all stages, because only then will the algorithm know which machine to assign the job to, considering idle times and earliest release. This is essentially what GRASP did and hence was able to incorporate processor blocking, but this prevents stage skipping from being incorporated.
- Some key information such as starting temperature, method of reduction of temperature is not provided and stage of application of inverse moving operator is not provided making its implementation difficult. Paper [21] provides information regarding tuning an initial temperature as well as cooling rate with sufficient detail, and can be incorporated if we proceed with this approach.

## 6. Results and Observations of NSA:

- The results are compared with other metaheuristics [Simple Simulated Annealing (SSA), Random Keys Genetic Algorithms (RKGA) in paper [15] and Immune Algorithm (IA) in paper [7], using the same problem instances generated. These methods tackle HFS with SDT and Stage Skipping same as NSA.
- A variety of different combinations and parameters were chosen for testing. For each combination, 10 different tests sets are generated with variations in values with distributions. Each combination was grouped in small, medium or large sets with a total of 2520 problems generated. Please see **Appendix section 8.3.4** for full parameter and combination list
- The algorithm was implemented in MATLAB 7.0, on an Intel Pentium Processor (2.4GHZ) PC with 512 MB of RAM. The following formula was used in order to compare the results that they obtained:

$$\boldsymbol{RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}}}$$

- where $Alg_{sol}$ = Makespan obtained by 1 test-value of 1 combination of specific algorithm and the $Min_{sol}$ represents the minimum value of the makespan received across all the different algorithms of the same combination over all of the 10 sets. By averaging the values obtained of the 10 tests for a single algorithm, the RPD value is calculated. For small, medium or large sets, RPD of all combinations within those sets are averaged, to get RPD over entire instance.
- In table X, NSA is compared with SSA (No inverse moving operator), showing its superior performance over large and medium combination sets, with a decrease in

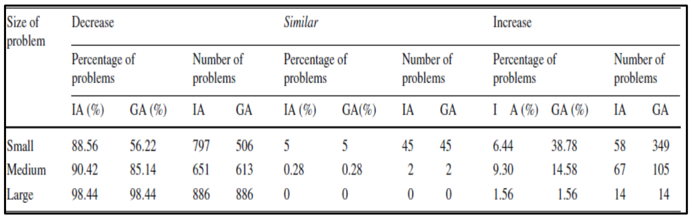


| Size of problem | Decrease | Similar | Increase | RPD |
|---|---|---|---|---|
| | Percentage of problems (%) | Percentage of problems (%) | Percentage of problems (%) | |
| Small | 8.95 | 91.05 | 0 | 0.00457 |
| Medium | 81.50 | 18.50 | 0 | 0.01547 |
| Large | 97.25 | 2.75 | 0 | 0.02089 |

**Figure 6:** Performance of NSA vs SSA

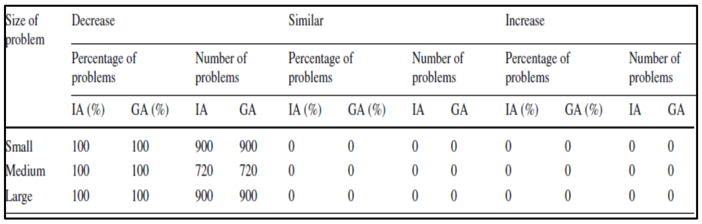
makespan over 97.25% and 81.50% respectively with very low RPD suggesting closeness to the best solution obtained on average.

- According to **Figure 7** and **Figure 8**, In comparison to RKGA and IA, NSA outperforms both. In terms of computation values, it performs better in 100% of combinations for all test sets against both algorithms.
- For makespan values, we see that its capabilities in terms of small and medium datasets are not as pronounced as the computation time, especially for GA's where in 38% and 14.58% of small and medium instances respectively it performs better. However, in 98.44% of large cases it performs much superior, which is the case size that our machining problem closely resembles, hence it is a strong indicator for us.



| Size of problem | Decrease | | | | *Similar* | | | | Increase | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Percentage of problems | | Number of problems | | Percentage of problems | | Number of problems | | Percentage of problems | | Number of problems | |
| | IA (%) | GA (%) | IA | GA | IA (%) | GA(%) | IA | GA | I A (%) | GA (%) | IA | GA |
| Small | 88.56 | 56.22 | 797 | 506 | 5 | 5 | 45 | 45 | 6.44 | 38.78 | 58 | 349 |
| Medium | 90.42 | 85.14 | 651 | 613 | 0.28 | 0.28 | 2 | 2 | 9.30 | 14.58 | 67 | 105 |
| Large | 98.44 | 98.44 | 886 | 886 | 0 | 0 | 0 | 0 | 1.56 | 1.56 | 14 | 14 |

**Figure 7:** Makespan Comparison (NSA vs IA vs RKGA)



| Size of problem | Decrease | | | | Similar | | | | Increase | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Percentage of problems | | Number of problems | | Percentage of problems | | Number of problems | | Percentage of problems | | Number of problems | |
| | IA (%) | GA (%) | IA | GA | IA (%) | GA (%) | IA | GA | IA (%) | GA (%) | IA | GA |
| Small | 100 | 100 | 900 | 900 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Medium | 100 | 100 | 720 | 720 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Large | 100 | 100 | 900 | 900 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 8:** Computation Time Comparison (NSA vs IA and RKGA)

## 6.1 Results in the context of our problem:

We see that NSA performs extremely well over large test instances (largest being -100 jobs, 8 stages with a maximum of 10 machines per stage). With an increase in size, the algorithm performs better, suggesting that it could provide quality solutions with our instance of approximately 300 jobs, 5-30 machines/stage and 4 stages

1. Testing was performed way back in 2008, and on **MATLAB 7.0**, suggesting with new **MATLAB 2024a**, much better solutions can be obtained. The largest instance for 10 Repeats, 100 Stages, and 20 Neighbor search, it took 56.68 seconds only. Increasing the number of repeats, stages and temperatures drastically, and running it for our available time of 8 hours will most likely give us high quality solutions.

2. Purchasing MATLAB is 980$/year. MATLAB is more ideal for numerical computations but Python is quite close in terms of performance, so we can implement our algorithm in Python 12.1 for cost-saving.

3. Testing has not been done on exactly our instances of problem size; therefore, we are not exactly sure of its performance. It is still likely to give feasible sequences based of the nature of the heuristic. We can also adopt a division of jobs and machines in different "cells", dividing similar types of jobs as discussed in **section 4.2**, and compare makespan.

4. As I said in **section 5.2**, they have not provided their initial temperature used and their cooling schedule, so even after incorporating methodologies from paper [21], we may not reach the same quality of results that were obtained by the authors in their test cases of using NSA as their parameters may be inherently different.

5. Even though the sequencies generated by this problem are infeasible, this is the most promising method of incorporating stage skipping inefficiencies for HFS with SDT that is implemented in literature as they have compared it directly with other methods that target the same problem structure. It may make sense for us to employ this method, at your discretion, the pros and cons of which I have discussed in the next section.

## 7. Final Discussion and Conclusion

To compare both methods, I have made a pros and cons table below of HGA and NSA. I will exclude MILP model, because it is not likely to be effective for higher sized instances of problems. If you still wish to use MILP model, we can use CPLEX 20.1, to see if solutions get outputted in our times and the quality of them in comparison to HGA and NSA.

- As NSA provides infeasible sequences, we cannot use it directly. My recommendation would be to test each HGA and NSA for our instances, and if the makespan differences obtained are significant because of stage skipping, the benefits obtained may outweigh the costs of investing in buffer space.

- Paper [16] offers a method to incorporate stage skipping for HGA, which can allow us both processor blocking and stage skipping. I wasn't able to find a way in literature than can directly account for processor blocking for NSA because of the way the heuristic to generate an initial solution is designed, and because of its circular logic if we implement processor blocking.
- For either algorithm we do not know its performance in terms of how close it is to optimal solution. Our best approximation would be running the algorithms for 8 hours and see how the solutions compare to each other. It will be interesting to see if the stage skipping efficiencies and unlimited buffer assumption accounted by NSA, will allow for a better solution than the HGA which explores the solution space much more.

| | **Hybrid Genetic Algorithm** | **Novel Simulated Annealing** |
|---|---|---|
| **Pros** | - Paper is more recent and builds on previous literature.<br>- Computational results reflect recent software capabilities.<br>- Optimizes both job sequences and machine allotment (GA and GRASP), allowing greater exploration of solution space.<br>- Less Likely to get stuck in local optimums.<br>- Will give feasible sequences for our problem<br>- Performed extensive parameter tuning. (comparative to NSA)<br>- Have actual makespan values to compare to. | - Explicitly accounts for Stage Skipping Efficiencies.<br>- Methodology is much clearer than HGA<br>- Tested with bigger instance sizes close to our problem<br>- Computationally much quicker than HGA<br>- Results show efficiency increases in larger problem instances.<br>- Easy to implement due to algorithmic simplicity. |
| **Cons** | - Does not account for stage skipping efficiencies<br>- High computation time due to extensive solution space search<br>- Assumptions made as exact methodology not provided.<br>- Number of instances tested are low with a maximum of 25 jobs,<br>- Cannot find optimal solution with non-permutational sequence<br>- Will be time consuming to implement because of a lack of clear methodology compared to NSA and algorithm complexity. | - Infeasible sequences unless we invest in buffer space<br>- Specifications about cooling and, initial temperature and inverse moving operator are not provided.<br>- Limited machine allotment combinations tested and overall, less solution space explored for job sequences.<br>- Older paper and hence doesn't build on new advancements, computational results are old as well.<br>- Parameter tuning not extensive, we need to try different parameters needing time. |

In conclusion if we implement the solution directly from the literature, HGA will provide feasible sequences and possibly high-quality solutions even though it doesn't account for stage skipping. However, NSA will not, due to its rigidity in adopting processor blocking. Even though our exact problem sizes have not been tested, but given our time of 8 hours, we are likely to receive good quality solutions from both algorithms. We can implement MILP in **CPLEX 20.1**, and HGA or NSA in MATLAB2024a or **Python 12.1.**

I look forward to hearing back from you regarding the approach you would want me to follow, and I can begin work on it.

Sincerely,
**Neil Punjani**

## 8. References

[1] Ruiz, Rubén, and José Manuel Vázquez-Rodríguez. “The Hybrid Flow Shop Scheduling Problem.” European Journal of Operational Research, vol. 205, no. 1, 16 Aug. 2010, pp. 1–18, https://doi.org/10.1016/j.ejor.2009.09.024. Accessed 21 Apr. 2023.

[2] Shao, Weishi, et al. “Modeling and Multi-Neighborhood Iterated Greedy Algorithm for Distributed Hybrid Flow Shop Scheduling Problem.” Knowledge-Based Systems, vol. 194, Apr. 2020, p. 105527, https://doi.org/10.1016/j.knosys.2020.105527. Accessed 23 Jan. 2022.

[3] Aqil, Said & Allali, Karam. (2021). On VNS-GRASP and Iterated Greedy Metaheuristics for Solving Hybrid Flow Shop Scheduling Problem with Uniform Parallel Machines and Sequence Independent Setup Time. 10.1007/978-3-030-58930-1.

[4] Garey, M. R., et al. “The Complexity of Flowshop and Jobshop Scheduling.” *Mathematics of Operations Research*, vol. 1, no. 2, 1976, pp. 117–129, www.jstor.org/stable/3689278?seq=1. Accessed 25 Mar. 2024.

[5] Gupta, Jatinder N. D. “Two-Stage, Hybrid Flowshop Scheduling Problem.” *The Journal of the Operational Research Society*, vol. 39, no. 4, Apr. 1988, p. 359, https://doi.org/10.2307/2582115. Accessed 17 May 2022.

[6] Naderi, B., et al. "An Improved Simulated Annealing for Hybrid Flowshops with Sequence-Dependent Setup and Transportation Times to Minimize Total Completion Time and Total Tardiness." Expert Systems with Applications 36.6 (2009): 9625-33. Web.

[7] Mostafa Zandieh, et al. “An Immune Algorithm Approach to Hybrid Flow Shops Scheduling with Sequence-Dependent Setup Times.” Applied Mathematics and Computation, vol. 180, no. 1, 1 Sept. 2006, pp. 111–127, https://doi.org/10.1016/j.amc.2005.11.136.

[8] Jolai, F., et al. “A Novel Hybrid Meta-Heuristic Algorithm for a No-Wait Flexible Flow Shop Scheduling Problem with Sequence Dependent Setup Times.” International Journal of Production Research, vol. 50, no. 24, 15 Dec. 2012, pp. 7447–7466, https://doi.org/10.1080/00207543.2011.653012. Accessed 10 Feb. 2020.

[9] Long, Jianyu, et al. "Scheduling a Realistic Hybrid Flow Shop with Stage Skipping and Adjustable Processing Time in Steel Plants." Applied Soft Computing 64 (2018): 536-49. Web.

[10] Karmakar, Sandipan & Mahanty, Biswajit. (2010). Minimizing Makespan for a Flexible Flow Shop Scheduling Problem in a Paint Company.

[11] Maciel, I., et al. “A Hybrid Genetic Algorithm for the Hybrid Flow Shop Scheduling Problem with Machine Blocking and Sequence-Dependent Setup Times.” Journal of Project Management, vol. 7, no. 4, 2022, pp. 201–216, growingscience.com/beta/jpm/5495-a-hybrid-genetic-algorithm-for-the-hybrid-flow-shop-scheduling-problem-with-machine-blocking-and-sequence-dependent-setup-times.html. Accessed 25 Mar. 2024.

[12] Mirsanei, H S, et al. “A Simulated Annealing Algorithm Approach to Hybrid Flow Shop Scheduling with Sequence-Dependent Setup Times.” Journal of Intelligent Manufacturing, vol. 22, no. 6, 19 Jan. 2010, pp. 965–978, https://doi.org/10.1007/s10845-009-0373-8.

[13] Zandieh, Mostafa and Eghbal Rashidi. “An Effective Hybrid Genetic Algorithm for Hybrid Flow Shops with Sequence Dependent Setup Times and Processor Blocking.” (2009).

[14] Pessoa, Ronaldo & Maciel, Ingrid & Moccellin, Joao & Pitombeira-Neto, Anselmo & Prata, Bruno. (2019). Hybrid flow shop scheduling problem with machine blocking, setup times and unrelated parallel machines per stage. Investigacion Operacional. 20.

[15] Kurz, Mary E., and Ronald G. Askin. "Scheduling Flexible Flow Lines with Sequence-Dependent Setup Times." European Journal of Operational Research 159.1 (2004): 66-82. Web.

[16] Mir Abbas Bozorgirad, and Rasaratnam Logendran. “A Comparison of Local Search Algorithms with Population-Based Algorithms in Hybrid Flow Shop Scheduling Problems with Realistic Characteristics.” The International Journal of Advanced Manufacturing Technology, vol. 83, no. 5-8, 7 Aug. 2015, pp. 1135–1151, https://doi.org/10.1007/s00170-015-7650-9. Accessed 25 Mar. 2024.

[17] Oĝuz, Ceyda, and M. Fikret Ercan. “A Genetic Algorithm for Hybrid Flow-Shop Scheduling with Multiprocessor Tasks.” Journal of Scheduling, vol. 8, no. 4, July 2005, pp. 323–351, https://doi.org/10.1007/s10951-005-1640-y. Accessed 10 May 2021.

[18] Eliana María González-Neira, and Jairo R Montoya-Torres. “A GRASP Meta-Heuristic for the Hybrid Flowshop Scheduling Problem.” Journal of Decision Systems, vol. 26, no. 3, 3 July 2017, pp. 294–306, https://doi.org/10.1080/12460125.2017.1351863. Accessed 25 Mar. 2024.

[19] Hasani, Ali, and Seyed Mohammad Hassan Hosseini. "Auxiliary Resource Planning in a Flexible Flow Shop Scheduling Problem Considering Stage Skipping." Computers & Operations Research 138 (2022): 105625. Web.

[20] J. -q. Li, Q. -k. Pan, and P. -y. Duan. "An Improved Artificial Bee Colony Algorithm for Solving Hybrid Flexible Flowshop with Dynamic Operation Skipping." *IEEE Transactions on Cybernetics* 46.6 (2016): 1311-24. Web.

[21] Wang, HM., Chou, FD. & Wu, FC. A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan. Int J Adv Manuf Technol 53, 761–776 (2011). https://doi.org/10.1007/s00170-010-2868-z

[22] Minor use of ChatGPT in making some paragraphs more concise after writing the explanations by myself.

[23] Professor Carter and TA Jasper Chen for helping me understand concepts.

## 9. Appendix

### 9.1 MILP formulation and Constraint Explanation

$$\sum_{i=\{0,\dots,n|i\neq j\}} \sum_{p=1}^{m_k} x_{ijkp} = 1 \;\forall\, j = \{1,\dots,n\}, k = \{1,\dots,g\} \quad \textbf{(1)}$$

$$\sum_{j=1}^{n} \sum_{p=1}^{m_k} x_{ijkp} \leq 1 \;\forall\, i = \{1,\dots,n\}, k = \{1,\dots,g\} \quad \textbf{(2)}$$

$$\sum_{j=1}^{n} x_{0jkp} \leq 1 \;\forall\, k = \{1,\dots,g\}, p = \{1,\dots,m_k\} \quad \textbf{(3)}$$

$$\sum_{h=\{0,\dots,n|h\neq i,h\neq j\}} x_{hikp} \geq x_{ijkp} \;\forall\, i = \{1,\dots,n\}, j = \{1,\dots,n \mid i\neq j\}, k = \{1,\dots,g\}, p = \{1,\dots,m_k\} \quad \textbf{(4)}$$

$$t_{0kp} = 0 \;\forall\, k = \{1,\dots,g\}, p = \{1,\dots,m_k\} \quad \textbf{(5)}$$

$$t_{jkp} + M(1 - x_{ijkp}) \geq t_{ikp} + S_{ijk} + P_{jk} \;\forall\, i = \{0,\dots,n\}, j = \{1,\dots,n \mid i\neq j\}, k = \{1,\dots,g\}, p = \{1,\dots,m_k\} \quad \textbf{(6)}$$

$$t_{jkp} - P_{jk} \geq t_{j(k-1)p0} - M\left(1 - \sum_{i=0}^{n} x_{ijkp}\right) \;\forall\, j = \{1,\dots,n\}, k = \{2,\dots,g\}, p = \{1,\dots,m_k\}, p0 = \{1,\dots,m_{k-1}\} \quad \textbf{(7)}$$

$$t_{j(k-1)p0} - S_{ij(k-1)} - P_{j(k-1)} \geq t_{i(k)p} - P_{ik} - M(1 - x_{ij(k-1)p}) \;\forall i = \{0,\dots,n\}, j = \{1,\dots,n\}, k = \{2,\dots,g\}, p = \{1,\dots,m_k\}, p0 = \{1,\dots,m_{k-1}\} \quad \textbf{(8)}$$

$$C_{max} \geq t_{jgp} \;\forall j = \{1,\dots,n\}, p = \{1,\dots,m_k\} \quad \textbf{(9)}$$

$$x_{ijkp} \in \{0,1\} \;\forall i = \{0,\dots,n\}, j = \{1,\dots,n\}, k = \{1,\dots,g\}, p = \{1,\dots,m_k\} \quad \textbf{(10)}$$

$$t_{jkp} \geq 0 \;\forall j = \{1,\dots,n\}, k = \{1,\dots,g\}, p = \{1,\dots,m_k\} \quad \textbf{(11)}$$

- Constraint (1) sets that for each job $j$ at each stage $g$, it cannot be processed in more than 1 machine and that job $j$ must be preceded by exactly one job $i$.
- Constraint (2) helps by setting that for each job $i$, for each stage $g$, no more than one job $j$ can be follow that job and ensures that job $i$ is only processed on one machine at that stage
- Constraint (3) states that for all jobs $j$ which does not include the dummy job, no more than 1 job can follow the dummy job for each machine p at each stage $g$.
- Constraint (4) ensures that if a job $j$ on a particular machine at a particular stage is scheduled then its predecessor job $i$, must also be scheduled on the same machine for that stage. This will ensure proper precedence between the jobs.
- Constraint (5) ensures that the release time of completion for the dummy job is 0 for all machines and all stages.
- Constraint (6) enforces that the time that if job $j$ is the one that follows job $i$ on a machine $p$, the time that job $j$ is released is greater than the time of release for previous job $i$ at that machine + the sequential dependent set time of $j$ following i with the processing time of job $j$ at that stage. If a job $j$ does not follow job $i$, then the constraint becomes redundant due to our implementation of the big $M$.

- Constraint (7) enforces stage sequencing, ensuring that a particular job $j$ cannot be scheduled in the next stage at a particular machine without it being released in the prior stage. If that job $j$ is not sequenced on that machine $p$ in the next stage, then the big $M$ ensures that this constraint is redundant and does not conflict with our other constraints.

- Constraint (8) helps enforce blocking due to a lack of buffer capacity. If job $j$ follows job $i$, in stage $k$-$1$, then the time of it being beginning processing on stage $k$-$1$ must be greater than the time that the previous job $i$ started its job on the next stage $k$, as this ensures that the machine is available for job $j$ to begin in the previous stage. If job $j$ does not follow job $i$ in the previous stage for that machine, then the big $M$ helps ensure this constraint is not active and redundant.

- Constraint (9) enforces that our makespan objective is bigger than or equal to every single release time of the jobs for every machine at the last stage, quantifying the time it takes for us to finish the last job.

- Constraint (10) and (11) helps set the domain of our variables. Our decision variable $x_{ijk}$ is a binary variable as set by the problem.

## 9.2 Explanation in-depth of the Genetic Algorithm

The level of detail regarding the exact methodology is not extremely clear. Based on research, I have understood and provide below the main methodology.

### *9.2.1 Applying the GRASP algorithm for machine allotment*

The authors use a GRASP based heuristic to allocate jobs to machines. The equation provided by them is as follows for the GRASP algorithm:

$$position_{jk} = \min_{p}(P_{jk} + S_{ijk} + machine\_time_{kp}) \qquad (2)$$

Where $P_{jk}$ is the processing time of Job $j$ at stage $k$, $S_{ijk}$ is the sequence dependent set up time of job $j$ if scheduled after job $i$ and $machine\_time_{kp}$ is the idle time of each machine $p$ at stage $k$

**GRASP algorithm works as follows:**

1. After all the job sequences initially that were randomized were created, the first job sequence is considered for allotment for machines at different stages

2. Lets consider a case where we have a sequence {1,2,3,4 and 5} of jobs which have been fixed across all stages. Now GRASP will assign each job to the machine as per its sequence initially. Therefore, first Job 1 will be assigned to the machine at a stage that it can be released earliest from. The machine time will reflect the earliest time a machine p is idle at that stage.

3. Therefore, at the next job allotment, which is 2, the machine time will be updated for each machine. Therefore, the machine at which job 1 was allotted now has an updated $machine\_time_{kp}$ which is the time when that machine which job 1 was assigned to is idle. Job 2 will hence be assigned to the machine where it can be released the earliest from. It is important to note $S_{ijk}$ can vary depending on which job *i*, job *j* follows.

4. Each job is subsequently assigned at the same stage.

5. Now in order to respect the job sequence that was initially determined, the earliest idle time for $machine\_time_{kp}$ at the next stage would be the time that the job 1 was completed at the previous stage as it is the first in the sequence. Job 1 is given priority for assignment before any other job in the next stage.

6. Subsequently for job 2, the machine idle times would be updated such that it is the time that job 2 was finished at stage 2 and therefore, now job 2 is assigned with priority in the next stage based on the machine times available. If a machine time had been idle when job 2 was completed in the prior stage, then machine time for job 2 would be the same time that job 2 is completed, and the machine idle time for where job 1 is assigned would be as equation 2 is described.

7. Using this approach a solution can be created which would be a feasible sequence of job-machine allotments. But by using this approach only 1 fixed schedule can be created for each job sequence.

8. The authors wished to test multiple machine allotments for each job sequence to explore the solution space, therefore in order to tackle this approach, they introduce a randomization parameter, called $\alpha$ which uses the following equation to generate a list of possible allotments called the Restricted Candidates List (RCL).

$$\beta_i \leq min(\beta) + \alpha \times [max(\beta) - min(\beta)], \forall i \in \{1, \dots, \| \beta \|\} \qquad (3)$$

9. Equation (3) helps to understand how they create varying schedules. For example, if job 2, was going to be scheduled after job 1 was, then it would dynamically calculate its release time for each machine in that stage. The earliest time it releases would be as we did in bullet point 3. In the equation this would be $\min(\beta)$, and then subsequently the machine time where it would be released the latest would be calculated which in this case is likely to be on the machine where job 1 is completed.

10. An $\alpha$ parameter is chosen that is between 0 and 1. Then the right-hand side of equation (3) would be the range of release times that is allowed for each job to be scheduled on. Therefore, the RCL will consist of any machines that job 2 can be assigned to that releases job 2 within that time frame.

11. Then the algorithm selects a job machine assignment randomly from the RCL to assign job 2 to. Therefore, this process is iteratively performed for each specific job so that job – machine assignments can be varied across each stage.

12. However, it is important to note that the sequence initially set is still respected, and therefore, no job can begin on the next stage. For example, if job 2 completes later than job 3, job 3 cannot be scheduled on the next stage. The earliest machine time $machine\_time_{kp}$ would still be the time when job 2 is completed and job 2 would be given priority for assignment. This allows the sequence to be respected but leads to machine idle times, but is done to allow for exploration of solution space.

13. Once an initial solution is constructed using bullet points 1-12, then the algorithm uses a local neighbor method to search for the nearest methods. The method chosen by the authors is changing the job allotment for all jobs at one stage.

14. Consider stage 1 to be where swap the allotment of every single job, then, job 1 is forced to be assigned to another machine rather than the machine that was originally chosen in the initial heuristic solution, then based on this, job 2 is subsequently assigned but it cannot be assigned to the machine it was initially assigned to and so on. If this subsequent allotment is based on choosing another random solution from the RCL created or if using the greedy heuristic to assign to another machine that would finish it the earliest is not provided.

15. Due to the change in allotments, the subsequent processing of each job will change depending on idle machine times and the job that was run before it in the stage where job allotments change and therefore, makespan will change. Also, the subsequent job allotments to the machines in the other stages cannot change and the sequence must be respected and therefore, it is likely that machines can either start earlier or later in the next stages, again affecting the makespan.

16. GRASP will search for as many local neighbors that haven't been searched for and find the best makespan among them. Then depending on the number of GRASP iterations, a new initial solution is constructed, local neighbors are searched and makespan is returned. The best makespan out of all the GRASP Iterations is stored. This method is performed on every single job sequence that was initially constructed in the population.

### 9.2.2 Applying Selection, Crossover and Mutation for improvement in job sequences

I will be highlighting here the specifics of how the authors have described and performed subsequent improvement in job sequencing.

1. The selection of parents (Job sequences from population) is the first step in iteratively moving towards a new job sequence that can improve the solution space.

2. Using GRASP, we were able to get the best makespan for every single job sequence in the population. They use a roulette wheel method, that uses a probability distribution and selects 2 parents with higher likelihood that have lower makespan.

3. Subsequently these two parents are crossed-over using a method called Uniform Like Cross (ULX) which works as shown in the diagram from paper [11]:

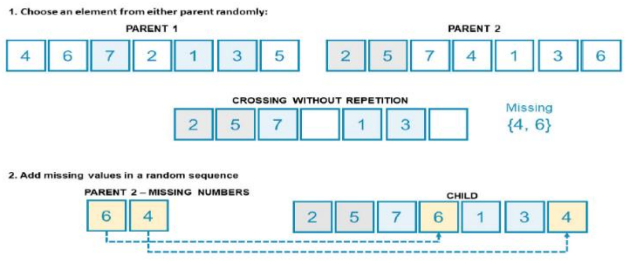

1. Choose an element from either parent randomly:
PARENT 1
4 6 7 2 1 3 5
PARENT 2
2 5 7 4 1 3 6
CROSSING WITHOUT REPETITION
2 5 7 1 3
Missing {4, 6}
2. Add missing values in a random sequence
PARENT 2 – MISSING NUMBERS
6 4
CHILD
2 5 7 6 1 3 4

**Figure 5:** Uniform Like Crossover Method

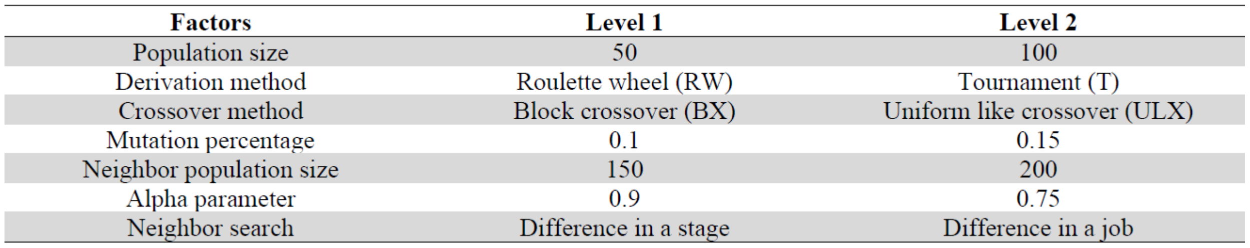
4. After the Uniform Like Crossover sequence method is performed, a probability generation is done. If the probability is within 15%, then two jobs are swapped in this new sequence with each other. This is called mutation and helps the algorithm by allowing to explore more solutions and not getting stuck in a local minima.

5. GRASP is applied to this new sequence that is generated, and the best makespan value is calculated. Based on the makespan values of all sequences in the population, the worst one is removed and replaced by this new offspring sequence.

6. Steps 1-6 are repeated over successive number of generations till iteration limit or time limit is reached.

#### *9.2.3 A note on parameter tuning*

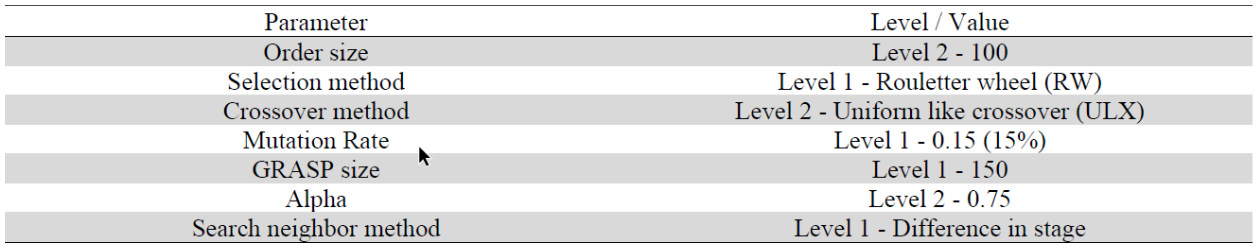
The authors have tuned different parameters of population size, selection method, crossover method, mutation percentage, local neighbor search size, alpha parameter for GRASP and the type of local neighbor search:



| Factors | Level 1 | Level 2 |
|---|---|---|
| Population size | 50 | 100 |
| Derivation method | Roulette wheel (RW) | Tournament (T) |
| Crossover method | Block crossover (BX) | Uniform like crossover (ULX) |
| Mutation percentage | 0.1 | 0.15 |
| Neighbor population size | 150 | 200 |
| Alpha parameter | 0.9 | 0.75 |
| Neighbor search | Difference in a stage | Difference in a job |

**Figure 6:** Parameters for tuning

They have generated various combination of each of these factors and tested it using a method called Taguchi's method which measures uses a measurement of tuning called the S/N (Signal to Noise) ratio that measures the variation in answers that are received for each instance. For our objective, which is minimization of the makespan, a lower S/N ratio is preferred. Based on testing they found the combination of factors that gives the least value across various test instances are as follows:



| Parameter | Level / Value |
|---|---|
| Order size | Level 2 - 100 |
| Selection method | Level 1 - Rouletter wheel (RW) |
| Crossover method | Level 2 - Uniform like crossover (ULX) |
| Mutation Rate | Level 1 - 0.15 (15%) |
| GRASP size | Level 1 - 150 |
| Alpha | Level 2 - 0.75 |
| Search neighbor method | Level 1 - Difference in stage |

The number of parameters tested are very minimal, especially considering the factors of alpha, mutation rate, population size and local neighbor size which are continuous variables. There is more potential to improve this method with a wider parameter tuning search process.

### 9.3 Explanation of Simulated Annealing in Depth

#### *9.3.1 The SPTCH and Machine Allotment Methodology for initial solution generation:*

- In order to generate an initial solution, the authors use a Shortest Processing Time Cyclic Heuristic, to generate an initial feasible sequence for our algorithm to start processing.

- At the first stage, the processing times of each job are calculated, based on this following equation:

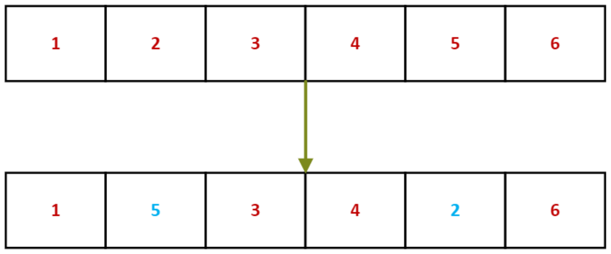
$$\tilde{p}_i^t = p_i^t + \min s_{ij}{}^t$$

Where the modified processing time $\tilde{p}_i^t$ is calculated by the processing time $p_i^t$ which is the processing time of job *i* at stage *t,* with addition of the minimum sequence dependent time it has across all jobs *j.* This is the least processing time for the job at that stage if it succeeded a particular job. Here t would be 1, because this heuristic is only applied at the first stage. This equation is independent of the machine times. The sequence is simply determined by which can be processed the quickest account for sequence dependency.

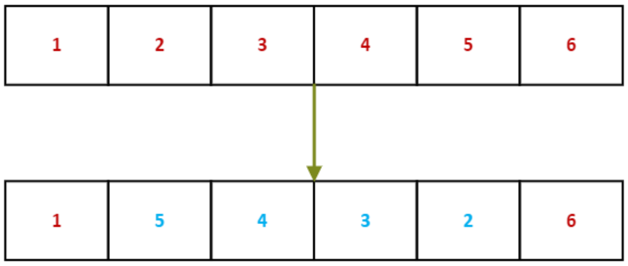
- Once this list is created, the jobs are sequenced in a non-decreasing processing order.
- Depending on the sequence, they will be assigned to the machines that can process them in the quickest time depending on each part before it. Therefore, if the sequence is (1,2,3,4) and we have 3 machines, job 4 will be scheduled on the machine that can allow it to be released the quickest. This accounts for the processing times of the previous jobs as well the sequence dependent time for job 4, after each of these jobs.
- They have not provided an equation for this but this is how it can be represented by the same equation as GRASP, however, now the idle time of the machine would simply be the time that the last job sequenced on it is completed rather than when the job in the sequence can move to the next stage, as they have assumed unlimited buffer.
- The algorithm needs to be careful that the machine time on next stage is such that it accounts for the time of completion of the job at the previous stage, otherwise the job could be assigned to the next stage without it having completed in the previous stage. So, the machine time for that job at the next stage ≥ time of completion in the previous stage. It is greater than or equal to because jobs may already be queued on that machine. However, there is definitely freedom in the sequence of the jobs unlike GRASP, because the time of completion had to be in the order of the fixed sequence but here it is dynamic.
- On the next stage, the jobs are sequenced according to the time of their completion in the previous stage, therefore, if job 2 completed before job 1 in the previous stage, and the rest of them finished in the same order they were assigned, the sequence becomes (2,1,3,4).
- The machine allotment occurs based on the same method, which schedules the jobs to the machines that allows it be completed the first. This process is dynamic, as after job is assigned, for the next job, the times of completion on each machine changes, due to the addition of the previous job to the queue.
- This process is continued, and an initial feasible sequence is created and then the makespan for this sequence is calculated.

### *9.3.2 Local Neighbor searches and Inverse Moving Operators*

- In order to search for local neighbors, the paper has used a pairwise moving operator algorithm. Their visual depiction for the inverse moving operator is not the best expressed, so I drew a similar example of how both of these moving operators work. The diagram below shows how a pairwise moving operator works



- After the initial solution is generated, the sequences of the jobs are swapped as per the picture above. This only occurs at one stage, while each of the next stages and previous stages sequences remain the same.
- Therefore, now that the sequence of the jobs is swapped, the machine allotment heuristic works in the same way that was initially selected but with allotting the jobs to the machines in a different order, in that particular stage.
- This method has the potential to improve a solution because even if the next stages sequences remain the same the next job in the sequence may be able to start sequencing earlier at the next stage than it was, because it was able to get completed earlier in the previous stage based on sequence dependent times and subsequent jobs before it.
- The sequence of the subsequent stages must remain the same, so a job even though it finished earlier cannot be assigned before the other jobs before have been sequenced. However, it could start at a lesser time than before, if swapping those jobs affected jobs in the middle of the sequence in the previous stage as well, because now they can start being processed earlier at the next stage.
- Therefore, now the machine ready times for each job changes in the same and subsequent stages, affecting makespan. It is important to know that the machine time for any part must atleast be the time that job 1 is completed in the previous stage (If job 1 is swapped in the previous stage), but the next stage's jobs start with job 1.
- Therefore, if the neighbor is a better solution, then now this neighbor is our new solution on which the pairwise moving operator will work on. The conditions of the sequencing must still remain true however depending on this new sequence. This is how the neighborhood search works for the algorithm, gradually trying to push it into more promising areas.
- Once all the neighbors are searched for each temperature, we must now repeat the entire process again from the first temperature itself to avoid local optimums and explore new regions of the solution space, but now we choose the best solution we got at the previous repeat from all the possible neighbors we generated, allowing us to explore potentially better solution spaces and sequences.
- However, the authors state that by just working with the best solution we obtained in the previous stage, we could essentially be stuck in a local optimum so in order to avoid this, they apply this heuristic called inverse moving operator to one of stages' sequences.



- The diagram shows that by applying the inverse moving operator, we alter the sequence between two moving points that we selected in reverse.
- This now becomes our new initial sequence over which the local neighbors and the temperature changes occur.

### *9.3.3 The Simulated Annealing Probability Distributions and Temperature Cooling*

- Once we have constructed an initial solution, the next step is to decide whether or not to accept a solution that is worse in makespan value than the solution we were at before.

- In the paper, however the authors have not provided their initial temperature selection or their cooling schedule but one thing for certain is that each level probability reduces.
- The probability distribution used by the authors is the metropolis probability criterion with the formula:

$$\exp\left(\frac{-\Delta s}{T}\right)\ or\ e^{\left(\frac{-\Delta s}{T}\right)}$$

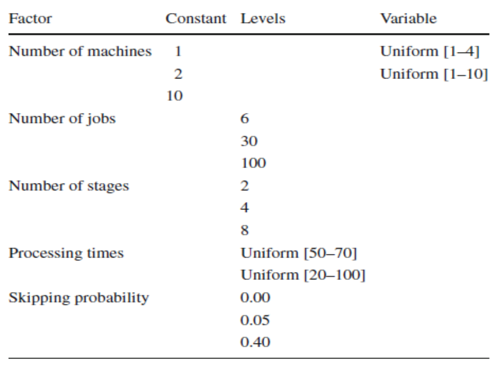
In this formula $\Delta s$ represents the difference between the neighbor solution and the initial solution. The value of T represents the temperature for that particular stage.

- If this $\Delta s$ value is negative, meaning that the neighbor solution is better than the initial solution, the value outputted will be greater than 1 and so it treats the probability as 1 and accepts the solution.
- Subsequently, if the value is positive, then the probability of acceptance depends on the difference between the values and the temperature. So if the difference between the solutions is lower and the temperature is lower the probability of acceptance would be higher.
- Now, when a neighboring solution is evaluated, its probability of acceptance is calculated, and then a random number is generated between 0 and 1. If this number is below the probability, then the solution will be accepted, and the worse solution becomes the neighbor.
- With each subsequent stage, the temperature reduces, hence the probability of acceptance becomes lower, pushing the algorithm to not choose worse solution as often.

### *9.3.4 Testing Combinations:*

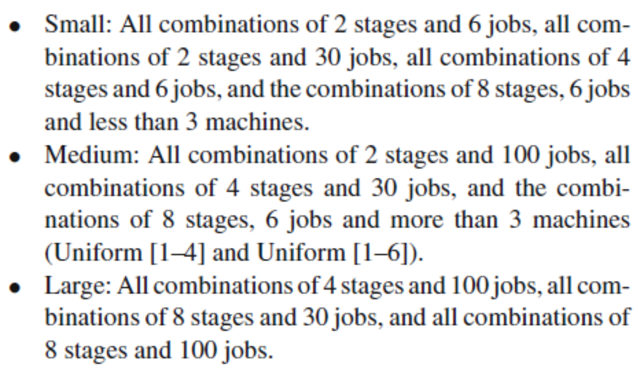
In order to test the effectiveness of NSA, the authors have tested various combinations of problems scenarios for NSA vs RKGA vs IA. The figure below represents the different parameters out of which combinations were formed (Setup times are uniform between 12 to 24). Then once these combinations are formed, 10 test instances are generated based on the distributions for the same combination for a total of 2520 test sets

They have excluded the combination of 6 jobs and 10 machines. 6 jobs and 1-10 machines per stage is changed to 6 jobs and 1-6 machines to ensure the number of jobs are greater than number of machines otherwise the problem is trivial. They have then grouped all of the instances in small, medium and large sized instance to see the effectiveness of the algorithm in distinct grouping cases, the results of which are discussed in **section 6.**



| Factor | Constant | Levels | Variable |
|---|---|---|---|
| Number of machines | 1 | | Uniform [1–4] |
| | 2 | | Uniform [1–10] |
| | 10 | | |
| Number of jobs | | 6 | |
| | | 30 | |
| | | 100 | |
| Number of stages | | 2 | |
| | | 4 | |
| | | 8 | |
| Processing times | | Uniform [50–70] | |
| | | Uniform [20–100] | |
| Skipping probability | | 0.00 | |
| | | 0.05 | |
| | | 0.40 | |

**Figure 2:** Different Combinations of Parameters



- Small: All combinations of 2 stages and 6 jobs, all combinations of 2 stages and 30 jobs, all combinations of 4 stages and 6 jobs, and the combinations of 8 stages, 6 jobs and less than 3 machines.
- Medium: All combinations of 2 stages and 100 jobs, all combinations of 4 stages and 30 jobs, and the combinations of 8 stages, 6 jobs and more than 3 machines (Uniform [1–4] and Uniform [1–6]).
- Large: All combinations of 4 stages and 100 jobs, all combinations of 8 stages and 30 jobs, and all combinations of 8 stages and 100 jobs.

**Figure 4:** Different test instances

### *9.3.5 A note on Parameter Tuning*

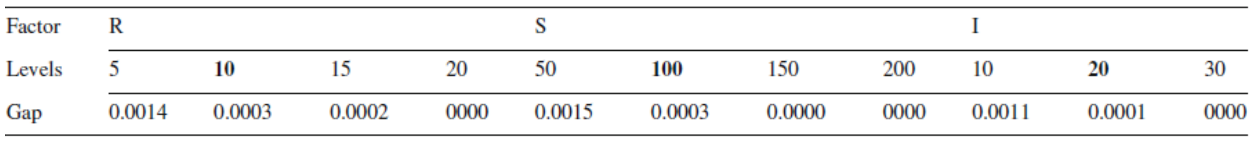
The authors have recognized the need for tuning the parameters of the number of neighbor searches, the number of stages and the number of repeats. They tried various combinations of repeats (5, 10, 15 and 20), Number of stages (50, 100, 150 and 200), Number of iterations (10,20,30).

- A total of 48 instances were run on different problem sets with each combination of these parameters.

- For all combinations of R where the value is the same, they calculate the average makespan, for all instances and all combinations
- They repeat this for all instances where a different value of R is used, but the R is the same within those solutions, and then they calculate the best average makespan, and compare the rest of the makespan to the factor of the makespan which gave the best average value, to see how different parameters fare.

$$\frac{R_{factor} - R_{best}}{R_{best}}$$

- They would do this for all combinations for stages and iterations as well and find the value at which they performed the best and how other values fared.



| Factor | R | | | | S | | | | I | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Levels | 5 | **10** | 15 | 20 | 50 | **100** | 150 | 200 | 10 | **20** | 30 |
| Gap | 0.0014 | 0.0003 | 0.0002 | 0000 | 0.0015 | 0.0003 | 0.0000 | 0000 | 0.0011 | 0.0001 | 0000 |

The table above is in the paper and they provide the parameters at which each parameter worked. They chose I = 20, rather than 30 which performed, better for what I think was to reduce the computational load as the solution was not too far away in terms of being better. They did the same R and S.

- This does give rise to the question as to how much their performance would have improved with even higher number of values because for each solution, the higher the number of iterations, stages and repeats, the better the solution.
- This gives opportunity to focus on research in this field in the future to see how increasing parameters does in terms of the solution quality itself considering the increase in computational power since 2010 when this paper was published.