5-2022

A Study of Scheduling Problems with Sequence Dependent Restrictions and Preferences

Nitin Srinath
nsrinat@clemson.edu

A study of scheduling problems with sequence dependent restrictions and preferences

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Nitin Srinath
May 2022

Abstract

In some applications like fabric dying, semiconductor wafer processing, and flexible manufacturing, the machines being used to process jobs must be set up and serviced frequently. These setup processes and associated setup times between jobs often depend on the jobs and the sequence in which jobs are placed onto machines. That is, the scheduling of jobs on machines must account for the sequence-dependent setup times as well. These setup times can be a major factor in operational costs. In fabric dyeing processes, the sequence in which jobs are processed is also important for quality, i.e., there is a strong preference to run a certain type of job after another. There are also preferences associated with scheduling jobs on specific machines. Along with these preferences, there are also sequence-dependent restrictions, i.e., certain jobs are not allowed to immediately follow certain others on the same machine. In this dissertation, a mixed-integer linear programming (MILP) formulation is first developed to schedule jobs onto the machines while accounting for the setup times and the sequence-dependent restrictions. The newly introduced preferences have been modeled as two new objectives along with traditionally used objectives such as makespan, lateness, and total setup times. This MILP is found to be inadequate in solving some of the objectives. So, improvements are made to this formulation in the form of the addition of valid inequalities and solving some underlying separation problems to obtain tighter bounds. The different versions of the MILPs are compared for computational times and optimality gaps obtained on the multiple objectives. The improved MILPs perform significantly better on all objectives and hence are more usable as well. But, they also are found to be useful only for a certain level of problem sizes, beyond which their ability to obtain optimal solutions is severely hampered by the curse of dimensionality. To tackle larger problems, multiple construction heuristics have been developed/ adapted from literature. Some of these heuristics are basic job-placement rules like SPT (Shortest processing time) or EDD (earliest due date) which have been modified to handle the sequence-based restrictions and machine eligibility rules. Other construction heuristics have been developed to incorporate the newly introduced preferences. Next, a problem size breakdown method has been developed. This heuristic method incorporates a MILP to remove the fewest possible edges in the machine eligibility graph so that a larger problem can be broken down into independently solvable smaller problems solved to optimality. These heuristic methods are compared against the optimal solutions for small problems for solution quality on all five objectives. They are also compared against each other for larger problems. The MILP-based heuristic is also tested for computational performance. From the comparison results, it was seen that the methods developed so far are unable to tackle all the objectives simultaneously, and produce multiple solutions so that the user can evaluate trade-offs among the objectives. So, two popular multi-objective meta-heuristic frameworks have been adapted to tackle the new objectives as well as provide multiple Pareto-optimal solutions. Within each of the two methods, the job placement has been performed by two approaches: a construction heuristic approach and a semi-optimal approach. The two methods along with the two job placement approaches have been compared for computation times and quality of results. The best meta-heuristic methods are found to perform better than the construction heuristics on almost all objectives consistently, at the expense of more computational effort.

Acknowledgments

This dissertation has been the most exciting and challenging endeavor of my life. It would not have been possible without the help of many people who have constantly supported me and encouraged me at all times. I am extremely grateful for all the help and support I have received, both technical and moral.

Firstly, I thank my advisor, Dr. Kevin Taaffe, without whom this work would not have been possible. He has truly been a pillar of strength. If I were to go back and pursue the Ph.D. again, I would do it under him again.

Next, I thank Dr. Mary Beth Kurz, who has helped me immensely with all the work relating to construction heuristics and meta-heuristics. I would also like to thank Dr. Herve Kerivin. I am very grateful for his help in improving the MILP formulations. His perspective on the problems has helped us create significantly improved solution techniques. Next, I thank Dr. Tugce Isik for being my committee member. Her inputs during the few meetings have led to the creation of some of the best methods developed in this dissertation.

Next, I would like to thank Ozan Yilmazlar, my fellow doctoral candidate, who has contributed immensely to the project and the publications. I would also ike to thank everyone in the industrial engineering department at Clemson through the years: Dr. Smith, Dr. Eksioglu, Dr. Mason, etc., for supporting me throughout this journey. The department has been wonderful to me and I am truly happy that I chose Clemson for my studies.

Finally, I would like to thank my parents, my grandmother, my girlfriend, my aunt, uncle, and my friends who have stayed by my side during my ups and downs and have been constant pillars of support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Sequencing and scheduling decisions play a critical role in today's industries, whether it is manufacturing, transportation, energy, or other service industries. Effective scheduling and sequencing decisions are necessary for optimal use of resources and labor, meeting due dates, reducing costs, maintaining quality, etc [81].

An important scheduling problem is one of sequencing jobs with sequence-dependent setup times (SDSTs). Setups are sometimes required to ensure machines are ready to run the jobs. Setups can also be viewed as loading-unloading actions within transportation problems [110]. Sometimes, the time required for these setups is dependent on the sequence of jobs scheduled. This is especially true for scheduling applications in fabric processing, wafer probing, circuit assembly, packaging, last-mile delivery logistics, pulp, and paper processing, biomass handling, shoe manufacturing, etc [7].

This research addresses a parallel machine scheduling and sequencing problem in the fabric dying industry with SDSTs, due dates, and machine eligibility constraints. Additionally, there are also sequence eligibility constraints that ensure that certain products are not scheduled after certain others, often used to reduce defects in the product. For example, white fabrics should not be scheduled right after black fabrics on the same machine, because it is very hard to remove all the black dye from the machine, resulting in spots on the white fabric. Also, there is a preference matrix among the jobs. For example, it is preferable to dye a purple fabric after a blue rather than a yellow after a blue, for similar reasons. Also, when a schedule is generated, it is preferable to have some machines run lighter fabrics consistently and others run darker fabrics. Even if they move from one shade to another, it is preferable to have a smooth transition within the shades of the products scheduled on the same machine. These sequence preferences are also set to maintain quality. These sequence preferences, scheduling preferences, and sequence eligibility constraints make this problem very unique. The preferences need to be modeled as objectives and the eligibility restrictions have to be modeled as constraints. The problem is multi-objective due to the need to reduce delays, reduce imbalance among the machines in the schedule, reduce setups and maximize preferences.

In this dissertation, we first develop a new mixed-integer linear programming formulation to tackle all the objectives and the constraints such as setup times, machine eligibility, as well as unique sequence-dependent restrictions. Two new objectives are introduced along with the makespan, lateness, and setup time minimization objectives, to handle the sequencing and scheduling preferences. This formulation is tested on all five objectives individually to understand how well it performs on each objective. It is found that the three objectives of reducing setup times, maximizing preferences, and minimizing shade inconsistencies are easy to solve for small and medium-sized problems. However, the makespan and lateness objectives are found to be harder to solve. So, improvements are made to the formulation by the addition of some variables, some valid inequalities, and by solving a few underlying separation problems to add valid cuts during the branch and bound stage of the MILP solution. Callback functions have been used to access the solutions at every node of the solution to solve the separation problems and add valid cuts. These improvements led to multiple versions of the formulations, which have been tested against each other on all the objectives. With the added inequalities and callbacks, the formulations have been improved to solve the harder objectives as well as solve the easier objectives faster, for small and medium-sized problems under two hours with less than a 10% optimality gap. For larger problems, the solution quality is severely hampered due to the curse of dimensionality. The problem is still NP-Hard for all the objectives [88].

So, in the next chapter of this dissertation, we have developed multiple construction heuristics to solve larger-sized problems. Heuristics have often been used in many scheduling applications because they provide feasible, good-quality solutions even if they do not guarantee optimality. In this chapter, four rule-based construction heuristics have been developed or adapted from literature for our problem. The heuristics are based on dispatching rules, insertion-based methods, and bin-packing-based methods.

Next, a MILP-based network algorithm has been developed to create partitions of jobs that could be run as independent problems. This network partitioning problem is also NP-hard [52], but for the size of problems we are interested in, this MILP is capable of solving to optimality or with little optimality gap in under an hour. This MILP breaks down the problem into smaller sub-problems with the least amount of information loss. However, this method does not guarantee optimality, and it is still a construction heuristic that can provide high-quality feasible solutions. All the rule-based and MILP based construction heuristics are compared against optimal solutions (for small problems) as well as among themselves for solution quality on all the objectives. However, with all these methods, we have not addressed this problem in a multi-objective sense. We have treated each objective separately up to this point.

So, in the next and final chapter of this dissertation, we have adapted two popular multi-objective meta-heuristic frameworks - Non-dominated sorting genetic algorithm (NSGA-II) [34] and multi-objective evolutionary algorithm based on decomposition (MOEA-D) [109] from literature for our problem. We have also improved upon the traditional NSGA-II framework by storing an external population of non-dominated solutions across the iterations to obtain a larger, more diverse pool of solutions. These methods are used in the hope of obtaining solutions that are not just good on one of the objectives but instead perform well (if not optimal) on all objectives simultaneously. Within these methods, the job placement has been made using two approaches - a construction approach and a semi-optimal MILP-based approach. These two frameworks along with the two job placement approaches have been compared against each other for solution quality and computation times. These methods can generate a large number of solutions with varying levels of performance on the five objectives. So, they have been compared using both single-objective as well as multi-objective metrics to see which method and job placement approach is the overall best.

Chapter 2

Literature Review

The literature in scheduling can be partitioned in many ways [1]:

1. Deterministic or stochastic based on the variability of the parameters involved,
2. Dynamic or static based on how the parameters and conditions change through time,
3. Based on the type of machine settings - single machine, parallel machine, flow shop, etc.,
4. Based on the need for setup times - sequence-dependent or independent,
5. Based on objectives - minimizing total completion time, minimizing makespan, minimizing setup times, minimizing costs, minimizing tardiness or lateness, etc.,
6. Based on constraints they need to incorporate such as setup times, machine eligibility, due dates, etc.,
7. Based on the type of methods employed: exact methods such as MILPs, Branch and bound (BnB) or branch and cut techniques, or approximate methods such heuristic algorithms, etc.

The problem that we address in this project is a parallel machine, deterministic, static scheduling problem with constraints of machine eligibility, due dates, and sequence-dependent setup times. The objectives of the methods developed are to reduce setup time, reduce makespan, reduce lateness and obtain schedules that maximize the sequence and schedule preferences.

There are several papers in the literature that have addressed the problem of sequence-dependent setup times on parallel machines. The following literature review is structured based on the methods used (both exact and heuristic) as well as by meeting objectives within the methods.

Most authors proposing exact methods for scheduling problems with SDSTs have developed MILPs and attempted to solve them using commercial packages such as CPLEX or Gurobi. Others have developed branch and bound algorithms which are fed a partially feasible solution before implementation.

Most of the papers that have proposed MILPs have either minimized tardiness [8, 73, 79, 58, 19] or reduced the makespan [61, 12, 47, 49, 50, 45, 3, 19]. The sum of completion times of all jobs (total completion time) is also a commonly-used objective [85, 38, 43, 98, 19]. [4] and [96] have proposed MILPs with weighted sums of earliness and tardiness as objectives.

While earliness and tardiness are seen from a customer standpoint (due-date based), makespan or total completion time objectives are resource-conserving approaches [1]. To address these competing goals in a scheduling context, a common approach is to introduce a multi-objective MILP. [42] uses a weighted sum of total completion time and total tardiness. [94] has used the same two objectives but minimizes the objectives one after another in lexicographical order. [67] uses a goal programming approach to multi-objective problems where one of the objectives is converted as a constraint while optimizing over the other. This paper uses makespan and total tardiness as the two objectives. [84] uses an unweighted sum of tardiness and makespan as a single objective. [102] uses a similar approach but instead uses a weighted sum of the total number of late jobs and makespan as objectives.

While these papers address the importance of having jobs ready on time (tardiness) or balancing the available resources (makespan) in a sequence-dependent setup environment, they do not consider reducing the setup time or cost as a primary objective. Setup times are included within the constraints to ensure that the schedules are feasible. In that direction, a vehicle routing problem-based MILP has been developed to reduce setup times [35]. They use a sum of total tardiness and total setup time as their objective. But in this case, the setup times are dynamic and are used as variables that depend on past sequences and are not determined by what job runs after the other. In most cases, the setup times between jobs are considered as parameters (s_{ij}) as the setup time between job i and job j if j is immediately processed after j. Only one paper [53] has proposed a MILP model to minimize a weighted sum of setup times and tardy times, but they have concluded that the model is not tractable for any more than 10 jobs.

Those who have not used MIPs/MILPs with solvers have developed custom branch and bound algorithms. These algorithms make use of the unique properties of the problem to develop better bounds and reach optimal solutions faster. [13] uses a BAB algorithm to minimize tardiness and [40] have used a BAB algorithm to minimize total completion times. [84] have used a BAB algorithm along with their MILP formulation to minimize a weighted sum of tardiness and makespan. However, the method developed by [84] can only perform well for 30 or fewer jobs. The BAB algorithm at its worst can evaluate all the possible solutions to a combinatorial problem. Also, the computational efficiency of a BAB algorithm depends on the starting solution fed to it and its ability to compute better bounds. Branch and bound and branch and cut algorithms are generally used by MILP solvers as one of the steps in solving MILPs.

In the last ten years, exact methods have not been discussed extensively in the literature of scheduling due to their observed ineffectiveness [1]. It is evident from the literature that exact methods currently available are not tractable for large sets of jobs. So, most authors also have developed a plethora of heuristic or approximate algorithms which are computationally more efficient than exact methods while guaranteeing solutions that are at least close to optimality. In the next section, we introduce many heuristic models that have been used to tackle parallel machine scheduling problems with sequence-dependent setup time constraints.

2.1 Construction Heuristics

Heuristic methods are step-by-step algorithms that provide feasible and sometimes optimal solutions to problems. Having a problem-specific set of rules to schedule jobs helps to design heuristics in some cases. [56, 75, 104, 21] have used specific dispatching rules are used to schedule jobs and minimize a single objective such as tardiness [75, 104] or a mix of objectives such as late jobs, completion times, etc., [56], or maximize throughput of jobs and maximize schedule robustness [21].

The commonly used basic dispatching rules are SPT (shortest processing time), W-SPT (Weighted SPT), EDD (Earliest due date), LPT (Longest processing time), etc. There are also more complex dispatching rules such as ATC (Apparent Tardiness Cost) [101], COVERT [24] and MODD (modified operation due date) [14] which are specifically designed for minimizing weighted tardiness. SPT is commonly used to reduce total completion times while LPT is commonly used to reduce makespan. For a single machine scheduling problem with setups and the aim of reducing weighted tardiness, the ATCS (ATC with setups) rule has been developed [62]. [63] proposed a heuristic for a parallel machine problem with SDSTs to obtain a starting solution for a simulated annealing procedure using the ATCS rule with an estimated makespan as input. Their objective was to reduce weighted tardiness as well. [27] has also developed a heuristic based on ATCS and simulated annealing for a batch scheduling problem with parallel machines and SDSTs. [68] have proposed a heuristic to minimize makespan in which the dispatching rule uses an index based on setup times on the machine and similarity between the jobs. Since the similarity between the jobs affected the setup times between them, this index helped reduce setup times overall. [20] have also developed an index based dispatching rule which works within a hybrid meta-heuristic framework to maximize throughput of the schedule.[69] proposed a dispatching rule for a parallel machine problem with machine eligibility based on the LFJ (Least flexible job) order to reduce the makespan. This rule is used to create an initial solution for an ant colony optimization meta-heuristic. [30] proposed a heuristic based on a bin-packing approach where each machine is given a certain time capacity to accept jobs, and if all machines reach that capacity, then additional time is added to each machine. This method, called the multi-fit method, also aims to reduce makespan. [61] has also proposed a similar Multi-fit heuristic algorithm to minimize makespan for a problem with parallel machines, ready times, and SDSTs.

Another type of construction heuristics used is insertion-based heuristics which are extensively used to schedule tasks on to processors in computer systems. Insertion-based scheduling policies are used in many computer task scheduling applications such as in [31]. A few such insertion-based heuristics have also been used to find solutions to vehicle routing and delivery problems [76, 22]. Insertion-based greedy heuristics have been used in [108] to minimize makespan and total electricity costs. [61] has also proposed an insertion-based heuristic for a scheduling problem to reduce makespan.

Greedy algorithms have been used in [9, 70, 106] to minimize tardiness and in [54] to minimize total completion time. In [59], greedy algorithms have been used to minimize flow time in job shops with SDSTs.

With better computational power and improved MILP solvers, there has been a recent trend of using MILP based heuristics to solve these problems. [41] has proposed a MILP based heuristic for a large-scale train timetabling problem where the large problem is broken down into smaller problems. In each such small problem, trains are added in batches and these small batches are solved to optimality. Any infeasible boundary solutions are corrected by rule-based heuristics. [87] has developed an LP relaxation-based heuristic where the integer program is relaxed as an LP, and constraints added iteratively to approximate the values of the variables. [15] has proposed a decomposition-based method to minimize makespan in a flexible manufacturing environment. In this method, an initial solution is constructed using a rule-based heuristic. This solution is improved iteratively using a MILP which re-solves for a small subset of jobs from the initial solution. [99] has proposed a linear relaxation-based heuristic for obtaining electric vehicle charging schedules that minimize electricity costs. In this method, the values of the LP relaxation are rounded to the nearest integer and if the solution obtained by the approximation is infeasible, additional rules are incorporated to fix the values of variables in such solutions.

2.2 Metaheuristics

For general combinatorial optimization problems including scheduling problems, a range of methods called meta-heuristics have been developed and used. These methods rely on somewhat randomly searching through the solution space and iteratively improving on the objective. While these methods also do not guarantee optimality, they have proven to be fast and efficient in generating good-enough solutions in a large number of cases. This is evident in the large number of papers that have developed or utilized these methods. The most commonly used meta-heuristic methods are genetic algorithms, tabu search, simulated annealing, randomized neighborhood search, and ant colony/ bee colony optimization.

Genetic algorithms (GAs) vary according to the rules of mutation and crossover among the parent and children population as well as how children are added to the population. For example, in [34], a Pareto front concept is used to retain parts of the population. Genetic algorithms have been used in many papers to schedule jobs on parallel machines with sequence-dependent setup times (SDST). Similar to the literature on exact methods, papers have utilized genetic algorithms on one or more objectives in different ways. [9] and [58] have used GAs to minimize tardiness, [25, 47, 48, 49, 50, 60, 83, 100, 2, 45, 3] have used GAs to minimize makespan and [98] have used GAs to minimize total completion times.

GAs have also been used to provide solutions to multi-objective problems as seen in [23, 95] to minimize both tardiness and completion times. In [67], GAs have been used to minimize both tardiness and makespan.

Simulated annealing (SA) is a heuristic model based on the physical process of cooling to decrease its inherent energy. That translates to methods in which solutions are changed to provide better objective values. SA has been used in parallel machine scheduling problems with SDSTs to minimize tardiness [28, 64, 58] and makespan [25, 107, 45]. Simulated annealing has also been used in multi-objective problems been used in to minimize tardiness, earliness, and makespan [18, 16].

Tabu search is a method of iteratively searching the solution space while making sure newer solutions are reached by declaring some of the recently seen solutions as Tabu. Tabu search algorithms have been used in [26, 55, 75] to minimize tardiness and in [46] to minimize makespan.

Multiple meta-heuristics have been developed based on the structures and movement in ant colonies, bee colonies, bee swarms, firefly motion patterns, etc. Ant colony optimization algorithms have been used in [11, 18, 16] to minimize makespan and tardiness. Artificial bee colony algorithms have been used in [23, 72, 83, 71] to minimize the same two objectives. Particle swarm optimization algorithm is used in [97] to minimize tardiness and total completion times. Firefly algorithms [36], worm optimization [10], and symbiotic organism inspired heuristics [37] has also been developed to minimize makespan.

Some other methods such as variable neighborhood search [18, 16, 17], artificial neural networks [4], Lagrangian relaxation based heuristics [33], column generation algorithms [38] have been developed to tackle tardiness and make span objectives in scheduling parallel machines with SDST constraints.

2.3 Takeaways

From the comprehensive literature review above, it can be seen that the objective of reducing setup times alone has not been addressed. In that direction, [53] has used a genetic algorithm to reduce a weighted sum of tardiness and setup times. Other than this work, there is no evidence of work done towards minimizing setups to the best of our knowledge. This is important to address as in some applications, the setups are the most cost-adding/labor-intensive component of the process. [53] is also the only paper to have developed a MILP that minimizes setup times as an objective but they have declared that the MILP is computationally not feasible for any more than 10 jobs. Also, there has been no work done previously to address the quality-based preferences and sequencing constraints while sequencing jobs. This approach is important in scheduling applications in the fabric dyeing industry to maintain high-quality standards. The key contributions of this dissertation are to not only develop exact and heuristic/meta-heuristic methods that can effectively incorporate these new objectives and constraints, but also improve upon existing methods to provide solutions faster and closer to optimality.

Chapter 3

Mixed Integer Linear Programs

In this chapter, we first propose a MILP formulation for this problem. We introduce new parameters, variables, objectives, and constraints to tackle the preferences, and sequence-based restrictions unique to this problem. Then we propose the addition of valid inequalities to obtain tighter formulations. These improved formulations are compared against each other to determine which version of the MILP is the best in terms of computation times and optimality gaps for the different objectives.

3.1 The Story of the Chapter

When we have any optimization problem on our hands, we first always start with a linear mathematical model. If that is not possible, in many real-world applications of scheduling, transportation, supply chain, etc., we build mixed-integer linear programs. Then, we test them on our data. Usually, the optimization problem for the underlying MILP is NP-Hard. And usually, when we use commercially available solvers to solve these MILPs, we end up with very large solution times, or we are unable to solve the problems to optimality even after many days of running the solver. Even though most researchers start with a MILP, they soon give up on using a MILP to solve their problems, and quickly resort to heuristic and meta-heuristic methods. This is evident from the literature review.

In recent times, meta-heuristics have gained in popularity while MILPs have declined.

In this chapter, we have tried to improve MILP methods for our problem significantly. Like many real-world problems, the MILP we first developed also could not solve our problems. Considering that the industrial sponsor was not enthusiastic about exact solution approaches, we also explored heuristic and meta-heuristic methods. But, at the same time, we constantly wondered whether we could improve our MILP. We first developed a model that worked for all the objectives. We modeled the unique sequence-dependent constraints. This model, we called the base version - v1. Then, we tested them on all the objectives separately. As expected, we could not generate optimal solutions in a two-hour time frame (the commonly used time frame in the literature). Next, we started exploring the root relaxation solutions. When the integer constraints were removed, we observed that the values of the variables violated some logical constraints that can be added to the problem. We were able to add some of these constraints to the first version, and we called this extended formulation v1m. Then, we tested them on all the objectives. Even if v1 could solve some objectives to optimality, we wondered if the extended formulation could solve them faster, or could solve them by exploring fewer nodes in the branch and bound tree. With v1m also, we could not solve to optimality for some of the objectives. We again observed the root relaxation values and found that by adding certain variables, we could model certain constraints that could cut off the relaxation solutions. For the next step, we added the new variables and constraints and we called this version v2 or version 2. Again, we tested these versions on all the objectives. When we added the new variables, we realized that they contained a lot of information and we had other variables that were redundant. Next, we developed a formulation that tried to simplify v2 to reduce some variables and constraints. We called this v3 and tested it on all the objectives. We observed then that some objectives were still not solved to optimality for the problems we tried. Once again, we started exploring the solutions at the root as well as at some of the nodes in the tree. While we observed some constraints that could be added to the problem, we realized that the number of such constraints was very large. To find only the violated inequalities, we developed methods to solve the underlying separation problems for such constraints and solved them using callback functions at every node of the tree.

While our problem has some unique objectives and constraints, we wondered if the improvements we made on the MILP can be translated to a common scheduling problem with sequence-dependent setup times and parallel machines and commonly used objectives. Hence, for all the methods, we removed the constraints that make the problem unique and tested all versions of the formulation on a popularly used MILP.

At the end of this chapter, we summarize the results and recommend the best version of the MILP for each objective, for both our problem and the common scheduling problem. However, after this exploration and effort to improve the MILP, we believe there is still some room for improvement, and we leave this for future work.

The rest of this chapter details the objectives in our problems, the base MILP with the unique sequence-dependent constraints (v1), the extended MILP (v1m), the MILP with the additional variables and constraints (v2), the simplified MILP (v3), and the details of the separation problems solved through callbacks for all the versions. Finally, we summarize the results for each objective, and the MILP version based on the optimality gap after 2 hours, solution times, and a few other parameters.

3.2 Methods

In this section, we first describe a common parallel machine scheduling problem. We detail the sets, parameters, variables, and constraints, including two additional constraints that accommodate scheduling/sequencing preferences. Next, we propose additional inequalities introduced to the problem at each stage. Then, we present the inequalities added as cuts during the branch-and-bound stage of the solution process and the algorithms used to obtain the violated inequalities. Finally, we propose an alternative formulation to solve the same problem.

3.2.1 Sets and Parameters

The sets in this formulation include the set of jobs I and the set of machines J. Let n and m denote the number of jobs and machines, respectively, that is, $n = |I|$ and $m = |J|$. We also define the sets $I_i = I \setminus i$ for every job $i \in I$ and $J_j = J \setminus j$ for every machine $j \in J$.

The parameters for the formulations include machine eligibility of job $i \in I$ on machine $j \in J$.

$$\alpha_{i,j} = \begin{cases} 1 & \text{if job } i \text{ can be dyed on machine } j, \\ 0 & \text{otherwise,} \end{cases}$$

and setup time β_{i_1,i_2} required between distinct jobs $i_1 \in I$ and $i_2 \in I$. For every job $i \in I$, are also given its processing time p_i, its due date d_i, and its shade value $s_i \in [0, 100]$. The shade values of the jobs are used to group them into shade groups. Each job $i \in I$ belongs to a shade group s_i^g that can be any of the four groups: Very dark (1-25), dark (26-50), light (51-75), very light (76-100). This information is as used by [90].

3.2.2 The base formulation - v1

This section describes a commonly used MILP formulation to solve a scheduling problem with parallel machines, SDSTs, machine eligibility, and due dates.

3.2.2.1 Variables

The binary variables for this formulation include the nm scheduling variables

$$x_{i,j} = \begin{cases} 1, & \text{if job } i \text{ is processed on machine } j \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in I, j \in J,$$

$2nm$ start and end variables

$$x_{i,j}^{strt} = \begin{cases} 1, & \text{if job } i \text{ is the first job on machine } j \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in I, j \in J,$$

$$x_{i,j}^{end} = \begin{cases} 1, & \text{if job } i \text{ is the final job on machine } j \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in I, j \in J,$$

and $n(n-1)$ sequencing variables

$$y_{i_1,i_2} = \begin{cases} 1, & \text{if job } i_1 \text{ is processed immediately before } i_2 \\ 0, & \text{otherwise} \end{cases} \qquad \forall i_1, i_2 \in I, i_1 \neq i_2.$$

The n continuous variables include finish time c_i for every job $i \in I$. Depending on the considered objective functions, we might add some variables that we introduce later.

3.2.2.2 Constraints

The constraints ensure that the solutions returned from solving the formulation are feasible. Constraint set (3.1) ensures that a job can only be processed on a machine that is eligible to process it. Constraint set (3.2) ensures that if a job is the starting job on a machine, then it is being processed on that machine. Constraint set (3.3) ensures that if a job is the ending job on a machine, then it is being processed on that machine. Constraint sets (3.2) and (3.3) link the x_{strt}, x_{end}, and the x variables. Constraint set (3.4) ensures that if a job is a starting job on a machine, then it cannot be preceded by another job. Constraint set (3.5) ensures that if a job is the ending job on a machine, then it cannot be followed by another job. Constraint sets (3.6) and (3.7) that there is only one starting and ending job for each machine. Constraint set (3.8) ensures that each job is processed by a machine. Constraint set (3.9) ensures that if a job is a starting job on any machine, then it must not be preceded by any of the other jobs. Constraint set (3.10) ensures that if a job is an ending job on any machine, then it must not be succeeded by any other job. These constraints reinforce the formulation in addition to constraint sets (3.4) and (3.5). Constraint set (3.11) ensures that the finish time for each job is greater than its processing time. Constraint set (3.12) ensures that within a pair of jobs, one job cannot follow and succeed the other job. It can either follow or succeed the other job. Constraint set (3.13) ensures that two jobs that immediately precede or succeed each other on the same machine do not overlap. Constraint set (3.14) ensures that if two jobs are on different machines, they cannot immediately precede or succeed each other on the same machine. Constraint set (3.15) ensures that if a job is both the starting and ending job on a machine, then there are no other jobs allotted for that machine. Constraint sets (3.16), (3.17), (3.18), (3.19) ensure that the binary restrictions on the variables are satisfied.

$$x_{i,j} \leq \alpha_{i,j} \quad \forall i \in I, j \in J \tag{3.1}$$

$$x_{i,j}^{strt} \leq x_{i,j} \quad \forall i \in I, j \in J \tag{3.2}$$

$$x_{i,j}^{end} \leq x_{i,j} \quad \forall i \in I, j \in J \tag{3.3}$$

$$x_{i,j}^{strt} \geq x_{i,j} - \sum_{i_1 \in I_i} y_{i_1,i} \quad \forall i \in I, j \in J \tag{3.4}$$

$$x_{i,j}^{end} \geq x_{i,j} - \sum_{i_1 \in I_i} y_{i,i_1} \quad \forall i \in I, j \in J \tag{3.5}$$

$$\sum_{i \in I} x_{i,j}^{strt} = 1 \quad \forall j \in J \tag{3.6}$$

$$\sum_{i \in I} x_{i,j}^{end} = 1 \quad \forall j \in J \tag{3.7}$$

$$\sum_{j \in J} x_{i,j} = 1 \quad \forall i \in I \tag{3.8}$$

$$\sum_{j \in J} x_{i,j}^{strt} = 1 - \sum_{i_1 \in I_i} y_{i_1,i} \quad \forall i \in I \tag{3.9}$$

$$\sum_{j \in J} x_{i,j}^{end} = 1 - \sum_{i_1 \in I_i} y_{,i_1} \quad \forall i \in I \tag{3.10}$$

$$c_i \geq p_i \quad \forall i \in I \tag{3.11}$$

$$y_{i_1,i_2} + y_{i_2,i_1} \leq 1 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.12}$$

$$c_{i_1} \leq c_{i_2} - p_2 - \beta_{i_1,i_2} + M(1 - y_{i_1,i_2}) \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2, j \in J \tag{3.13}$$

$$x_{i_1,j} + \sum_{j_1 \in J_j} x_{i_2,j_1} + y_{i_1,i_2} + y_{i_2,i_1} \leq 2 \quad \forall i_1, i_2 \in I, i_1 \neq i_2, j \in J \tag{3.14}$$

$$x_{i,j}^{strt} + x_{i,j}^{end} + x^{i_2,j} \leq 2 \quad \forall i_1, i_2 \in I, j \in J \tag{3.15}$$

$$x_{i,j} \in \{\ 0, 1\} \quad \forall i \in I, j \in J \tag{3.16}$$

$$y_{i_1,i_2} \in \{\ 0, 1\} \quad \forall i_1, i_2 \in I, i_1 \neq i_2 \tag{3.17}$$

$$x_{i,j}^{strt} \in \{\ 0, 1\} \quad \forall i \in I, j \in J \tag{3.18}$$

$$x_{i,j}^{end} \in \{\ 0, 1\} \quad \forall i \in I, j \in J \tag{3.19}$$

3.2.2.3 Problem specific constraints

In most scheduling problems with parallel machines, we see constraints to tackle machine eligibility, ensure no overlap, consider setup times, etc. We have a unique additional requirement in our problem: Jobs that are very drastically different in shade cannot be processed in immediate succession on the same machine. The limit in the difference between the shades of two jobs that can be processed in immediate succession is 60. This constraint is to ensure the quality of the output product. For example, this constraint ensures that a light yellow job is not immediately processed after a dark blue/ navy job. Even though we have setups that clean the machines, such drastic differences between the shades can still cause issues. Continuing on the same example, if a navy job is run and the machine is cleaned, there can be small residues of navy in the machine. This residue might be okay if a green or a blue job is processed afterward. But, if a light yellow job is processed, the residual can cause stains or spots, making the product unacceptable to the customer. We add two new unique constraints to the problem to model this requirement. Constraint sets (3.20) and (3.21) ensure that the difference in shade between any two jobs in immediate succession does not exceed 60.

$$(s_{i_1} - s_{i_2})y_{i_1,i_2} \leq 60 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.20}$$

$$(s_{i_2} - s_{i_1})y_{i_1,i_2} \leq 60 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.21}$$

3.2.2.4 Objectives, objective specific variables and constraints

There are five objectives in this problem: Balance the workload on the different machines, minimize delays on jobs to meet service levels, reduce setups to reduce setup costs associated with cleaning chemicals and loss of time, make preferable color transitions among jobs on the same machine and run consistent shades on machines, to maintain a good quality of output.

To balance the machines, the makespan objective is used. We use the makespan variable defined in (3.22) to record the value of the makespan of the schedule. The objective is to minimize the makespan variable as shown in equation (3.23). The makespan variable is subject to the constraint set (3.24) which ensures that the makespan is greater than or equal to the completion time of each job. In combination with the makespan objective, this constraint ensures that the makespan is the maximum value of the completion times of all the jobs.

$$c_{max} = \text{makespan of the schedule} \tag{3.22}$$

$$\text{Makespan:} \quad \min c_{max} \tag{3.23}$$

$$c_{max} \geq c_i \quad \forall i \in I \tag{3.24}$$

The lateness objective is used to minimize the extent to which jobs miss their due dates. It does not provide incentives for jobs that are completed ahead of time but only penalizes those that are late. To record the delay for each job, we introduce the lateness variables as defined in (3.25). The objective (3.26) is to minimize the sum of delays for all the jobs. Constraint set (3.27) ensures that the delay is greater than or equal to 0. This ensures that we do not provide an incentive to jobs that are finished ahead of their due dates. Constraint set (3.28) ensures that the delay for a job is greater than the difference between the completion date and due date of the job.

$$u_i = \text{lateness for job i} \tag{3.25}$$

$$\text{Lateness:} \quad \min \sum_{i \in I} u_i \tag{3.26}$$

$$u_i \geq 0 \quad \forall i \in I \tag{3.27}$$

$$u_i \geq c_i - d_i \quad \forall i \in I \tag{3.28}$$

While the makespan and the lateness objectives benefit from less time spent setting up the machines between the jobs, sometimes, setups add a significant cost to the company.

In our problem, every setup incurs costs in the form of chemicals used, and additional power spent to clean the machines. The objective, as shown in equation (3.29) minimizes the total setup time in the schedule.

$$\text{Setup times:} \quad 1/(n-m) * \min \sum_{i_1 \in I} \sum_{i_2 \in I \setminus i_1} \beta_{i_1,i_2} y_{i_1,i_2} \tag{3.29}$$

The color preference objective (3.30) has been introduced by [90] to handle the transition of colors in the scheduling of jobs. For example, it is preferred to schedule an orange-colored fabric right after a red one instead of a blue one.

$$\text{Average Color Preference:} \quad 1/(n-m) * \max \sum_{i_1 \in I} \sum_{i_2 \in I \setminus i_1} \gamma_{i_1,i_2} y_{i_1,i_2} \tag{3.30}$$

The average shade difference objective has been introduced by [90] to ensure consistent transitions of shades on each machine.

To formulate this objective, two new sets of variables, as described in (3.31) and (3.32) have been introduced.

$$s^d_{i_1,i_2} = \text{shade difference between successive jobs } i_1 \text{ and } i_2 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.31}$$

$$s^{gd}_j = \text{ shade group difference from first to last job on machine j} \quad \forall j \in J \tag{3.32}$$

This objective (3.33) is formulated as a sum of two parts: The first part handles transitions among the jobs with respect to the shades, while the second part handles the shade differences on a machine from the first to the last scheduled job on that machine.

If two distinct jobs i_1 and i_2 are scheduled on the same machine, i_2 immediately after i_1, then their shade difference is $s^d_{i_1,i_2}$. For the entire schedule, There are $n-m$ such pairs of jobs that are run in immediate succession. For each such pair of jobs, the maximum possible difference in the shade is 100. Also, since this part contributes half to the objective, the denominator becomes $200(n-m)$.

The second part of this objective rewards schedules with machines consistently running similarly shaded fabrics. For example, if the first job scheduled on a machine is very dark and the last job on the same machine is light, then the shade group difference for that machine is $3-1=2$. Let the shade group difference for every machine j be s_j^{gd}. Considering that there are m machines and the maximum shade group difference from the first to the last job is 3, and this part contributes half to the objective, the denominator for this part is $6m$.

$$\text{Average Shade Consistency:}\quad \min \sum_{i_1 \in I} \sum_{i_2 \in I \setminus i_1} s^d_{i_1,i_2}/200(n-m) + \sum_{j \in J} s_j^{gd}/6m \qquad (3.33)$$

To ensure that the values for the shade difference and shade group difference variables are feasible, constraints (3.34) to (3.37) have been introduced. Constraint set (3.34) ensures that the shade difference between a pair of jobs in immediate succession on the same machine is recorded by the shade difference variable, and constraint set (3.35) ensures that the difference is greater than 0. We do this because, for our problem, a shade transition from a lighter shade to a darker shade is preferred over a transition from a darker shade to a lighter shade. We penalize the transitions that go from dark to light to account for this. However, we have already introduced constraints (3.20) and (3.21) that ensure that there are no drastic differences either way. Constraint sets (3.36) and (3.37) ensure that the shade group difference between the first and last jobs on each machine is captured by the shade group difference variables.

$$s^d_{i_1,i_2} \geq (s_{i_2} - s_{i_1})y_{i_1,i_2} \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.34}$$

$$s^d_{i_1,i_2} \geq 0 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.35}$$

$$s^{gd}_j \geq \sum_{i \in I} s^g_i x^{strt}_{i,j} - \sum_{i \in I} s^g_i x^{end}_{i,j} \quad \forall j \in J \tag{3.36}$$

$$s^{gd}_j \geq \sum_{i \in I} s^g_i x^{end}_{i,j} - \sum_{i \in I} s^g_i x^{strt}_{i,j} \quad \forall j \in J \tag{3.37}$$

3.2.3 Tightening the base formulation

This section presents additional valid inequalities that lead to a tighter formulation. These inequalities were identified by observing the root relaxation solutions obtained from solving the base formulation without the integer constraints. For example, in appendix A where we present the root relaxation solution for a sample problem, the first job on machine $M0$ is $J4$. The completion time for J is 5 hours. We also notice that the job $J14$ is on the same machine, with a completion time of 4 hours which is not possible. To eliminate such solutions, we introduce a few valid inequalities.

3.2.3.1 Addition of constraints in polynomial number

Constraint set (3.38) ensures that if a job is both a starting and ending job on any machine, then no other job precedes or succeeds it. It also ensures that if a job is followed and preceded by other jobs, it is neither a starting job nor an ending job on any machine. Constraint set (3.39) together with (3.13) ensures that when two jobs i_1 and i_2 are in immediate succession on the same machine, with i_2 following i_1, then the completion time of i_2 is exactly equal to the completion time of i_1 plus the setup time between the two jobs β_{i_1,i_2} plus the processing time for job i_2. Constraint set (3.40) ensures that if two jobs are in immediate succession, then the completion time of the second job must be greater than or equal to the processing times of the two jobs and the setup time between the two jobs. Constraint set (3.41) ensures that if a job i is on machine j, then its completion time must be greater than or equal to the processing time of the starting job on that machine plus the processing time of its preceding job, plus the setup time between the two jobs in immediate succession. Constraint set (3.42) ensures that if a job is the ending job on a machine, then its completion time must be greater than or equal to the sum of the processing times of all the jobs on that machine. Once these constraints were added, we call the modified formulation version 1m or v1m.

$$\sum_{j\in J}(x^{strt}_{i,j} + x^{end}_{i,j}) + \sum_{i_1\in I_i}(y_{i_1,i} + y_{i,i_1}) \leq 2 \quad \forall i \in I \tag{3.38}$$

$$c_{i_1} \geq c_{i_2} - p_{i_2} - \beta_{i_1,i_2} - M(1 - y_{i_1,i_2}) \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.39}$$

$$c_i \geq \sum_{i_1\in I_i}(p_{i_1} + p_i + \beta_{i_1,i})y_{i_1,i} \quad \forall i \in I \tag{3.40}$$

$$c_i \geq \sum_{i_1\in I}(p_{i_1}x^{strt}_{i_1,j}) - M(1 - x_{i,j}) \quad \forall i \in I, j \in J \tag{3.41}$$

$$c_i \geq \sum_{i_1\in I} p_{i_1}x_{i_1,j} - M(1 - x^{end}_{i,j}) \quad \forall i \in I, j \in J \tag{3.42}$$

After adding these constraints and observing the root relaxation values for the extended formulation (v1m), we observed some more possible valid inequalities. But, to model them, we needed to introduce additional variables as described in (3.43).

3.2.3.2 Addition of variables and constraints in polynomial number

$$w_{i_1,i_2,j} = \begin{cases} 1, & \text{if } i_1 \text{ is immediately preceeds } i_2 \text{ on j} \\ 0, & \text{otherwise} \end{cases} \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2, j \in J \tag{3.43}$$

These w variables are useful in adding additional valid inequalities, which cut off some more relaxation solutions. Constraint set (3.44) and (3.45) ensures that $w_{i_1,i_2,j}$ can be equal to 1 only if both i_1 and i_2 are on the same machine j. Constraint set (3.46) ensures that between two jobs i_1 and i_2, if i_1 follows i_2, then i_2 cannot follow i_1. Constraint set

(3.47) ensures that if i_1 immediately precedes i_2 on the same machine, then the w variable corresponding to them is equal to 1. Constraint set (3.48) ensures that the number of jobs on a machine is one greater than the number of transitions on that machine. Constraint set (3.49) ensures that the makespan is greater than the sum of processing times and the sum of setup times on each machine. Constraint set (3.50) ensures that if two jobs are in immediate sequence, then they are in immediate sequence on the same machine. Constraint set (3.51) ensures that if a job is not followed by any other job on machine j, then it must be the ending job on machine j. Similarly, constraint set (3.52) ensures that if a job is not preceded by any other job on machine j, then it must be the starting job on machine j. Constraint set (3.53) ensures that if a job is the last job on a machine, then its completion time is greater than or equal to the sum of processing times for all jobs on that machine and the sum of setup times between all jobs on that machine. Constraint set (3.54) ensures that the w variables are positive. Note that they need not be restricted to take values less than 1 because constraints in sets (3.44) and (3.45) ensure that they take values less than 1. We also need not add constraints to make the w variables binary because they will naturally be binary if x and y variables are binary. This reduces the number of binary variables in the formulation, reducing the computational effort in the branch and bound stage of the computation. But, the addition of a large number of continuous variables can affect the solution times of the linear relaxations adversely. After adding these constraints and variables to v1, we call the formulation version 2 or v2. We also added these constraints and variables to v1m, and we call this modified version of v2 as v2m.

$$\sum_{i_2 \in I_{i_1}} w_{i_1,i_2,j} \leq x_{i_1,j} \quad \forall i_1 \in I, j \in J \tag{3.44}$$

$$\sum_{i_2 \in I_{i_1}} w_{i_2,i_1,j} \leq x_{i_1,j} \quad \forall i_1 \in I, j \in J \tag{3.45}$$

$$w_{i_1,i_2,j} + w_{i_2,i_1,j} \leq x_{i_1,j} \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2, j \in J \tag{3.46}$$

$$w_{i_1,i_2,j} \geq y_{i_1,i_2} + x_{i_1,j} - 1 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2, j \in J \tag{3.47}$$

$$\sum_{i \in I} x_{i,j} = \sum_{i_1 \in I} \sum_{i_2 \in I_{i_1}} w_{i_1,i_2,j} + 1 \quad \forall j \in J \tag{3.48}$$

$$c_{max} \geq \sum_{i \in I} p_i x_{i,j} + \sum_{i_1 \in I} \sum_{i_2 \in I_{i_1}} \beta_{i_1,i_2} w_{i_1,i_2,j} \quad \forall j \in J \tag{3.49}$$

$$y_{i_1,i_2} - \sum_{j \in J} w_{i_1,i_2,j} = 0 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2 \tag{3.50}$$

$$x_{i,j}^{end} + \sum_{i_1 \in I_i} w_{i,i_1,j} - x_{i,j} = 0 \quad \forall i \in I, j \in J \tag{3.51}$$

$$x_{i,j}^{strt} + \sum_{i_1 \in I_i} w_{i_1,i,j} - x_{i,j} = 0 \quad \forall i \in I, j \in J \tag{3.52}$$

$$c_i \geq \sum_{i_1 \in I} p_{i_1} x_{i_1,j} + \sum_{i_1 \in I} \sum_{i_2 \in I_{i_1}} (\beta_{i_1,i_2} w_{i_1,i_2,j}) - M(1 - x_{i,j}^{end}) \tag{3.53}$$

$$w_{i_1,i_2,j} \geq 0 \quad \forall i_1 \in I, i_2 \in I, i_1 \neq i_2, j \in J \tag{3.54}$$

After adding these constraints and observing the root relaxation values for the extended formulation (v1m), we observed some more possible valid inequalities. For example, in one of the root relaxation solutions, we found that the values of $y_{2,4}, y_{4,7}, y_{7,2}$ was 0.96 each, and the sum of those three variables added up to 2.88. But, logically, for any subset of jobs $S \subset I$ (I being the set of jobs), the sum of all possible y variables in that subset must be less than or equal to $|S| - 1$.

3.2.3.3 Addition of non-polynomially available constraints

Proposition 1. *Let $\{J_1, J_2\}$ be a partition of J. For two distinct orders i_1 and i_2 in I, the* 2-precedence inequality

$$y_{i_1,i_2} + y_{i_2,i_1} + \sum_{j_1 \in J_1} x_{i_1,j_1} + \sum_{j_2 \in J_2} x_{i_2,j_2} \leq 2 \tag{3.55}$$

is valid.

Proof. Suppose first that $y_{i_1,i_2} + y_{i_2,i_1} = 0$. For the last two terms of the left-hand side of (3.55) are bounded from above by 1, (3.55) is satisfied.

Suppose now that $y_{i_1,i_2} + y_{i_2,i_1} = 1$. Both orders i_1 and i_2 then are assigned to the same jet, say $\overline{j}$. W.l.o.g. assume $\overline{j} \in J_1$. For $\{J_1, J_2\}$ is a partition of J, we have

$$\sum_{j_2 \in J_2} x_{i_2,j_2} = 0$$

and consequently (3.55) is satisfied. □

Notice that in the proof of Proposition 1, we haven't used the property that $J_1 \cup J_2 = J$, only that their intersection is empty. We, however, consider a partition $\{J_1, J_2\}$ of J to make the 2-precedence inequalities tighter.

Proposition 2. *Let J' be a nonempty proper subset of jets. For every order $i \in I$, the* start inequality

$$\sum_{j \in J'} x_{i,j} - \sum_{i' \in I \setminus \{i\}} y_{i',i} - \sum_{j \in J'} x^{strt}_{i,j} \leq 0 \tag{3.56}$$

and the end inequality

$$\sum_{j \in J'} x_{i,j} - \sum_{i' \in I \setminus \{i\}} y_{i,i'} - \sum_{j \in J'} x^{end}_{i,j} \leq 0 \tag{3.57}$$

are valid.

Proof. We have

$$\sum_{j\in J'} x^{\text{strt}}_{i,j} = (\sum_{j\in J'} x_{i,j})(1 - \sum_{i'\in I\setminus\{i\}} y_{i',i}),$$

That is, the left-hand side also equals 1 if and only if both terms of the right-hand side equal 1. Using the linearization technique of Glover and Woolsey (1974), we obtain the following inequalities.

$$\begin{aligned}
&\textstyle\sum_{j\in J'} x^{\text{strt}}_{i,j} \geq 0\\
&\textstyle\sum_{j\in J'} x^{\text{strt}}_{i,j} \leq \sum_{j\in J'} x_{i,j}\\
&\textstyle\sum_{j\in J'} x^{\text{strt}}_{i,j} \leq 1 - \sum_{i'\in I\setminus\{i\}} y_{i',i}\\
&\textstyle\sum_{j\in J'} x_{i,j} + (1 - \sum_{i'\in I\setminus\{i\}} y_{i',i}) - \sum_{j\in J'} x^{\text{strt}}_{i,j} \leq 1.
\end{aligned}$$

The third inequality exactly is (3.56). The proof for (3.57) is similar. □

Proposition 3. *Let i be an order in I and j' be a jet in J. Adding both the start and end inequalities associated with i and $J' = \{j'\}$ guarantee that variables $x^{strt}_{i,j}$ and $x^{end}_{i,j}$ are binary.*

Proof. In the current model, we already have the inequalities

$$x^{\text{strt}}_{i,j'} \leq x_{i,j'} \tag{3.58}$$

and

$$\sum_{i'\in I\setminus\{i\}} y_{i,i'} + \sum_{j\in J} x^{\text{strt}}_{i,j} = 1.$$

For $x^{\text{strt}}_{i,j} \geq 0$ for every $j \in J$, this last equation yields

$$\sum_{i'\in I\setminus\{i\}} y_{i',i} + x^{\text{strt}}_{i,j'} \leq 1. \tag{3.59}$$

If $x_{i,j'} = 0$ or $\sum_{i'\in I\setminus\{i\}} y_{i',i} = 1$, then (3.58) and (3.59) together with $x^{\text{strt}}_{i,j'} \geq 0$ imply

$x^{\text{strt}}_{i,j'} = 0$.

Suppose now that $x_{i,j'} = 1$ and $\sum_{i' \in I \setminus \{i\}} y_{i',i} = 0$. The start inequality

$$x_{i,j'} - \sum_{i' \in I \setminus \{i\}} y_{i',i} - x^{\text{strt}}_{i,j'} \leq 0$$

yields $x^{\text{strt}}_{i,j'} \geq 1$. From (3.58), we deduce $x^{\text{strt}}_{i,j'} = 1$. $\square$

Proposition 4. *Let i_1 and i_2 be two distinct orders in I and let j be a jet in J. For any subset $X \subset I$ of orders such that $i_1 \in X$ and $i_2 \in \overline{X} = I \setminus X$, the following* end-connectivity inequality

$$x_{i_1,j} + x_{i_2,j} - \sum_{i \in X} \sum_{\bar{\imath} \in \overline{X}} y_{i,\bar{\imath}} - \sum_{i \in X} x^{end}_{i,j} \leq 1 \tag{3.60}$$

is valid.

Proof. If $x_{i_1,j} + x_{i_2,j} \leq 1$ then (3.60) is obviously satisfied because $\mathbf{y} \geq \mathbf{0}$ and $\mathbf{x}^{\text{end}} \geq \mathbf{0}$. So suppose that $x_{i_1,j} = x_{i_2,j} = 1$. If the last order processed on jet j belongs to X, then the left-hand-side of (3.60) clearly is bounded from above by 1, and (3.60) is satisfied. If the last order processed on jet j belongs to $\overline{X}$, then there must exist an order $i \in X$ and an order $\bar{\imath} \in \overline{X}$ such that $y_{i,\bar{\imath}} = 1$ for otherwise we would have a sequence of orders being consecutively processed on jet j. $\square$

Since $\sum_{i \in I} x^{\text{end}}_{i,j} = 1$, (3.60) is equivalent to

$$\sum_{\bar{\imath} \in \overline{X}} x^{\text{end}}_{\bar{\imath},j} + x_{i_1,j} + x_{i_2,j} - \sum_{i \in X} \sum_{\bar{\imath} \in \overline{X}} y_{i,\bar{\imath}} \leq 2. \tag{3.61}$$

A similar family of inequalities can be derived by considering the x^{strt}-variables. The next proposition thus is provided with a proof for it is similar to the one of Proposition 4.

Proposition 5. *Let i_1 and i_2 be two distinct orders in I and let j be a jet in J. For any subset $X \subset I$ of orders such that $i_1 \in X$ and $i_2 \in \overline{X} = I \setminus X$, the following* start-connectivity inequality

$$x_{i_1,j} + x_{i_2,j} - \sum_{i \in X} \sum_{\overline{\imath} \in \overline{X}} y_{\overline{\imath},i} - \sum_{i \in X} x_{i,j}^{strt} \leq 1 \tag{3.62}$$

is valid. □

Since $\sum_{i \in I} x_{i,j}^{\text{strt}} = 1$, (3.62) is equivalent to

$$\sum_{\overline{\imath} \in \overline{X}} x_{\overline{\imath},j}^{\text{strt}} + x_{i_1,j} + x_{i_2,j} - \sum_{i \in X} \sum_{\overline{\imath} \in \overline{X}} y_{\overline{\imath},i} \leq 2. \tag{3.63}$$

Consider a subset of jobs $S \subset I$, where I is the set of all jobs. Even if all the jobs in the subset were placed on the same machine, and all jobs in the subset were in continuous sequence, then the sum of all possible y variables for that subset S must be less than or equal to $|S| - 1$, i.e.,

$$\sum_{i_1,i_2 \in S} y_{i_1,i_2} \leq |S| - 1 \quad \forall S \subset I \tag{3.64}$$

If we consider this scheduling problem from the viewpoint of vehicle routing problems, this inequality ((3.64)) is a classic sub-tour elimination inequality. Observe that we have no-overlap constraints in our formulation (3.13) that ensure that there are no sub-tours in any feasible solution. But, these constraints only apply when the y variables are binary. However, within root relaxation solutions, we observed from the dominant y values for each job that sub-tours were present, with some subsets of jobs violating inequality (3.64).

When observing the root relaxation values from the different versions of the formulation previously explained, we were able to find the valid inequalities (3.38) - (3.42) that could be added to the formulation before the start of the solution process, i.e., the number of such inequalities was polynomial. The problem with constraints (3.55) - (3.64) is that none of these sets of inequalities can be added in polynomial time. For example, in (3.55), it is impossible to find all the partitions of the set of machines in polynomial time. In (3.64), finding all possible subsets of the set of jobs is not possible in polynomial time. Constraints (3.55) - (3.64) could not be listed at the beginning of the model development.

However, we can find some violated families of inequalities by analyzing the relaxation solutions at every node of the branch and bound stage of the solution. When we analyze the solution and find certain inequalities violated, we can add those constraints as cuts during the branch and bound stage of the solution. This is done by using callback functions that help retrieve and analyze the relaxation solutions at every node and provide the functionality to add cuts when certain constraints have been violated. To identify which constraints have been violated, we use separation problems designed for each family of inequalities. In this section, we explore the underlying separation problems for constraints (3.55) - (3.64). These separation problems (solvable in polynomial time) are solved in the branch and bound stage of the solution through callback functions. Once the separation problems are solved, certain valid cuts are added to the problem to make the formulation tighter.

3.2.3.4 Separation Problems

Let $\mathcal{F}$ be a family of inequalities. The *separation problem* for $\mathcal{F}$ consists of, given a vector $\mathbf{x}$, deciding whether $\mathbf{x}$ satisfies all the inequalities in $\mathcal{F}$ and if this is not the case, finding an inequality in $\mathcal{F}$ violated by $\mathbf{x}$.

Proposition 6. *The separation problem for the 2-precedence inequalities* (3.55) *can be solved in polynomial time.*

Proof. Let $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$ be two non-negative vectors. Consider two distinct jobs i_1 and i_2 and the problem

$$\max\{\sum_{j_1 \in J_1} x_{i_1,j_1} + \sum_{j_2 \in J_2} x_{i_2,j_2}, \{J_1, J_2\} \text{ partition of } J\}. \tag{3.65}$$

The partition $\{J_1^*, J_2^*\}$ of J, where $J_1^* = \{j \in J : \overline{x}_{i_1,j} \geq \overline{x}_{i_2,j}\}$ and $J_2^* = J \setminus J_1^*$, is optimal to problem (3.65) and can be found in $O(|J|)$. If

$$\sum_{j_1 \in J_1^*} \overline{x}_{i_1,j_1} + \sum_{j_2 \in J_2^*} \overline{x}_{i_2,j_2} > 2 - \overline{y}_{i_1,i_2} + \overline{y}_{i_2,i_1}$$

then the 2-precedence inequality associated with i_1, i_2, J_1^*, and J_2^* is violated by vectors $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$. Otherwise, all the the 2-precedence inequality associated with i_1 and i_2 are satisfied by $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$.

Repeating this process $\frac{|I|(|I|-1)}{2}$ times (i.e., for every pair (i_1, i_2) of jobs with $i_1 < i_2$) yields a separation algorithm for inequalities (3.55) that runs in $O(|I|^2|J|)$. □

Proposition 7. *The separation problem for the start/end inequalities* (3.57)/(3.56) *can be solved in polynomial time.*

Proof. Let i be a job in I and $\overline{\mathbf{x}}, \overline{\mathbf{y}}$, and $\overline{\mathbf{x}}^{\mathrm{strt}}$ be three vectors. For $\sum_{i' \in I \setminus \{i\}} \overline{y}_{i',i}$ does not depend on set J', the separation problem for the start inequalities (3.57) associated with i reduced to solving

$$\max\{\sum_{j \in J'} (\overline{x}_{i,j} - \overline{x}_{ij}^{\mathrm{strt}}), J' \subset J\}. \tag{3.66}$$

Problem (3.66) can be easily solved. An optimal solution $\overline{J}'$ is composed of all the machines j in J for which $\overline{x}_{i,j} - \overline{x}_{ij}^{\mathrm{strt}}$ is positive, that is,

$$\overline{J}' = \{j \in J, \overline{x}_{i,j} - \overline{x}_{ij}^{\mathrm{strt}} > 0\}.$$

If the optimal value to (3.66) is less than or equal to $\sum_{i' \in I \setminus \{i\}} \overline{y}_{i',i}$, then all the start inequalities (3.57) associated with i are satisfied by $\overline{\mathbf{x}}, /\overline{\mathbf{y}}$, and $\overline{\mathbf{x}}^{\mathrm{strt}}$. If the optimal value to (3.66) is greater than $\sum_{i' \in I \setminus \{i\}} \overline{y}_{i',i}$, then $\overline{J}'$ induces a violated start inequality (3.57) associated with i.

Repeating this process $|I|$ times (i.e., for every job) yields a separation algorithm for inequalities (3.57) that runs in $O(|I||J|)$. The proof for separating (3.56) is similar. □

Proposition 8. *The separation problem for the end-connectivity inequalities* (3.60) *and* (3.61) *reduces to* $O(|I|^2|)$ *maximum flow problems.*

Proof. We prove the result for (3.60), the proof being similar for (3.61). Let $\overline{\mathbf{x}}$, $\overline{\mathbf{x}}^{\mathrm{end}}$, and $\overline{\mathbf{y}}$ be three non-negative vectors. Consider two distinct jobs i_1 and i_2 in I and a machine j in

J. We claim that problem

$$\min\{\sum_{i\in X}\sum_{\overline{\imath}\in\overline{X}} y_{i,\overline{\imath}} + \sum_{i\in X} x^{\text{end}}_{i,j} : X \subset I \text{ with } i_1 \in X, i_2 \notin X\} \tag{3.67}$$

reduces to solving a maximum-flow problem. Let $G = (V, A)$ be the undirected graph where $V = \{s, t\} \cup I$ and $A = \{(i, i') : i \in I, i' \in I \setminus \{i\}\} \cup \{(s, i) : i \in I\} \cup \{(i, t) : i \in I\}$ and $\mathbf{c}$ be the vector of $\mathbb{R}^A_+$ defined as

$$c_a = \begin{cases} \overline{y}_a & \text{if } a \in \{(i, i') : i \in I, i' \in I \setminus \{i\}\}, \\ 0 & \text{if } a \in \{(s, i) : i \in I \setminus \{i_1\}\}, \\ \infty & \text{if } a = (s, i_1), \\ \overline{x}^{\text{end}}_{i,j} & \text{if } a = (t, i) \text{ with } i \in I \setminus \{i_2\}, \\ \infty & \text{if } a = (i_2, t). \end{cases}$$

Let $S \subset V$ be a minimum s-t cut in G with capacity vector $\mathbf{c}$. Notice that S can be found using any polynomial-time max-flow algorithm in . As $c_{(s,i_1)} = c_{(t,i_2)} = \infty$, we clearly have $i_1 \in S$ and $i_2 \in \overline{S}$. Let $X = S \setminus \{s\}$. The capacity of the s-t cut S is

$$\begin{aligned} c(\delta^{\text{out}}(S)) &= \sum_{a\in\delta^{\text{out}}(S)} c_a \\ &= \sum_{\overline{\imath}\in\overline{X}} c_{(s,\overline{\imath})} + \sum_{i\in X}\sum_{\overline{\imath}\in\overline{X}} c_{(i,\overline{\imath})} + \sum_{i\in X} c_{(i,t)} \\ &= 0 + \sum_{i\in X}\sum_{\overline{\imath}\in\overline{X}} \overline{y}_{(i,\overline{\imath})} + \sum_{i\in X} \overline{x}^{\text{end}}_{i,j}, \end{aligned}$$

that is, it is equal to the optimal value of problem (3.67). If $c(\delta^{\text{out}}(S)) \geq \overline{x}_{i_1,j} + \overline{x}_{i_2,j} - 1$, then all the end-connectivity inequalities (3.60) associated with jobs i_1 and i_2 and machine j are satisfied by $\overline{\mathbf{x}}$, $\overline{\mathbf{x}}^{\text{end}}$, and $\overline{\mathbf{y}}$. If $c(\delta^{\text{out}}(S)) < \overline{x}_{i_1,j} + \overline{x}_{i_2,j} - 1$, the end-connectivity inequality (3.61) associated with jobs i_1 and i_2, machine j, and set X is violated by $\overline{\mathbf{x}}$, $\overline{\mathbf{x}}^{\text{end}}$, and $\overline{\mathbf{y}}$. □

To eliminate sub tours (to check for violations of constraint set (3.64)), [80] has provided an exact approach to solve the underlying separation problem. This method involves solving n^2 max-flow problems, where n is the number of nodes in the graph. Considering that each max-flow problem is solved in $O(n^2)$ using the Pre-flow push algorithm by [44], the overall complexity of the method is $O(n^4)$. Thus, to save computation time, we devised a heuristic algorithm (1 to identify such violated inequalities. This algorithm creates a path (sequence) using the dominant y values for each job. If the path circles back to the starting point (job), then we know there exists a sub tour. We add a cut to eliminate such a solution in such a case. This algorithm does not guarantee to find all sub tour violations. However, it computes in $O(n^2)$.

Proposition 9. *The Algorithm 1 does not find all sub tour violations.*

Proof. Consider a subset of jobs $J^* \in J = \{J_1, J_2, J_3\}$ and the fractional solution of y variables obtained from the linear relaxation as shown Figure 3.1. Assuming all constraints relating to machine eligibility are satisfied, and none of these jobs are starting or ending jobs, then these y values satisfy the constraints in the LP. However, the sub tour elimination constraint:

$$\sum_{i_1 \in J^*} \sum_{i_2 \in J^*} y_{i_1 i_2} \leq 2$$

is not satisfied, as the sum is 2.05. Our algorithm 1, which follows the path of the maximum values of the y variables from each job to the next, has failed to capture this violated constraint. Hence, this shows that the algorithm does not work to find all violated sub tour elimination constraints.

□

Algorithm 1 Subtour elimination cut generation algorithm

```
Obtain node relaxation values of all $y$ variables
for Each job $i$ in the set of jobs do
    NEXTJOB=$i$, PATH[0]=$i$, COUNT=0
    while TRUE do
        Find job $i_1$ such that $y_{NEXTJOB,i_1} > y_{NEXTJOB,i_2} \quad \forall i_2 \in I \setminus i$
        Set NEXTJOB=$i_1$, PATH[COUNT++]=$i_1$
        if $\sum_j x^{end}_{NEXTJOB,j} = 1$ then
            BREAK
        else if NEXTJOB=$i$ then
            if $\sum_{i_1,i_2 \in PATH} y_{i_1,i_2} > |PATH| - 2$ then
                Add cut: $\sum_{i_1,i_2 \in PATH} y_{i_1,i_2} \leq |PATH| - 2$
            end if
            BREAK
        end if
    end while
end for
```

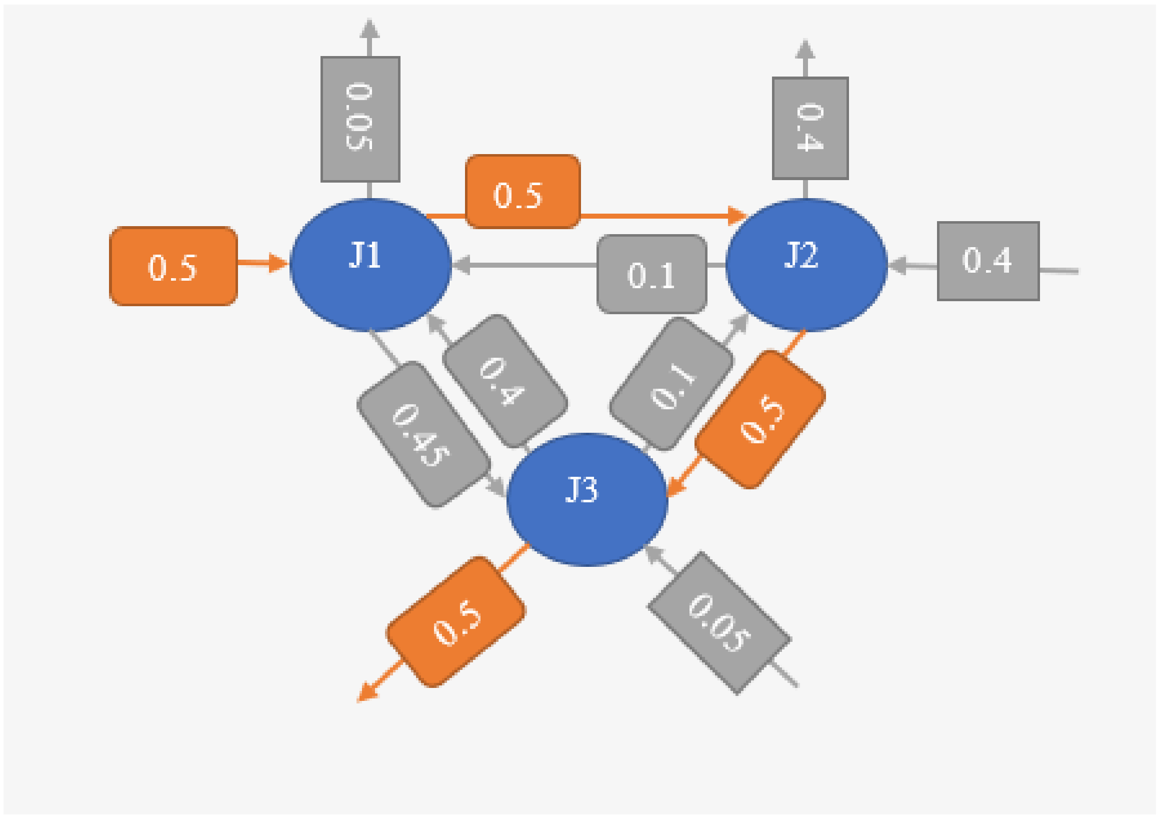

Figure 3.1: Sample scenario of fractional y values

3.2.4 Alternate formulation - v3

Since the w variables contain so much information, including the information contained in the x and y variables, it is possible to develop a formulation with just the w variables. In this section, we develop such a formulation and remove the x and y variables. Note that without the x and y variables, the w variables will have to be binary, which increases the number of binary variables, but the overall number of variables is smaller for this formulation. In this formulation, we introduce a dummy starting and ending job s_j and e_j respectively for each machine j and expand the set of jobs as shown in 3.2.4. Then, we redefine the scope of the w variables as in (3.68). The variables defined for the completion times, makespan, delays, and shade differences remain the same as the base formulation.

$I^* = I \cup \{s_j, e_j \quad | j \in J\}$

$$w_{i_1,i_2,j} = \begin{cases} 1, & \text{if } i_1 \text{ immediately precedes } i_2 \text{ on j} \\ 0, & \text{otherwise} \end{cases} \qquad \forall i_1 \in I^* \setminus e_j, i_2 \in I^* \setminus s_j, i_1 \neq i_2, j \in J \tag{3.68}$$

With respect to the objectives, the makespan, lateness, and shade differences' objectives remain the same, whereas the setup times and color preferences' objectives are defined using the w variables as shown in (3.69) and (3.70) respectively.

$$1/(n-m) * \min \sum_{j \in J} \sum_{i_1 \in I^* \setminus e_j} \sum_{i_2 \in I^* \setminus i_1, s_j} \beta_{i_1,i_2} w_{i_1,i_2,j} \tag{3.69}$$

$$1/(n-m) * \max \sum_{j \in J} \sum_{i_1 \in I^* \setminus e_j} \sum_{i_2 \in I^* \setminus i_1, s_j} \gamma_{i_1,i_2} w_{i_1,i_2,j} \tag{3.70}$$

Constraint set (3.71) ensures that if a job is not an ending job on a machine, then it must be followed by another job. Similarly, constraint set (3.72) ensures that if a job is not a starting job on a machine, then it has to be preceded by another job. Constraint sets (3.73) and (3.74) ensures machine eligibility restrictions are satisfied. Constraint set (3.75) ensures that if a job i_1 has a preceding job i_2 on machine j, then it must have a succeeding job on the same machine. Constraint set (3.76) ensures that if a job is the only job on a machine, then it has no predecessors or followers on that machine. Constraint set (3.77) ensures that within a pair of jobs, there can at most be only one predecessor and one follower. Constraint set (3.78) ensures that only one job follows the starting job on each machine. Constraint set (3.79) ensures that only one job precedes the last job on each machine. Constraint set (3.80) ensures that the makespan is greater than the sum of processing times and setup times on each machine. Constraint set (3.81) ensures that the completion time for the dummy starting job is zero. Constraint set (3.82) ensures that for each job, the completion time is greater than its processing time. Constraint set (3.83) ensures that the makespan is greater than the completion time for each job. Constraint sets (3.84), (3.85), (3.86) ensure no overlap between the jobs scheduled next to each other on the same machine. Constraint sets (3.87) and (3.88) ensure that the shade difference between consecutive jobs is less than 60 places. Constraint set (3.89) ensures binary restrictions are satisfied on the w variables.

$$\sum_{j\in J}\sum_{i_2\in I} w_{i_1,i_2,j} = 1 - \sum_{j\in J} w_{i_1,e_j,j} \quad \forall i_1 \in I \tag{3.71}$$

$$\sum_{j\in J}\sum_{i_2\in I} w_{i_2,i_1,j} = 1 - \sum_{j\in J} w_{s_j,i_1,j} \quad \forall i_1 \in I \tag{3.72}$$

$$\sum_{i_2\in I\cup e_j} w_{i_1,i_2,j} \leq \alpha_{i_1,j} \quad \forall i_1 \in I, j \in J \tag{3.73}$$

$$\sum_{i_2\in I\cup s_j} w_{i_2,i_1,j} \leq \alpha_{i_1,j} \quad \forall i_1 \in I, j \in J \tag{3.74}$$

$$\sum_{i_2\in I^*} w_{i_1,i_2,j} = \sum_{i_2\in I^*} w_{i_2,i_1,j} \quad \forall i_1 \in I, j \in J \tag{3.75}$$

$$w_{s_j,i_1,j} + w_{i_1,e_j,j} + \sum_{i_2\in I}(w_{i_1,i_2,j} + w_{i_2,i_1,j}) \leq 2 \quad \forall i_1 \in I, j \in J \tag{3.76}$$

$$\sum_{j\in J} w_{i_1,i_2,j} + w_{i_2,i_1,j} \leq 1 \quad \forall i_1 \in I, i_2 \in I \tag{3.77}$$

$$\sum_{i\in I} w_{s_j,i,j} = 1 \quad \forall j \in J \tag{3.78}$$

$$\sum_{i\in I} w_{i,e_j,j} = 1 \quad \forall j \in J \tag{3.79}$$

$$c_{max} \geq 0.5\sum_{i_1\in I}\sum_{i_2\in I}(p_1 + p_2)(w_{i_1,i_2,j} + w_{i_2,i_1,j}) + \sum_{i_1\in I}\sum_{i_2\in I}\beta_{i_1,i_2} w_{i_1,i_2,j} \quad \forall j \in J \tag{3.80}$$

$$c_{s_j} = 0 \quad \forall j \in J \tag{3.81}$$

$$c_i \geq p_i \quad \forall i \in I \tag{3.82}$$

$$c_{max} \geq c_i \quad \forall i \in I \tag{3.83}$$

$$c_{s_j} \leq c_{i_1} - p_{i_1} + M(1 - w_{s_j,i_1,j}) \quad \forall i_1 \in I, j \in J \tag{3.84}$$

$$c_{i_1} \leq c_e + M(1 - w_{i_1,e_j,j}) \quad \forall i_1 \in I, j \in J \tag{3.85}$$

$$c_{i_1} \leq c_{i_2} - p_{i_2} - \beta_{i_1,i_2} + M(1 - w_{i_1,i_2,j} \quad \forall i_1 \in I, i_2 \in I, j \in J \tag{3.86}$$

$$(s_{i_1} - s_{i_2}) w_{i_1,i_2,j} \leq 60 \quad \forall i_1 \in I, i_2 \in I, j \in J \tag{3.87}$$

$$(s_{i_2} - s_{i_1}) w_{i_1,i_2,j} \leq 60 \quad \forall i_1 \in I, i_2 \in I, j \in J \tag{3.88}$$

$$w_{i_1,i_2,j} \in 0,1 \quad \forall i_1 \in I^*, i_2 \in I^*, i_1 \neq e_j, i_2 \neq s_j, i_1 \neq i_2, j \in J \tag{3.89}$$

3.2.5 Summary of Methods

In the previous sections, we have described a base version of the formulation. Then, we added valid inequalities in stages. Note that the base version consists of the base constraints (3.1)-(3.19)(plus the problem-specific constraints for the unique problems (3.20)-(3.21)), the valid inequalities without the w variables are (3.38) - (3.42),. and the inequalities with the w variables are (3.44) - (3.54). The details of the objective functions and constraints for each version and objective is shown in Table 3.2.5.

Next, we described some valid inequalities added during the branch and bound stage of the solution in the form of cuts. These cuts (3.55) - (3.64) can be added when solving any of the versions of the formulation consisting of the x and y variables. One of the cuts we add is the sub tour elimination cuts which are added by using two different methods: a heuristic that does not guarantee all violated inequalities to be found but finds some cuts in less time and an exact method from the literature that can find all violated inequalities but takes longer to solve. To summarize, we have three versions of callback function implementation:

- Use only heuristic (c1)
- Use only exact method (c2)
- Use both heuristic and exact method (c3)

In c3, we first process the relaxation values through the heuristic, and if violated inequalities are found, we do not implement the exact method. If the heuristic does not find any violated inequalities, we call the exact method to ensure no inequalities are violated. If some cuts are found, then they are added.

Then we have v3, a different formulation that solves all objectives and provides feasible solutions. For v3 also, we implement the callback functions for only the sub tour elimination constraints. We do so by using the property of constraint (3.51), which states that $\sum_j w_{i_1,i_2,j}$ is equal to y_{i_1,i_2}. We can implement both the heuristic and the exact method of finding sub tour elimination inequalities using this property.

From a methods perspective, we analyze the performance of every version of the formulation. The versions are detailed below:

- base formulation (v1)
- v1 with callbacks (v1c1, v1c2, v1c3)
- v1 + valid inequalities without w variables (v1m)
- v1m with callbacks (v1mc1, v1mc2, v1mc3)
- v1 + valid inequalities with w variables (v2)
- v2 with callbacks (v2c1, v2c2, v2c3)
- v2 + valid inequalities without w variables (v2m)
- v2m with callbacks (v2mc1, v2mc2, v2mc3)
- formulation with only w variables (v3)
- v3 with callbacks (v3c1, v3c2, v3c3)

3.3 Computational results and discussion

This section introduces the experimental design of the tests and provides results across all formulations, solution methods, and objectives. In addition to indicating performance improvements across all forms of the SDST-P, we provide insights into the best performing method for each objective. The best version provides the least optimality gap on the solution and solves as many problems to optimality as possible. If multiple methods perform similarly or can solve all problems to optimality, we suggest the best method based on time needed to arrive at the optimal solution. If the time needed is also similar across multiple methods, we suggest a method that explores the least number of nodes in the branch and bound tree. Fewer nodes explored means that a feasible solution was obtained

Method	Makespan	Lateness	Setups	Colors	Shades
		Objective			
v1/v1c1/v1c2/v1c3	(3.23)	(3.26)	(3.29)	(3.30)	(3.33)
v1m/v1mc1/v1mc2/v1mc3					
v2/v2c1/v2c2/v2c3					
v2m/v2mc1/v2mc2/v2mc3					
v3/v3c1/v3c2/v3c3			(3.69)	(3.70)	
		Constraints			
v1/v1c1/v1c2/v1c3	(3.1)-(3.21), (3.24)	(3.1)-(3.21), (3.27)-(3.28)	(3.1)-(3.21)		(3.1)-(3.21), (3.34)-(3.37)
v1m/v1mc1/v1mc2/v1mc3	v1+ (3.38) - (3.42)				
v2/v2c1/v2c2/v2c3	v1+ (3.44) - (3.54)	v1+(3.44) - (3.54) except (3.49)			
v2m/v2mc1/v2mc2/v2mc3	v2 + (3.38) - (3.42)				
v3/v3c1/v3c2/v3c3	(3.71)-(3.89), (3.24)	(3.71)-(3.79), (3.81)-(3.89), (3.27)-(3.28)	(3.71)-(3.79), (3.81)-(3.89)		(3.71)-(3.79), (3.81)-(3.89), (3.34)-(3.37)

Table 3.1: Summary of methods

quickly and the lower bound (in case of minimization) was not far from optimal; hence, the formulation is better.

3.3.1 Design of experiments

Our problem has been inspired from a project sponsor in the fabric dying industry. The problem consists of multiple machines in parallel, and the jobs correspond to the customer orders for dyed and finished fabrics. The machines are used to dye the uncolored fabrics and add other finishing chemicals such as whiteners, fluorescent chemicals, etc. The problem sizes vary from as little as 10 jobs and 2 machines up to 50 jobs and 500 machines. Given that the production environment in the industry example called for disjoint groups of machines to support jobs that would only be assigned to a single machine group, in effect, this resulted in separable problems by machine group. We classify our problem sizes into small, medium, and large-sized problems. Small problems consist of less than 5 machines and less than 30 jobs. Medium-sized problems consist of up to 10 machines and 100 jobs. Large-sized problems consist of up to 50 machines and 500 jobs. Our research exercise was to develop usable exact methods which can tackle small and medium-sized problems. The aim was to develop methods that could reduce the optimality gap on all the objectives to under 10% within a 2-hour time window. While we wanted to improve the performance of exact methods for our specific problem, it was also of interest to us to know whether the improvements we implemented helped achieve better performance on the common scheduling problem with parallel machines, SDSTs, due dates, and machine eligibility constraints. This common problem pertains to the base constraints and base variables detailed in section 3.2.2.

We also test each version over a range of machine and job configurations corresponding to small and medium-sized problems. Finally, to ensure reduction in bias, we would like to test each possible configuration over multiple datasets.

To accomplish the requirements of the tests, the number of machines was chosen to be 2, 3, 5, 8, or 10. The number of jobs was chosen to be 20, 40, 60, 80, or 100. For each combination of jobs and machines, 5 datasets were tested (each dataset had unique machine eligibility parameters, due dates, processing times, etc.). This led to 125 datasets tested over ten versions of the formulation.

We have five objectives in our problem. Each dataset is tested over the ten versions of the formulation for all five objectives with a two-hour time limit. Each dataset is also tested with and without the unique sequence-dependent restrictions. In total there are 125 datasets * 5 objectives * 10 versions * 2 types of problems = 12500 tests.

The datasets were designed to mimic the customer orders in the industry. The job configuration was selected according to the following criteria:

- job color randomly selected from ['BLK', 'NVY', 'PUR', 'BLU', 'BRN', 'GRN', 'RED', 'ORG', 'YLW', 'WHT']
- job dye type randomly selected from ['A', 'B', 'D', 'BD'], where A is acidic, B is basic, D is disperse, and BD is basic disperse
- job shade value is selected from a uniform distribution based on its color: 'BLK': [10, 20], 'NVY':[15, 25], 'PUR': [20, 50], 'BLU':[25, 70], 'BRN':[20,60], 'GRN':[25, 70], 'RED':[25, 70], 'ORG': [40, 70], 'YLW':[60, 85], 'WHT':[85, 100]
- job fluorescence is select randomly as yes or no.
- job processing time is chosen from a uniform distribution between 3 and 15 hours
- job due date is chosen from a uniform distribution between -1 week and 2 weeks.
- the machine eligibility for a job-machine pair was selected randomly as yes or no
- the setup times between jobs was determined by a set of rules provided by the sponsor
- the rules were based on the colors, dye types, shades, and presence of fluorescent chemicals.

The tests were conducted on an eight-core CPU with 15GB RAM using the Clemson Palmetto cluster.

3.3.2 Makespan

On the makespan objective, the base formulation v1 performed extremely poorly. This was the worst-performing objective of all the five objectives, with an average gap of over 80% as seen in Figure 3.2. Using different callbacks to add additional cuts during the branch and bound stage of the solver makes the performance with respect to the average gap worse than the base formulation. This is evident from Figure 3.2, where v1c1, v1c2, and v1c3 all perform poorly with average gaps over 85%.

With the addition of the valid inequalities (3.38) to (3.42), the performance on the makespan objective improves significantly with respect to the average gap. This can be seen in Figure 3.2, where the average gap for version v1m is less than 60%. For this version also, the addition of callbacks makes the performance of the method worse.

The best performing method for this objective is v2, with the addition of the 'w' variable-based inequalities improving the gap significantly and reducing it to less than 5% on average. The addition of callbacks or the addition of the inequalities (3.38) to (3.42) does not improve the performance with respect to the average gap. However, when the 'w' variables are added as binary variables, the performance of the methods worsens, as seen in Figure 3.2 with v3 performing poorly in comparison.

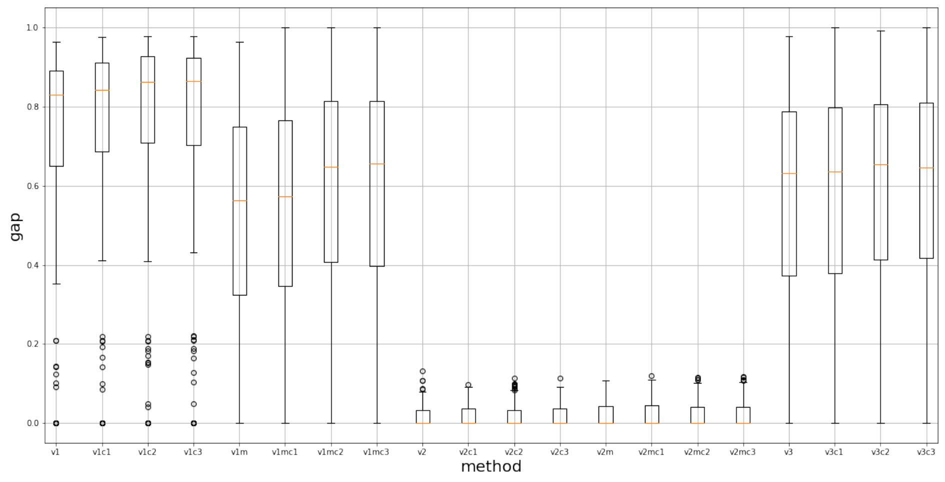

Figure 3.2: Optimality gap - makespan

The lower bound that the formulation was obtained was the most significant factor in determining the gap. In Figure 3.3, we provide the average values of upper bounds (objective values) and lower bounds obtained at the end of the solver runs from each method. The blue box plots represent the objective values, while the red plots represent the lower bound values. From the figure, it is evident that the objective values obtained from all the methods are not significantly different from each other. V2 does have a slight edge in obtaining lower values on the objective, but the difference is small. However, the lower bounds are significantly different across the methods. For the same problem, V2 and V2-based methods can cut off a large number of sub-optimal solutions with the help of the additional constraints added. For v1, v1m, and v3, the difference between the lower and upper bounds is significant, thus making the gap large. These formulations are unable to find better bounds quickly. Even though v3 has all the constraints from v2, it performs poorly because the number of binary variables in this version of the formulation is very large. This increases the size of the branch and bound tree significantly.

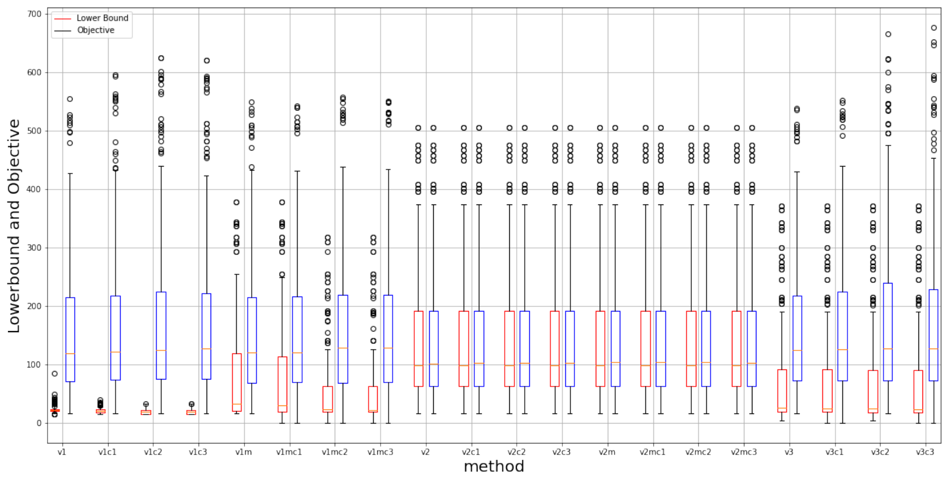

Figure 3.3: Lower bounds and Objective values - makespan

Another important consideration is the extent to which a formulation can solve problems with a small gap. We set this limit as 10% and noted the percentage of problems that were solved to under this limit in the time frame provided. This data is shown in Figure 3.4. In this case also, we see the base formulation v1 performing very poorly. Only 7% of the problems were solved with small gaps. Using callbacks on this version, in fact, made it worse by 2%. With the addition of the valid inequalities in v1m, the performance on this metric was even worse, with only 4% of problems solved with small gaps and 2% when callbacks were used. The numbers are similar, whether it is the common problem or the unique problem. However, with v2, the inequalities we added with the help of the w variables truly helped obtain a tighter formulation, with 99% of problems solved to near-optimality. When the inequalities from v1m were added to v2 to obtain v2m, the ability to solve problems to under 10% decreased slightly, with about 96% instances solved. The addition of callbacks for V2 helps in reducing the gap to an extent, but the improvement is lost after a certain point. This is evident in its improved performance on the optimality index, as seen in Figure 3.4, with nearly all problems solved to under 10% gap, but the average gap for these methods is slightly higher than v2.

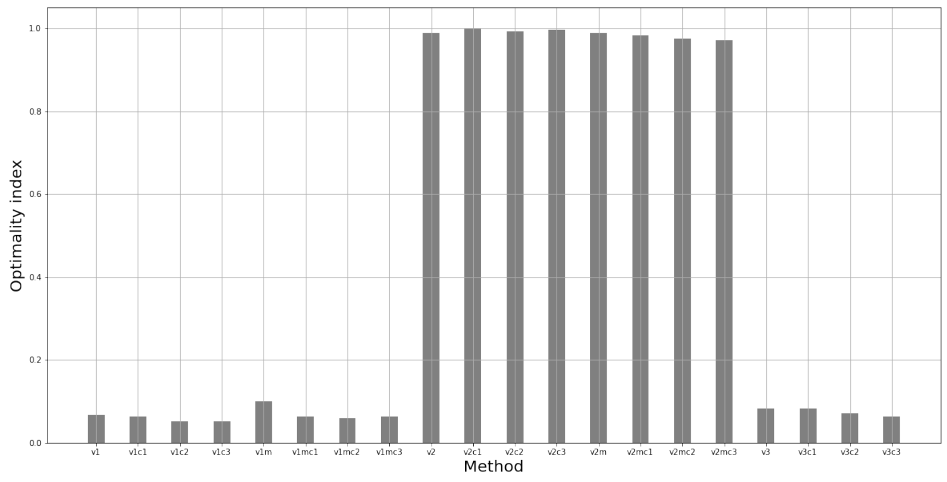

Figure 3.4: Optimality index - makespan

Since the methods v1, v1m, and v3 could not solve most instances to optimality, they also spent the most computation time. This is shown in Figure 3.5, with these methods taking the full 2 hours for almost all problems. Since v2 was able to solve most problems to optimality, its average solution time is small in comparison. However, for large problems, v2 also is not able to solve all the problems to optimality.

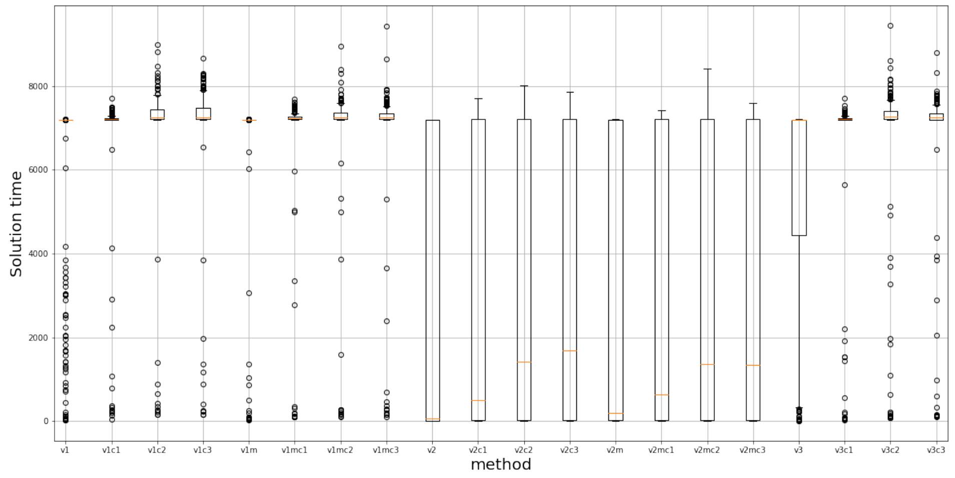

Figure 3.5: Solution time - makespan

Why does the performance of the methods worsen when callbacks are used? This is because there is a significant computational effort to obtain the relaxation values at every node and then solve the separation problems. This is shown in Figure 3.6, where we see that when callbacks are used, a large portion of the computation time is spent on solving the separation problems. When a large portion of the time is spent in the callbacks, there is less time to explore the branch and bound tree, and hence fewer nodes are explored. This leads to slightly worse performance. However, the difference is not very large because the cuts obtained from these callbacks help cut off portions of the solution space.

When the heuristic method is used to solve for the subtour elimination violations, the time spent in callbacks is about 40% for v1. However, when we use the exact method from [80], the time spent on callbacks increases (nearly doubles) because it takes significantly longer to solve the multiple max-flow problems to obtain the subtour elimination violations (or provide a certificate of no violations). The heuristic can obtain the cuts in $O(n^2)$ whereas the exact method takes $O(n^4)$. This result is valid for the other formulations as well, with c1 consuming less time than c2 or c3. However, it is interesting to note that for v3, the percentage of time spent on callbacks is significantly higher than the other methods, with v2c3 spending nearly all the computation time within the callback functions. This is because, in v3, the callbacks are based on the 'w' variables, which are large in number. Obtaining the relaxation values and solving the separation problems hence takes a long time.

Even when we use the heuristic in tandem with the exact method, there is no significant difference in the computational performance because in most nodes, when the heuristic cannot find a cut to add, the exact method is called.

In Figure 3.7, we present the percentage of subtour cuts obtained from each of the two methods in the c3 tests. From this figure, we see that while the heuristic can find many cuts, it misses out on average 25% of the cuts for any of the methods in question. So, while it saves time, it misses some of the violations found by the exact method of separating these constraints.

To Summarize, even though a large portion of time is spent, these methods can find cuts and explore fewer nodes of the tree and still provide solutions that are nearly as good.

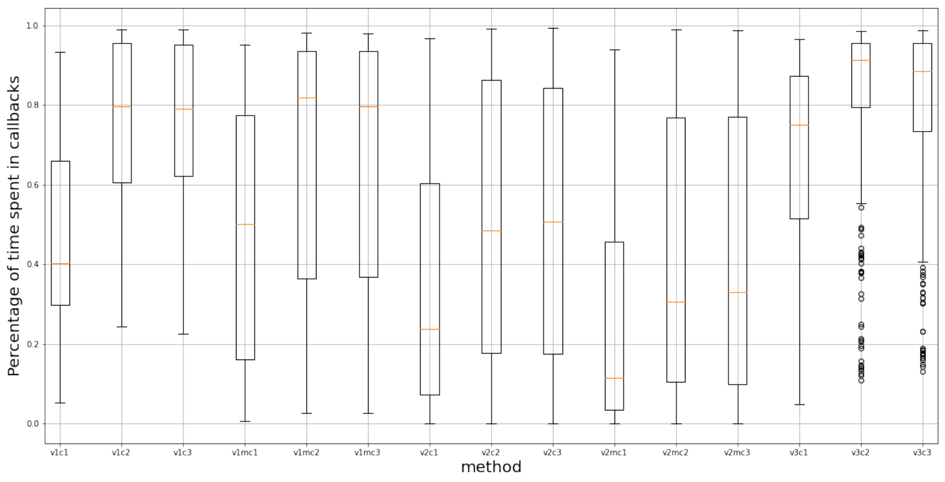

Figure 3.6: Percentage of time spent in callbacks - makespan

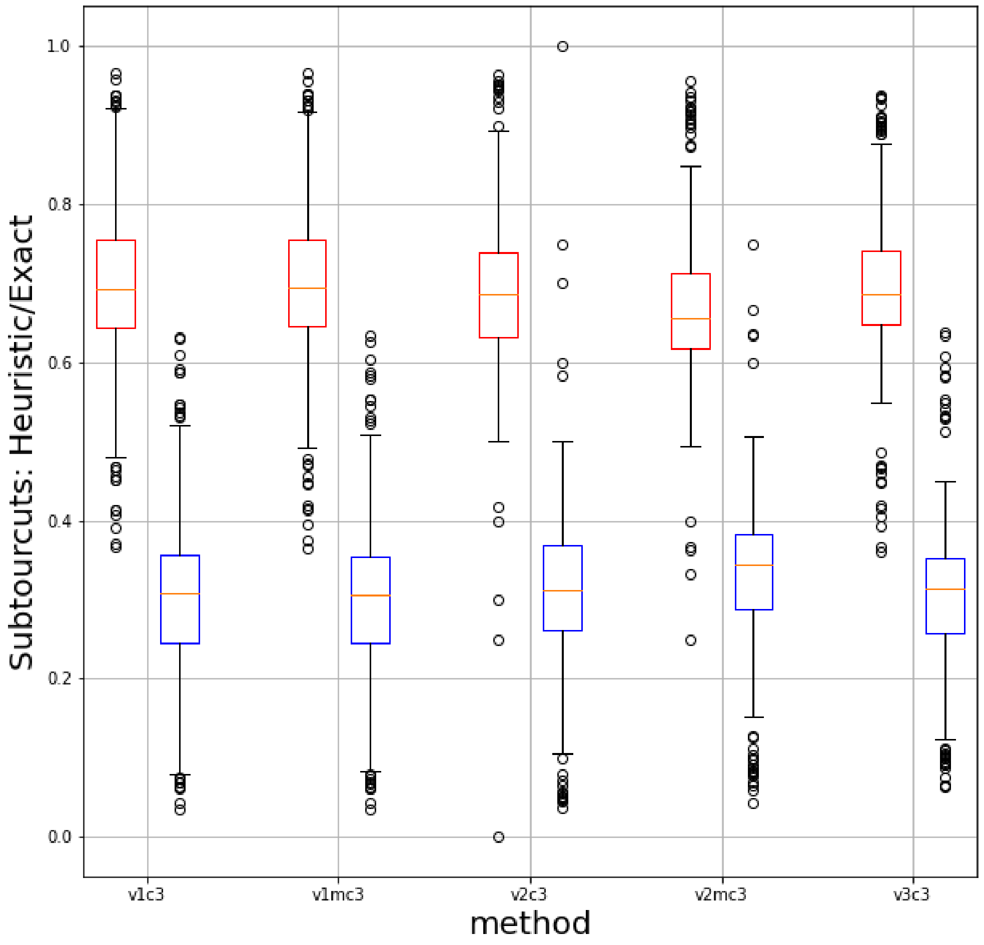

Figure 3.7: Percentage of subtour elimination cuts from heuristic and exact methods - makespan

As a summary on the makespan objective, we can conclude that the addition of callbacks does not significantly affect the performance with respect to the gap. However, it can find solutions by exploring significantly fewer portions of the tree. However, with faster and better-coded callback functions, easier access to the root relaxation solutions from commercially available solvers, and quicker methods to solve the linear relaxations, the use of these callback functions can improve the performance of any method. Finally, the most significant improvement was provided by using the w variables as continuous variables, along with the inequalities added with the help of the w variables. For any problem involving the minimization of makespan, we recommend the use of these variables and constraints because, on any metric, the usage of these additions helps in finding better solutions, verifying that the solutions are good, and finding them faster, both in terms of time and the extent of the tree explored.

3.3.3 Lateness

The next objective in our problem is the minimization of lateness or delays in the processing of jobs. On this objective, the performance of the base formulation v1 is not as bad as its performance on makespan. From figure 3.8, we see that the average gap for v1 is about 19%. The average again increases with the use of callbacks. However, when additional inequalities (3.38) to (3.39) are added, the gap improves slightly to about 14%. This is the best method for this objective. The addition of the 'w' variables in v2 or v2m does not provide improvement in the gap.

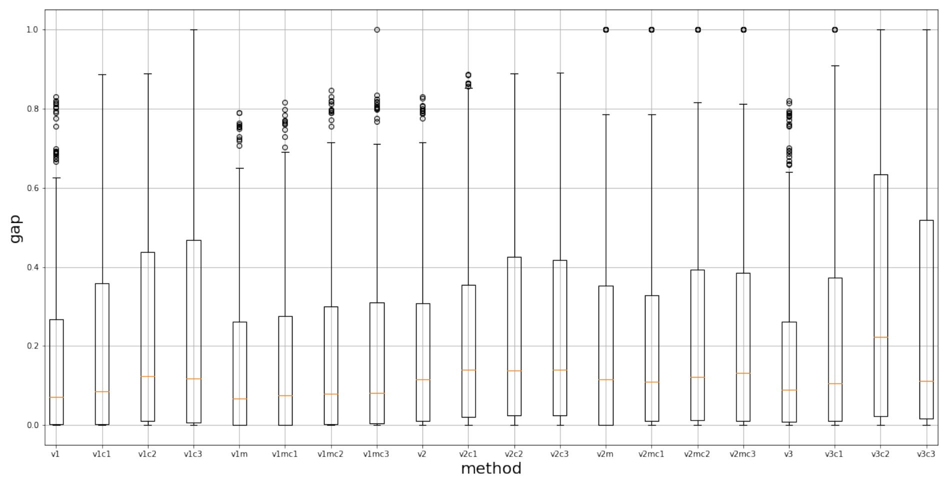

Figure 3.8: Optimality gap - lateness

In Figure 3.9, we provide the values of the objective functions and the lower bounds. This figure shows that the difference between the methods is not significant, either in terms of lower or upper bounds. However, v1m performs slightly better in the lower bounds. V2 methods perform somewhat worse on some problems in obtaining better feasible solutions (upper bounds).

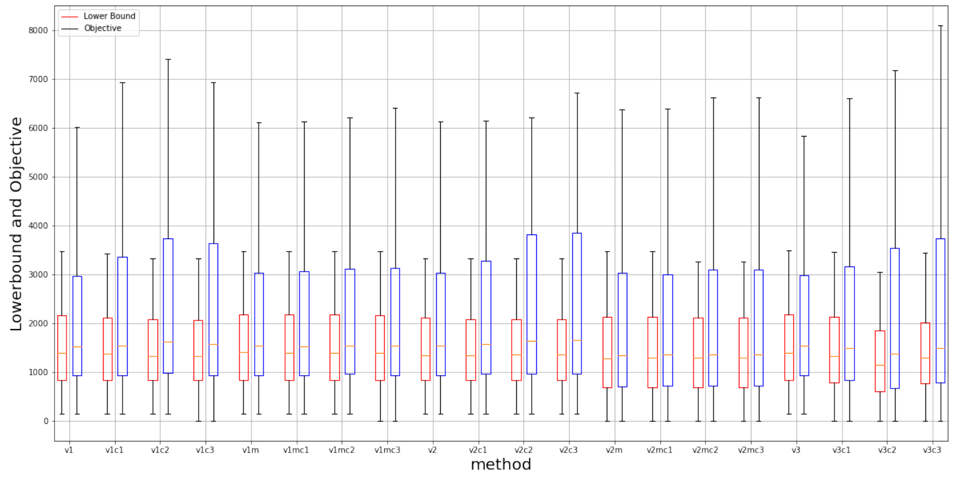

Figure 3.9: lower bounds and Objective values - lateness

When we observe the percentage of problems solved to under 10% optimality (Figure 3.10), we see that v3 is actually the best performer by a small margin, solving about 57% of the problems. V1m is the second-best method here, with about 55% problems solved. V2 methods are the worse performers with only about 40-45% of problems solved to under 10% gap.

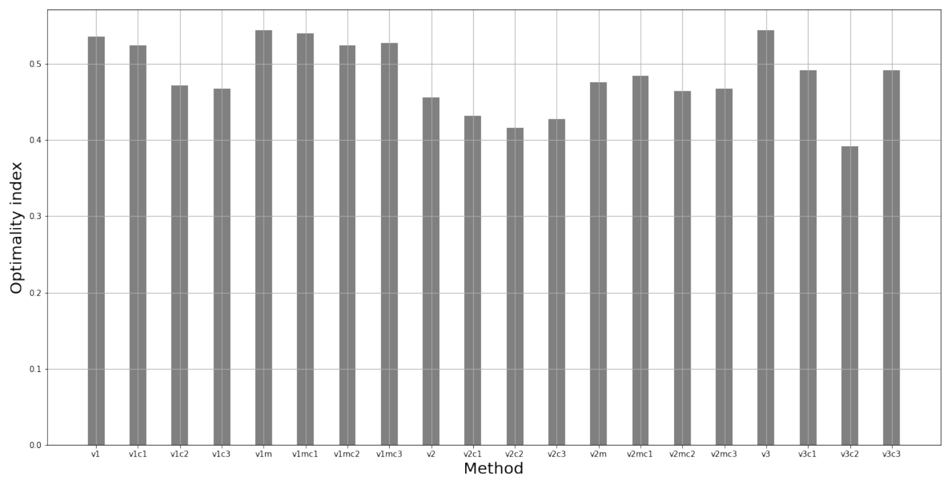

Figure 3.10: Optimality index - lateness

Considering that most problems are not solved to optimality, it is intuitive that the solution times are large and nearing the 2-hour limit. This can be seen in Figure 3.11 where we present solution times. We see that most problems take the full 2 hours to solve for v2, v2m, and other methods. Only v1, v1m, and v3 can solve some problems quickly, but even they struggle with larger problems, especially when the number of jobs is large with respect to the number of machines.

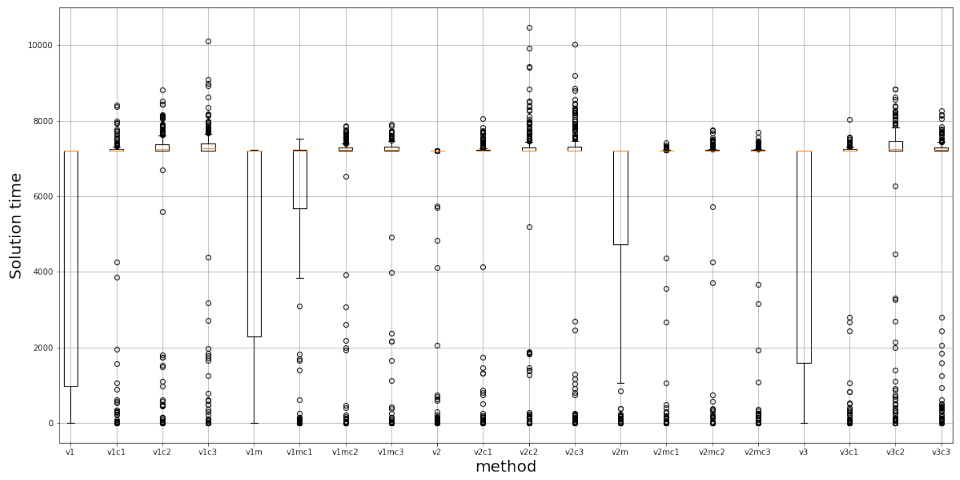

Figure 3.11: Solution time - lateness

The conclusions with respect to the callbacks are similar to the makespan objective. Using the heuristic to obtain the subtour elimination cuts helps in saving time (Figure 3.12), but about 25% of the cuts are missed, as seen in Figure 3.13. Using the exact method to solve the separation means that more time is spent, with some methods spending over 90% of the computation time in callbacks (Figure 3.12). However, as we saw from the gap analysis, these methods provide similar solutions by exploring significantly fewer nodes. For this objective also, in v3, when callbacks are used, especially the exact methods, then most of the computational effort goes towards solving the separation problems (Figure 3.12).

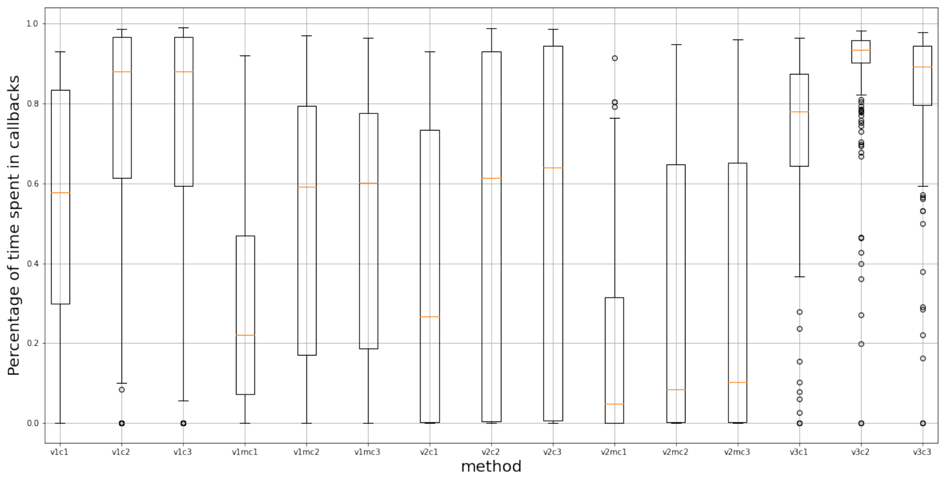

Figure 3.12: Percentage of time spent in callbacks - lateness

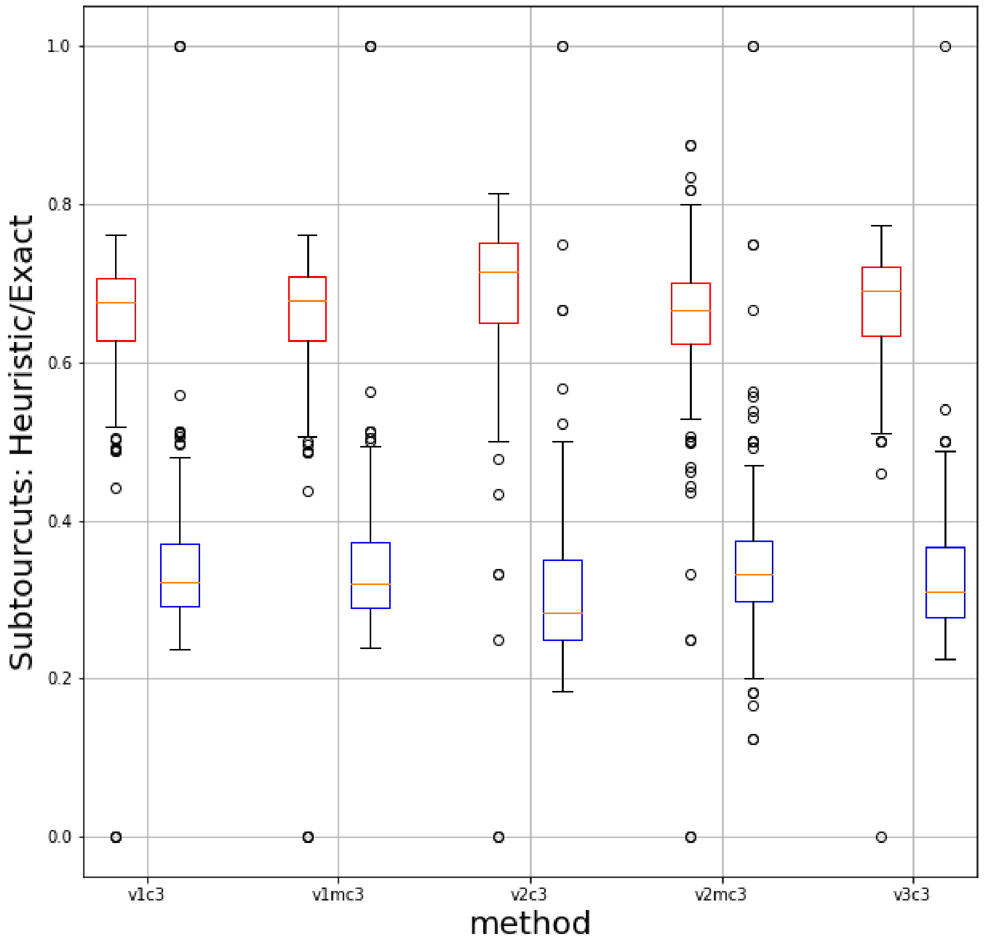

Figure 3.13: Percentage of subtour elimination cuts from heuristic and exact methods - lateness

3.3.4 Setup times

The next objective in our problem is the minimization of setup times or the minimization of total setups (since each setup takes the same amount of time). Figure 3.14

shows the average gap for every method across the instances of the problem tested. Here, we see that the v3 methods solve all problems to optimality. For this objective also, the v2 and v2m methods perform worse than the others. Even the worst method has an average gap of only 2%. This is the first of the easier objectives in our problem. From the tests conducted, all the methods were able to solve most instances to under 10% optimality gap.

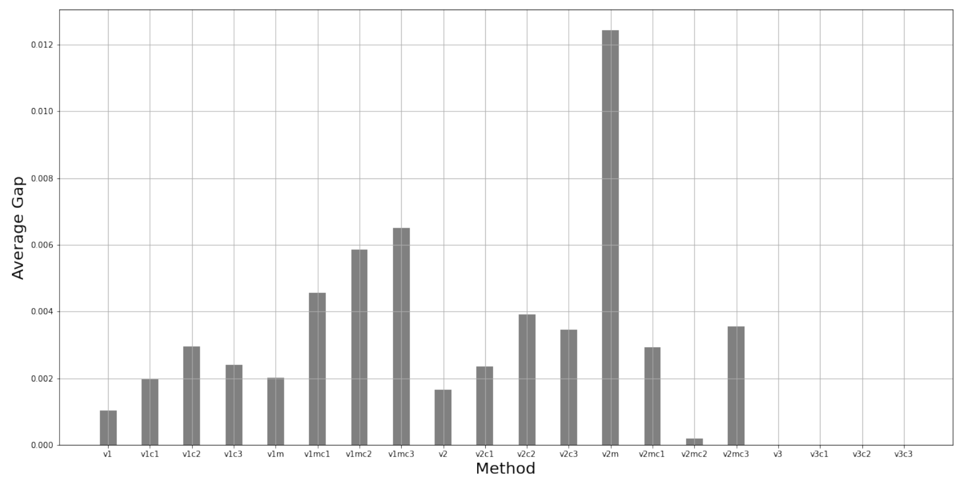

Figure 3.14: Optimality gap - setup times

Since all the methods performed well for this objective, we turn to the amount of time each method took to solve these instances to optimality. In figure 3.15, we see that the v3 methods can solve the instances significantly faster and takes less than 30s on average. This is about five times faster than the base formulation, which averages about 150s to solve. The usage of callbacks does not significantly affect the performance of the methods. However, it is interesting to note that the percentage of time spent on callbacks is lower than for the makespan and lateness objectives, with the c1 methods spending less than 20% of the total time solving the separation problems (Figure 3.16). The c2 and c3 methods take longer, but they also spend less than 30% of the total time in callbacks. This is because the root relaxation takes longer in comparison to the time spent in the branch and bound stage of the solver, i.e., it takes fewer nodes to reach optimality for this objective, and hence less time spent in trying to add more cuts.

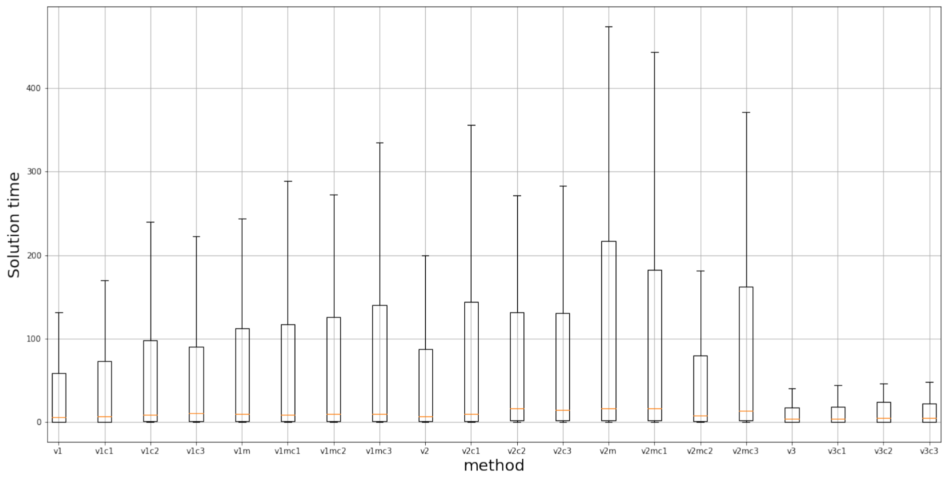

Figure 3.15: Solution time - setup times

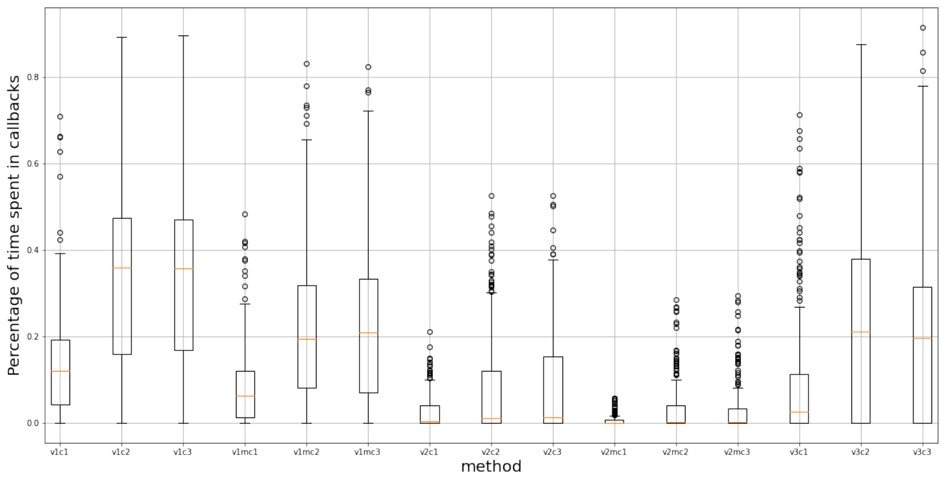

Figure 3.16: Percentage of time spent in callbacks - setup times

3.3.5 Color Preference

The fourth objective in our problem is the maximization of color preferences in the sequences of jobs on each machine. This objective is mathematically similar to the minimization of the setup times because the only differing component is the vector of the objective function coefficients.

Thus, it is expected that the performance of the methods on this objective is also similar to their performance on the setup times' objective. In Figure 3.17, we see that v1, v1m, and v3 methods can solve all instances to optimality. We also see that the worst-performing method is v2m, with an average gap of only 3%. In this case, using callbacks helps in improving the gap. In terms of solution times (Figure 3.18) also, v2m takes longer to solve, while v3 is the best performer, solving all instances in under 100s, with an average of only 20s. The base formulation is not much worse, with an average solution time of about 35s. For this objective also, the methods spend less time on the callbacks because very few nodes are explored before reaching optimality. On average, the c1 methods spend less than 5% of the total time in callbacks, whereas the c2 and c3 methods spend about 10% of the time in callbacks (Figure 3.19). The percentage of cuts added by the heuristic and the exact methods does not vary for this objective in comparison to the makespan or lateness objectives, with the heuristic missing about 25% of the subtour violations.

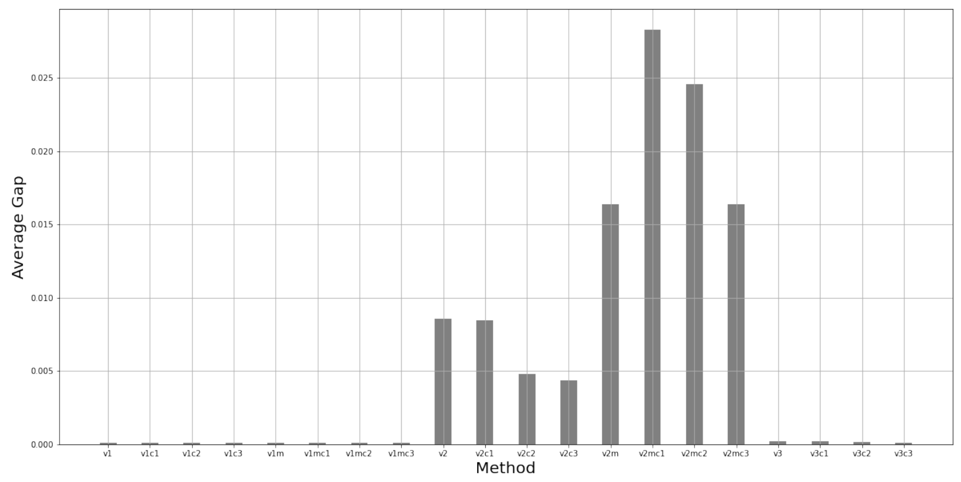

Figure 3.17: Optimality gap - color preference

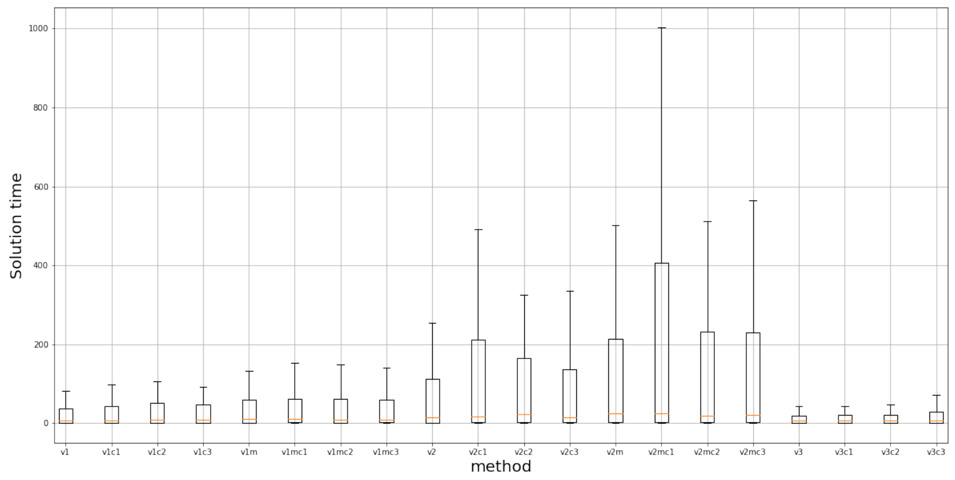

Figure 3.18: Solution time - color preference

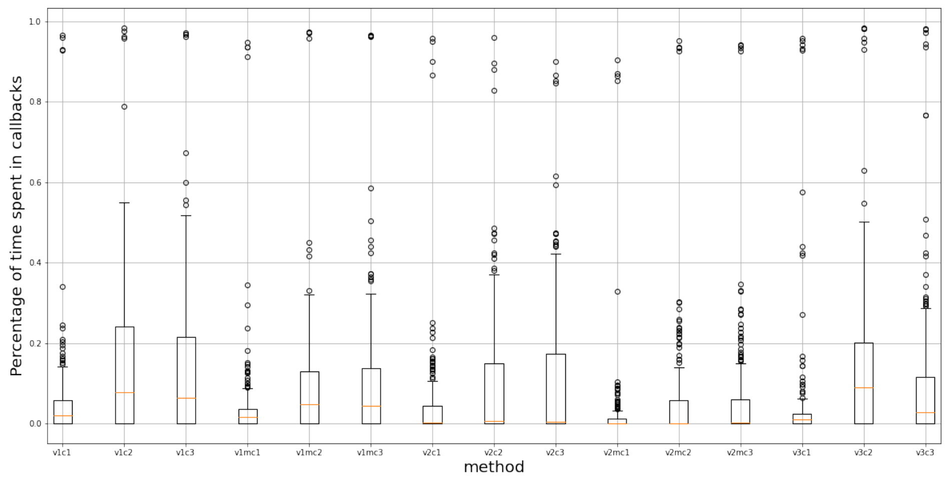

Figure 3.19: Percentage of time spent in callbacks - color preference

3.3.6 Shade Consistency

The last of the five objectives in our problem is the minimization of shade differences and improving the consistency of shades of the jobs on each machine. On this objective, the base formulation performed the best, solving all instances to optimality (Figure 3.20). However, the other methods were not significantly worse. V2 and v2m methods were also worse on this objective, with an average gap of about 10%. V3 had an average gap of about 5%. However, it is very interesting to note that v3 rarely reaches 0% gap. This is evident from Figure 3.21, where v3 takes the full 2 hours for most instances, especially when callbacks are used. This means that v3 can reach small gaps quickly but cannot close the gap to 0. V2 and v2m, however, can close the gap quickly for some problems, but for larger problems, they struggle to reach small gaps, thus increasing the average gap. Figure 3.22 shows that in v1c1, the amount of time spent on callbacks is, on average less than 5%. Again, here in v3, we see that more than 90% of the time is spent in callbacks, exploring many nodes in the process but unable to close the gap.

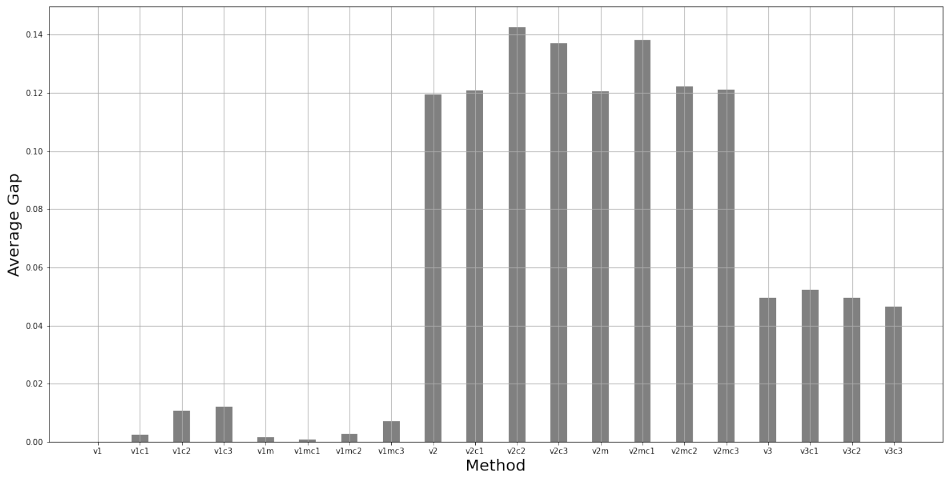

Figure 3.20: Optimality gap - shade consistency

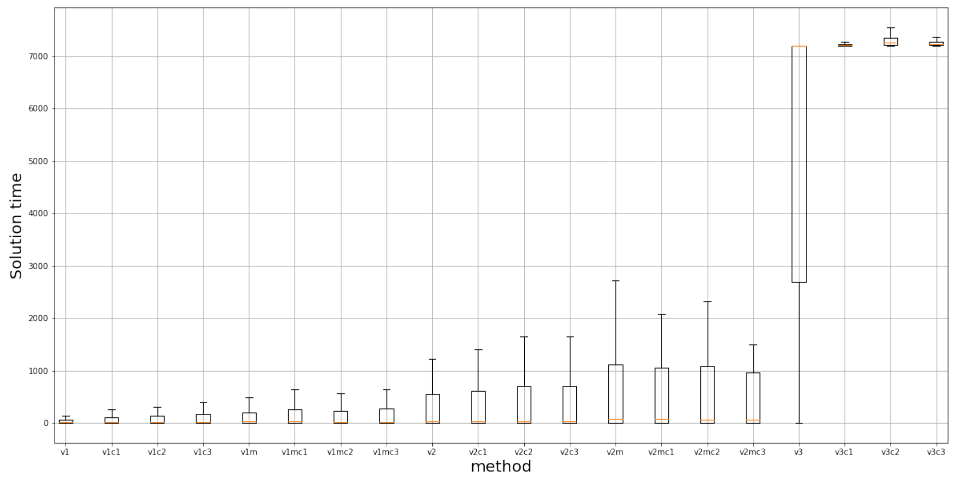

Figure 3.21: Solution time - shade consistency

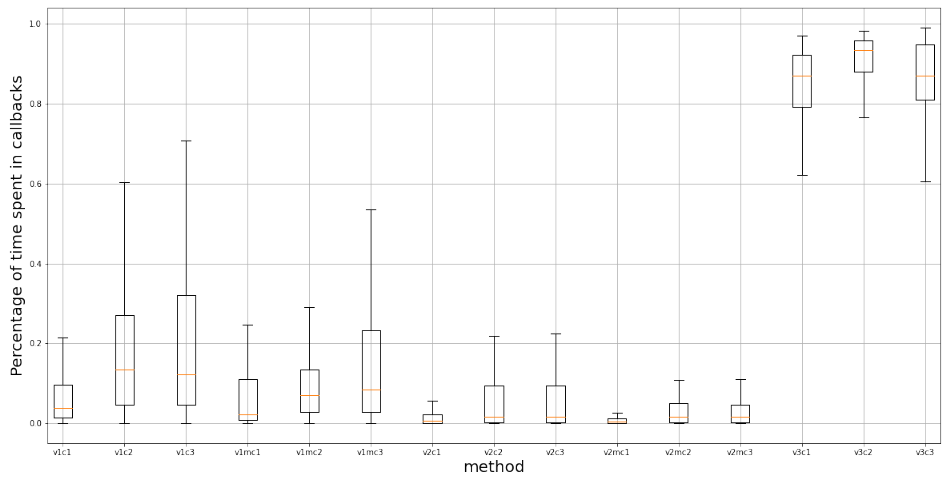

Figure 3.22: Percentage of time spent in callbacks - shade consistency

3.3.7 Summary of results

This section summarizes our recommendations for the best version of the formulation to solve each objective. The recommendations are outlined in table 3.3.7. For the makespan objective, we recommend using v2 without callbacks. This version provided the most significant benefit in terms of all metrics. The average gap is small; most instances were solved with small gaps of less than 10%, and the solution times were low as well. For the lateness objective, we recommend using both v1 and v1m with callbacks to obtain the best objective and verify that the solutions found are of high quality. On average, we found this combination to provide solutions with less than a 10% average gap for the instances we tested. This is a difficult objective to solve, so the solution times will not be low. But usually, high-quality solutions are found in less time, and a significant amount of time is spent making small improvements to both the upper and lower bounds. For the setup times and color preferences objective, we recommend v3, which is significantly better than the rest in being able to solve the problems fast. But, any of the formulations will be able to solve these objectives with small gaps in under 2 hours. For the shade consistency objective, we recommend the base formulation without callbacks. This was able to solve all our instances to optimality in under 2 hours and was able to solve them on average in under 3 minutes.

The usage of callbacks, while not highly useful in reducing gap or improving solution time, prove that there are plenty of violated inequalities to be found, and hence using them provides tighter formulations at every node. Also, if these functions were coded to compute faster, there would be an improvement in solution times, as they provide high-quality solutions by exploring fewer nodes.

In terms of the types of callback functions used, we do not see any improvement in using the heuristic along with the exact method. Using just the heuristic does provide a considerable benefit in computation time, regardless of the objective in question, but also is unable to find all violated inequalities. If the aim is to ensure all violated inequalities are found and added, we recommend using only the exact method to reduce code complexity.

	makespan	lateness	setups	colors	shades
Best method	v2	v1/v1m	v3	v3	v1

Table 3.2: Summary of recommended methods for each objective

Chapter 4

Construction Heuristics

In this chapter, we introduce different heuristic algorithms to provide good quality feasible solutions quickly. We have also developed an MILP-based heuristic method which breaks down large problems into smaller problems by removing some possible solutions from the decision space. Then, we compare these methods among themselves and with respect to optimal results to determine the quality of solutions obtained on each of the five objectives.

4.1 The Story of the Chapter

In the previous chapter, we have developed multiple MILP formulations to solve our problems of different sizes. The best MILP formulation we found also could only solve medium-sized problems (up to 100 jobs and 10 machines). While this is an improvement over existing exact approaches, the MILPs are still not usable to solve large problems. Moreover, our industrial client was skeptical about its ability to maintain a solution based on an optimization approach. So, there was a need to develop methods which were quick and could be altered slightly to return solutions which were reasonably good on all five objectives.

In this chapter, we present several simple construction heuristics to provide feasible solutions quickly, followed by several enhancements, including a heuristic method that involves solving segments of the problem as smaller MILPs.

4.2 Construction Heuristic 1: H1 (Job First Approach)

Without machine eligibility constraints and sequence-dependent setup times, the Shortest Processing Time (SPT) rule is optimal for the objective of reducing total completion time. Similarly, for the makespan objective, Longest Processing Time (LPT) rule is optimal. Similarly, Earliest Due Date (EDD) rule is optimal for the lateness objective. When there are machine eligibility constraints, but all processing times are equal, the Least Flexible Job (LFJ) rule is optimal ([81]). The motivation behind the first set of heuristics is that even with the machine eligibility, sequence-dependent setups and sequence-dependent restrictions among the jobs, these rules might give good-enough solutions. So, in heuristic H1, these rules are used to first sort the jobs based on their processing times, due-dates or flexibility. To obtain balanced solutions across multiple objectives, a combined sort of processing time or flexibility and due dates was also used. Once the list of sorted jobs was ready, the jobs would be placed on the machine which could finish the job first. When the list of machines available for the jobs is obtained, the machine eligibility rules and sequence-dependent restrictions are taken into consideration. For example, if a job is not eligible on the first machine, the first machine will not be listed for that job. Also, if a job is a very light color, then a machine which is currently running very dark colors is not listed because of the large shade difference. Once the machine is decided for the job, the machine ready-time is updated and the next job is selected for placement. Algorithm 2 is detailed below. Given that the method loops through all the jobs, its worst case running time is $O(nm)$.

Algorithm 2 Construction Heuristic H1

```
Obtain list of jobs
for every job $i$ in list of jobs do
    Obtain list of eligible machines
    for every machine $j$ in list of eligible machines do
        Finish time for job $i$ on machine $j$ = $\begin{cases} \text{finish time of current job on machine } j + \\ \text{setup time between current job and job } i + \\ \text{processing time for job } i \end{cases}$
    end for
    Place job $i$ on the machine where it finishes first
    Update current job and finish time on the chosen machine
end for
```

4.3 Construction Heuristic 2: H2 (Machine First Approach)

When developing the first set of heuristics H1, we did not explicitly tackle the setup or the preference objectives. This resulted in solutions that were poor with respect to these objectives. Also, to better tackle the preference objectives, the industrial client suggested a method that looked at machines selecting the next job as opposed to jobs selecting the next machine. With that concept in mind and the need to address the new objectives, heuristic H2 has been developed. In this heuristic, the list of jobs is not sorted. To start the algorithm, the machine which is ready first is selected. Based on the machine eligibility and sequence-dependent restrictions, the list of jobs which can be received by this machine is obtained. For each job on that list, a score is given based on the setup time, preference and shade difference between the current job on the machine and the incoming job.

For example, consider a machine which is currently running a light blue job, and there is an incoming light green job. In this case, there is no need for a setup. Also, the preference is 0.5, considering the change in color and the shade difference is also 0. We calculate a "job score" to represent the unweighted normalized average of setup times, preferences and shade differences. In such a case, the score for the incoming job would be $[(0/10)+(0.5/1)+(0/100)]/3$ if the maximum setup time across all jobs was 10 hours. Once the score for each jobs is obtained, the machine receives the job with the best score and earliest due date, and the ready-time and current job on the machine is updated. Then, the next machine in the list is selected to receive a job.

How does this method tackle all the objectives? The rule for selecting the machine is earliest ready times. This rule intuitively ensures a balance of jobs on the machines. The score function incorporated into the algorithm aims to improve the three objectives of setup times, color preferences and shade consistency. The ties broken using due-dates help improve the lateness objective. Algorithm 3 is presented below. Since this algorithm looks at all jobs in each iteration in the worst case, the running time is $O(n^2)$.

Algorithm 3 Construction Heuristic H2

```
Obtain list of jobs to be placed
Obtain list of machines
while Number of jobs to be placed > 0 or Number of eligible machines > 0 do
  Select the machine with the earliest ready time
  Obtain list of eligible jobs that can be placed on chosen machine
  if Number of eligible jobs=0 for chosen machine then
    Remove chosen machine from list of machines
    Continue
  end if
  for every job $i$ in the list of eligible jobs do
    Record the finish time of job $i$ if it is placed on chosen machine after the last job
  end for
  Select the job which has the least finish time among all the eligible jobs
  Update the ready time for chosen machine
  Remove chosen job from list of jobs to be placed
end while
```

4.4 Construction Heuristic 3: H3 (Machine First Insertion Approach)

Algorithm H2 worked well to improve the quality of schedules over H1. However, when the schedules were being inspected, it was evident that improvements were possible. When a schedule is being created, the machines in the factory would be processing some jobs. These jobs cannot be altered on the schedule since they are already being processed. But, when the schedule is being created for new jobs, we need not place jobs one after another. We can also insert jobs in between other jobs so that the final schedule is even better on the various objectives. This is the reasoning behind the development of heuristic H3. In this heuristic also, the machine first approach is used. However, when a machine is going to receive a job, it does not necessarily receive a job at the end of its schedule. The jobs can be inserted between other jobs at the time of creation of the schedule.

Once a machine is selected, the available slots on the machine is obtained. If no job has been placed on the machine yet (at the beginning of schedule creation), then there is only one slot available. Stated formally, if a machine has x jobs, then there will be $x+1$ available slots on that machine. Then, each job-slot pair gets a job-slot score. This score depends on the setup times, preferences and shade differences between the job in question and the jobs adjacent to the slot on the schedule. The job and slot with the best score is selected and the machine ready time is updated. The schedule is also updated.

Algorithm 4 is detailed below, with a computation time complexity for this algorithm is $O(n^3)$ if $n > m$.

Algorithm 4 Construction Heuristic H3

```
Obtain list of jobs to be placed
Obtain list of machines
while Number of jobs to be placed ¿ 0 or Number of eligible machines ¿ 0 do
    Select the machine with the earliest ready time
    Obtain list of eligible jobs that can be placed on chosen machine
    if Number of eligible jobs=0 for chosen machine then
        Remove chosen machine from list of machines
        Continue
    end if
    Obtain all the eligible slots for chosen machine
    for every job $i$ in the list of eligible jobs do
        for every available slot $k$ in list of eligible slots for chosen machine do
            Obtain the slot-job score of job $i$ if it is placed on slot $k$ chosen machine
        end for
    end for
    Select the job and slot pair which has the best slot-job score
    Update the ready time for chosen machine
    Remove chosen job from list of jobs to be placed
end while
```

4.5 Construction Heuristic 4: H4 (Bin Packing Approach)

The algorithm H3 provided benefit across all the objectives but lateness, and the makespan values and schedule were both acceptable to the industrial client. However, there was still opportunity for improvement in the setup and preference objectives. If the makespan values from H3 were acceptable, we could use those values to continuously place jobs using the same job-slot rules on the same machine until the makespan value for that machine is reached. This helps keep similar jobs together on the same machine and thus improves the setup times and preference objectives. This is the reasoning behind the development of heuristic H4, where a machine is selected and filled with jobs until a predetermined capacity is reached. Then the algorithm moves to the next machine. Algorithm 5 is detailed below, with a computational complexity for this algorithm also is also $O(n^3)$ if $n >> m$.

Algorithm 5 Construction Heuristic H4

```
Obtain list of jobs to be placed
Obtain list of machines
From the schedule created by H3, obtain total running times for all machines
Assign current running time for every machine = 0
while Number of jobs to be placed $> 0$ or Number of eligible machines $> 0$ do
  if All machine current running times $>$ total running times then
    Add one unit of time to total running times for all machines
  end if
  for Every machine $j$ in the list of machines do
    Obtain list of eligible jobs that can be placed on $j$
    if Number of eligible jobs=0 for chosen machine then
      Remove chosen machine from list of machines
      Continue
    end if
    while current machine time for $j <$ total running time for $j$ do
      Obtain list of eligible jobs that can be placed on $j$
      Obtain all the eligible slots for $j$
      for every job $i$ in the list of eligible jobs do
        for every available slot $k$ in list of eligible slots for chosen machine do
          Obtain the slot-job score of job $i$ if it is placed on slot $k$ on $j$
        end for
      end for
      Select the job and slot pair which has the best slot-job score
      Update the current running time for $j$
      Remove chosen job from list of jobs to be placed
    end while
  end for
end while
```

4.6 MILP-Based Heuristic - Problem Size Reduction (PSR)

In this section, we propose an MILP-based construction heuristic called the the Problem Size Reduction (PSR) method. In this method, we break down the original, single connected graph into several smaller unconnected graphs, removing as few edges as possible in the process. Each of these smaller problems can be solved to optimality, the results of which can be aggregated to form a feasible but possibly sub-optimal solution. The computation time of this PSR heuristic method will be longer than any of the prior construction heuristics, since it will involve solving several smaller MILPs. However, its solution quality should be competitive given that an optimization problem is still being incorporated into the solution.

Consider the example shown in Figure 4.1. In this example, there are four jobs {J1, J2, J3, J4} and two machines {M1, M2}. Job J1 is eligible to be processed on M1. Job J2 can be processed on M1 and M2. Jobs J3 and J4 can be processed only on M2. Now, if we remove only one edge in this graph, i.e., J2-M2, we can break down the larger problem into two small problems as shown. Now, those two small problems can be solved to optimality and the individual schedules obtained from those small problems can be combined together to obtain a feasible (not necessarily optimal) solution to the larger problem.

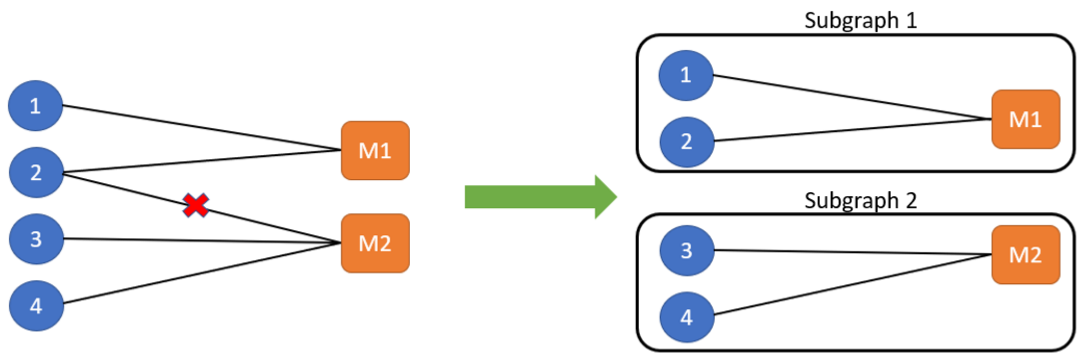

Figure 4.1: Illustration of the problem size reduction method

Now let's consider a set of jobs N to be scheduled on a set of machines M. Let the bipartite graph representing the machine eligibility for the jobs be $G(V,E)$, where V represents the set of nodes corresponding to the jobs and the machines, and E represents the set of edges, with an edge being present between a job node and a machine node if the job is eligible to be processed on that machine. Given the difficulty in solving the formulation to optimality for larger graphs, we propose removing selected edges in this graph and solving the disconnected subgraphs as separate problems. We can create optimal schedules for each of the subgraphs and combine them to get a complete schedule which is feasible as well. This is an instance of a graph partitioning problem with constraints. This is proved to be NP-Hard in [52].

4.6.1 Variables

The variables include the v variables that determine the side of the partition on which a node is present, and the w variables that determine which edges are removed.

$$v_i = \begin{cases} 1, & \text{if node i is in subgraph 1} \\ 0, & \text{if node i is in subgraph 2} \end{cases} \qquad \forall i \in V$$

$$w_{i,j} = \begin{cases} 1, & \text{if edge (i,j) is removed} \\ 0, & \text{otherwise} \end{cases} \qquad \forall (i,j) \in E$$

4.6.2 Objective

The PSR method partitions the given graph into two disconnected subgraphs by removing some edges from the given graph and may be iteratively applied to the larger remaining subgraph. The objective of this MILP (4.1) is to reduce the number of edges removed from the given graph. The fewer the edges removed, the less the loss of information that might affect solution quality.

$$\min \sum_{(i,j)\in E} w_{i,j} \tag{4.1}$$

4.6.3 Constraints

Let Q represent the maximum allowable size of the first subgraph. Then, the size of the second subgraph is automatically $|V| - Q$, since they are disjoint. For this dissertation, the maximum value of Q is set to 60, which is the maximum size of problems the MILP has been able to solve. If the MILPs in chapter 3 can be improved to handle larger sized problems, then Q can be larger. It then follows that constraint set (4.2) ensures that there are exactly Q nodes in the first subgraph and the rest of the nodes are in the second subgraph. Constraint sets (4.3) and (4.4) ensure that if two nodes with an edge between them are in different subgraphs, then that edge is removed from the given graph. The w variables act as a count for the number of edges removed from the original graph. Consider a pair of nodes i and j. If they are connected in the original graph, and v_i and v_j take the values 1 and 0 respectively, then it means that in the broken down graph, the node i lies in subgraph 1 and node j lies in subgraph 2. Considering that these two subgraphs are disjoint, there is no edge between i and j. This means that the edge ij has been removed from the original graph. Constraint set (4.5) ensures that each node has at least one other node connected to it. This ensures that every job has at least one machine on which it can be processed. Constraint sets (4.6) and (4.7) ensure binary restrictions are satisfied for the variables. Note that the w variables need not be constrained to be binary due to the enforcement of constraint sets (4.3) and (4.4).

$$\sum_{i \in V} v_i = Q \tag{4.2}$$

$$w_{i,j} \geq v_i - v_j \quad \forall (i,j) \in E \tag{4.3}$$

$$w_{i,j} \geq v_j - v_i \quad \forall (i,j) \in E \tag{4.4}$$

$$\sum_{j \in V-i} w_{i,j} \leq |G(i) - 1| \quad \forall i \tag{4.5}$$

$$0 \leq w_{i,j} \leq 1 \quad \forall (i,j) \in E \tag{4.6}$$

$$v_i \in \{\ 0, 1\} \quad \forall i \in V \tag{4.7}$$

4.6.4 Implementation

To implement the PSR method for our problem, we would do the following. Consider a setting with 500 jobs and 50 machines. Note that this problem when translated to a graph as described above has 550 nodes. Let Q be 50 for this case. This graph is first broken down into two subgraphs containing 50 and 500 nodes each. The 50 node subgraph can now be independently solved to optimality for any of the objectives. The 500 node subgraph is again divided into 50 and 450 node subgraphs. Similarly, the entire 550 node original graph is broken down in iterations into 11 subgraphs of 50 nodes each of which can be solved independently using the formulations proposed in chapter 3. Once the problems are all solved, the solutions are combined to obtain a feasible solution (sub-optimal) to the original problem.

4.7 Results

In this section, we first discuss the computational performance of the PSR heuristic method. It is important to confirm that this method, still dependent on an MILP solver, can find solutions in a reasonable amount of time for large problems. Otherwise, it would also need to be removed from consideration for those problem sizes. Once confirmed, we then explain how the experimental design was implemented for both small and large datasets.

We then compare solution quality of each heuristic against the optimal solutions for small problems. Finally, we compare solution quality among the heuristic methods only for larger problems.

4.7.1 Computation results for PSR

For the problem sizes of interest (≤ 50 machines, ≤ 500 jobs), the non-MILP-based construction heuristics could all solve within 30 seconds. So, they were not subjected to a thorough computational analysis. However, the PSR heuristic method involves the process of solving two MILP subproblems multiple times, as explained in Section 4.6. For larger problems, this method requires a significant computational effort. For this reason, the PSR method was tested using several combinations of machines and jobs. For problems with fewer than 20 machines, the PSR method could obtain solutions quickly as seen in Figure 4.2. For the larger cases, each subproblem implementation was limited to 10 minutes. From Figure 4.2, it can be seen that on average, when the number of machines or jobs increases, the overall PSR method takes longer to solve. This is due to increased complexity with more constraints and variables. The solution time is also dependent on the number of times the subproblem is required to be solved. For example, when there are 550 nodes, there is a need for 11 splits, which means the splitting algorithm (subproblem) is implemented 10 times. Given that the solution time is the total time needed to run the algorithm for all of the splits, the solution time typically increases with increasing number of nodes.

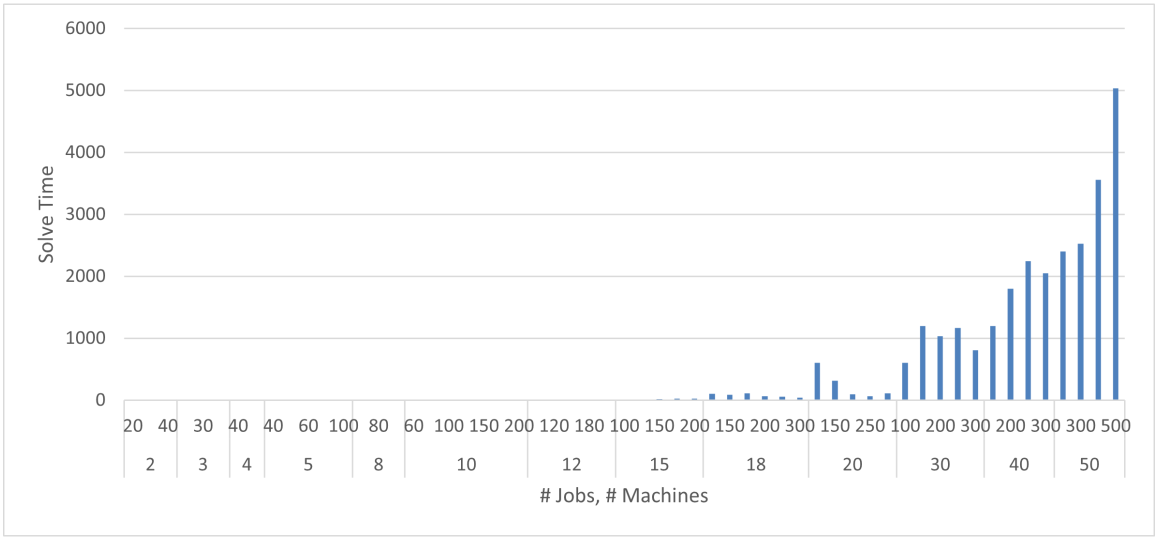

Figure 4.2: PSR Solution time vs number of machines, number of jobs

4.7.2 Design of Experiments

In order to understand how well a particular heuristic performs with respect to an objective, it is important to obtain optimal solutions (if possible) and compare them against the solutions obtained by the heuristic. Our MILP is able to handle problems with up to 50 jobs and 5 machines. In the first set of experiments, we use small problems and solve them to optimality. Then the same problems are also solved by the heuristics and the solutions are compared for optimality gap.

To test the PSR heuristic method and its performance with respect to optimal values on all the objectives, the smaller sized problems were broken up into 2, 3, and 4 groups of equal size, each group solved to optimality and combined to form a single schedule. For example, if there are 3 machines and 20 jobs, then there are 23 nodes. If dividing this problem into two groups, then there would be 12 nodes in one group and 11 in the other group. These two groups would then be solved to optimality individually. Thus, for these smaller problems, the results will include three PSR heuristics: PSR 2-split, PSR 3-split, and PSR 4-split.

For this set of experiments, we test configurations with 2, 3, 4, 5 machines and 10, 20, 30, 40, 50 jobs. The colors, shades, due dates, processing times, machine eligibility and setup times between the jobs are randomly assigned. The color is randomly selected among 10 colors. Each color has a range of possible shades, and the shade is selected from that range randomly with same probability. The due dates are randomly selected using a uniform distribution with a range of [-1 week, +3 weeks]. The processing times are randomly selected using a uniform distribution with a range of [3 hours, 15 hours]. Machine eligibility is selected based on each machine having equal probability of being able to or not being able to process a job. Setup times between jobs are assigned based on their shades and colors and can take either 0, 5 or 10 hours. Each setup is considered as 5 hours. If there is a 10-hour requirement, it is considered as 2 setups. Each configuration (ex: 2 machines, 10 jobs) is tested for 10 different datasets to ensure consistency of the results. For each configuration, the average objective function value and average gap (percent above optimal) across the 10 datasets are reported.

In the next set of experiments, containing problems too large to be solved using the MILP in a reasonable amount of time, the heuristics are compared among themselves on all five objectives for larger datasets. These large datasets contain 10, 20, 30, 40, 50 machines and 100, 200, 300, 400, 500 jobs. For the PSR heuristic method, the large dataset is divided into groups with each group containing a maximum of 60 nodes. For example, if the dataset has 400 jobs and 20 machines, it would be divided into 7 groups of 60 nodes (jobs/ machines) each, then each group would be solved to optimality, then combined to form a single schedule. Each configuration of the dataset has random due dates, processing times, machine eligibility, and setup matrices. Each configuration in this set also has 10 different datasets. The gap is calculated with respect to the best performing heuristic (percent above best solution) and the gaps are averaged across the 10 datasets.

Both sets of experiments are conducted on the all of the heuristics detailed in the previous section. However, for construction heuristic H1, the list of jobs are sorted in 7 different ways: Shortest processing time (SPT), Longest Processing time (LPT), Least flexible job (LFJ), earliest due date (EDD), SPT+EDD, LPT+EDD, and LFJ+EDD. When a combination of sorting methods are used, the jobs are first grouped based on the processing times and within the groups, they are sorted by earliest due date. Since the jobs range between 1 hour and 15 hours of processing times, they are grouped into 1-5 hour jobs, 6-10 hour jobs and 10+ hour jobs. For SPT+EDD and LPT+EDD methods, this grouping is used. For LFJ+EDD, the ties for the flexibility are broken using due dates. For the rest of the heuristics, the sorting of the jobs has little effect. For the smaller sized problems, the MILP is solved for each objective separately to obtain the optimal solution for that dataset and objective.

4.7.3 Comparison with respect to the optimal solutions

In this subsection, we consider the results of the first set of experiments with respect to each objective. The results are compared for optimality gaps with respect to the optimal solutions obtained from the MILP.

4.7.3.1 Makespan objective

For the makespan objective, the objective grid (Figure 4.3) indicates that the PSR 2-split method seems to be the better performing method as its values appear closer to the optimal value. However, within the non-MILP-based heuristics, it is not clear as to which method is better, as the values seem close to each other. When a Kruskal Wallis test was conducted on the gaps, the p-value was less than 0.05, which means that at least one of the methods is different than the others. When we look at the box plots (Figure 4.4), it is evident that the PSR 2-split has the best average gap, followed by LFJ methods, H2, H3 and LPT methods. The range of values for PSR 2-split is slightly large though. This is due to some cases when the splitting of the datasets resulted in skewed sub-dataset formation. For example, if there were 5 machines and 50 jobs, there was a chance of one group having 1 machine and 25 jobs and another group having 25 jobs and 4 machines.

4.7.3.2 Lateness objective

For the lateness objective, objective grid (Figure 4.5) indicates that EDD and PSR are better than the rest of the heuristics. This is also true on the box plots (Figure 4.6). When EDD was incorporated as a secondary sort on the list of jobs, it improved the gap slightly on the lateness objective. These conclusions are expected due to EDD's strength related to the lateness objective. With the flavor of optimality in the PSR heuristic method, we expect good results as well on all objectives. However, it is interesting to note that even EDD was 67% worse on average when compared to the optimal results. Also, the difference between EDD and other methods was at most 9%.

4.7.3.3 Setup ratio objective

For the setup objective, the objective grid (Figure 4.7) show that the PSR heuristic methods perform significantly better than the rest. This is also true on the box plots (Figure 4.8) where the gaps on these methods is significantly lower. For the PSR 2-split method, the average gap with respect to the optimal results is only 6%.

Among the non-MILP-based heuristics, the H2, H3 and H4 methods perform better. Considering that these methods were developed to improve this objective along with the preference and shade consistency, this result is expected. Also, within the algorithm, the scoring function has a setup component which tries to minimize setups.

4.7.3.4 Color preference objective

For the color preference objective too, the PSR heuristic is the best followed by the methods H2, H3 and H4. This is evidenced by the objective grid and the box plots (Figures 4.9 and 4.10). PSR methods benefit heavily by the use of optimal methods within them. The only loss of optimality is due to the removal of some edges from the machine eligibility graph.

H4 is the best performing non-MILP-based heuristic for this objective and the gap with respect to the optimal is on average only 13%. This is due to the binning approach followed by H4, which allows for similar jobs to stay together on the same machine.

4.7.3.5 Shade consistency objective

For this objective, the PSR heuristic is worse when compared to the rest of the objectives as seen in Figures 4.11 and 4.12. The rest of the heuristics have a gap of about 16% with respect to the MILP. There is almost no difference among the H1 heuristics for this objective. This objective comes into greater light for larger datasets with more jobs on each machine.

4.7.4 Comparison between the heuristics on large datasets

In this subsection, we consider the results of the second set of experiments with respect to each objective. The results are compared for gaps with respect to the best solutions found from among the different heuristic methods.

4.7.4.1 Makespan objective

On the makespan objective, it is hard to differentiate between the methods from the objective grid 4.13). However, it is interesting to note the few cases where PSR has performed terribly. This is again attributed to the few cases where the grouping was imbalanced. Such imbalance is more likely when there are fewer machines and more jobs. To tackle this imbalance in the future, the PSR method can be made more robust to include a better balance of machines and jobs within each group.

LPT performed the best among the non-MILP-based heuristic methods as seen in the box plots (Figure 4.14). The difference between the methods is small however, with the worst performer (PSR) only about 19% worse than the best. Other than PSR, the worst performing is H4 with about 12% gap. This may happen because each method has some mechanism to ensure that the machines are balanced. In H1, the machine which can finish the job first is picked whereas in H2 and H3, the machine with the least completion time is picked for receiving a job. On H4, the machine utilization values from H3 is used as guidance. This makes the methods perform similarly on this objective.

4.7.4.2 Lateness objective

On the lateness objective, once again, EDD performs the best as expected (Figures 4.15 and 4.16). PSR is a close second with an average gap of 11%. Combining EDD with other methods once again helped the objective, for example, the third best was SPT+EDD. Among the other methods, there was insignificant difference. Except for the PSR heuristic, the solutions from the non-EDD heuristics were about 35% worse when compared to the EDD solutions.

4.7.4.3 Setup ratio objective

On the setup objective, we see PSR heuristic dominate the other methods. This is evident in the objective grid (Figure 4.17) as well as the box plot (Figure 4.18).

Among the non-MILP-based heuristics, H4 is by far the best. In most cases, H2 and H3 performed better than H1, but even they were outperformed by H4. Having a large number of jobs helps H4 decide better to reduce the setups on each machine. When there are fewer jobs, the number of possible options for H4 is smaller so the effect of the algorithm design is not seen. In the large datasets, however, with plenty of options to select from, H4 performs significantly better than the rest. For this objective, H4 performs about 50% better than the worst performing method (EDD). H2 and H3 perform about 20% better.

4.7.4.4 Color preference objective

On this objective also, PSR performs the best in all cases, closely followed by H4, with an average gap of 1.4%. H2 and H3 follow closely behind as can be seen in the objective grid and the box plots (Figures 4.19 and 4.20).

H1 heuristics are much worse on this objective. This is because of the placement rules embedded in H2, H3 and H4 which consider the preferences, setups and shade difference values when placing the jobs. These rules are not present in H1 heuristics. On this objective, H4 performs about 31% better than H1 and H2 and H3 perform about 26% better.

4.7.4.5 Shade consistency objective

On this objective also, PSR performs the best by a large margin (Figures 4.21 and 4.22). The closest heuristic is again H4 which is also 78% worse. The rest of the methods perform very poorly in comparison to PSR.

4.7.5 Summary of Heuristic Comparison and Managerial Insights

Overall, PSR performs similar to other heuristics with respect to the makespan objective. In some cases, due to imbalance in splitting, the resulting schedules are skewed. This can be improved with a few constraints ensuring a better mix in each group. PSR is also only very slightly worse than EDD on the lateness objective. But, with respect to the other three objectives, it is significantly better than the rest. When compared to the optimal values also, it is only performing poorly on the lateness objective. An average gap of about 17% is acceptable on the makespan objective and shade consistency objectives. On the preference and setup objectives, the gap is less than 6%. So, overall, PSR is the best method among the tested methods. However, it does not provide quick solutions. For very large problems with more than 400 jobs, it takes a long time to create the optimal split. This gives the non-MILP-based heuristic methods an advantage over any optimal or semi-optimal methods.

Among the non-MILP-based heuristics, H4 is only about 11% worse on the makespan, and definitely bad on the lateness objective, But the gains on the other objectives is significant for H4. Also, on the lateness objective, it only performs very badly when compared to EDD but when compared to the rest of the methods, it is similar. So, overall, H4 seems like the best method. But if makespan is slightly more important, H3 is also a good method, considering that it performs well on the setups and color preference objectives also. However, if lateness is the most important objective, no method beats EDD. But EDD performs worse on all other objectives. Also, it performs badly when compared to the optimal values.

So, there is work that has to be done to tackle this objective.

Table 4.7.5 shows a summary of the methods and on which objectives, they perform well. An 'x' on the table suggests that the method performs well and an 'o' suggests that it does not. From the table it is clear that on the objectives alone, PSR is the best method overall. But it suffers from poorer computation times. Also, when industry implementation is considered, heuristics are easy to build and maintain. Among the heuristics, H4 is a fast heuristic which performs well on all objectives and only slightly worse than the rest on lateness. For lateness, EDD is the best objective. LPT and H3 are the best for makespan.

Method	Makespan	Lateness	Setups	Color	Shade	Computation	Implementation
H1-EDD	x	x	o	o	o	x	x
H1-LFJ	x	o	o	o	o	x	x
H1-LPT	x	o	o	o	o	x	x
H1-SPT	x	o	o	o	o	x	x
H1-LFJ+EDD	x	o	o	o	o	x	x
H1-LPT+EDD	x	o	o	o	o	x	x
H1-SPT+EDD	x	x	o	o	o	x	x
H2	x	o	x	x	o	x	x
H3	x	o	x	x	o	x	x
H4	x	o	x	x	x	x	x
PSR	x	x	x	x	x	o	o

Table 4.1: Summary of competitiveness of the methods with respect to the five objectives, computation time and ease of implementation for a client

4.8 Objective function graphs and box plots for gap

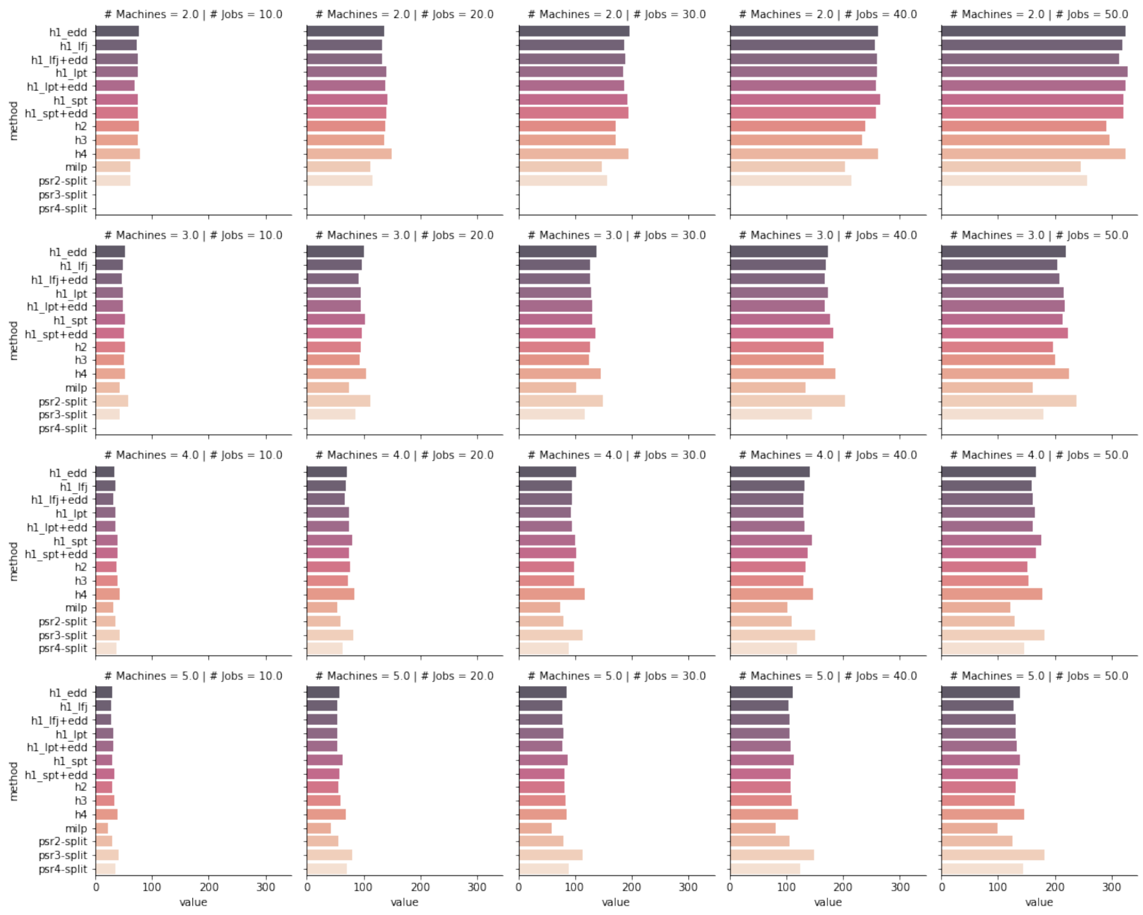

Figure 4.3: Makespan small datasets values

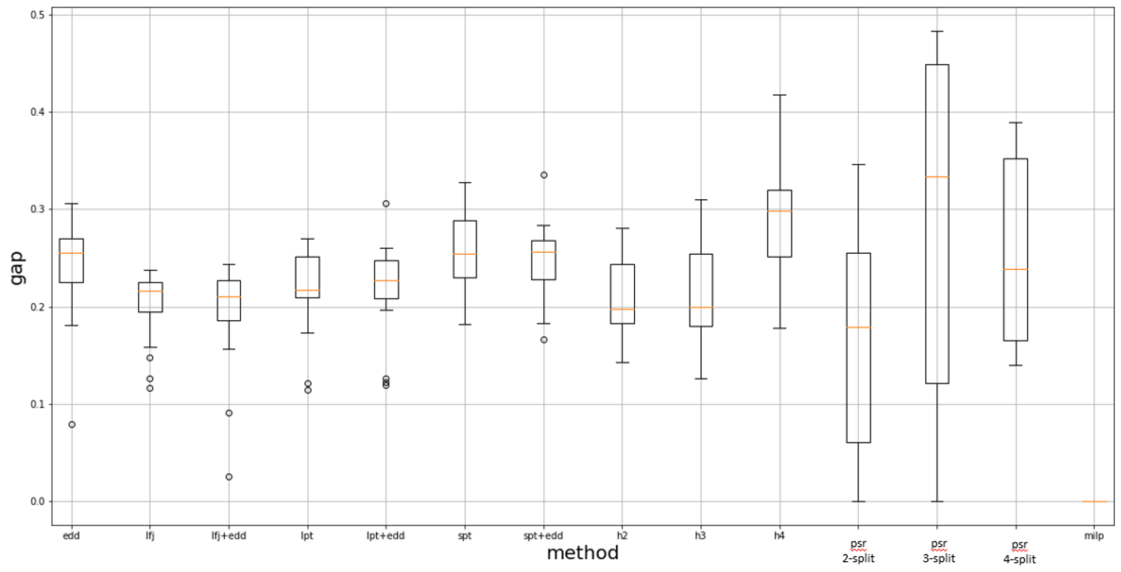

Figure 4.4: Makespan small datasets box-plots for gap

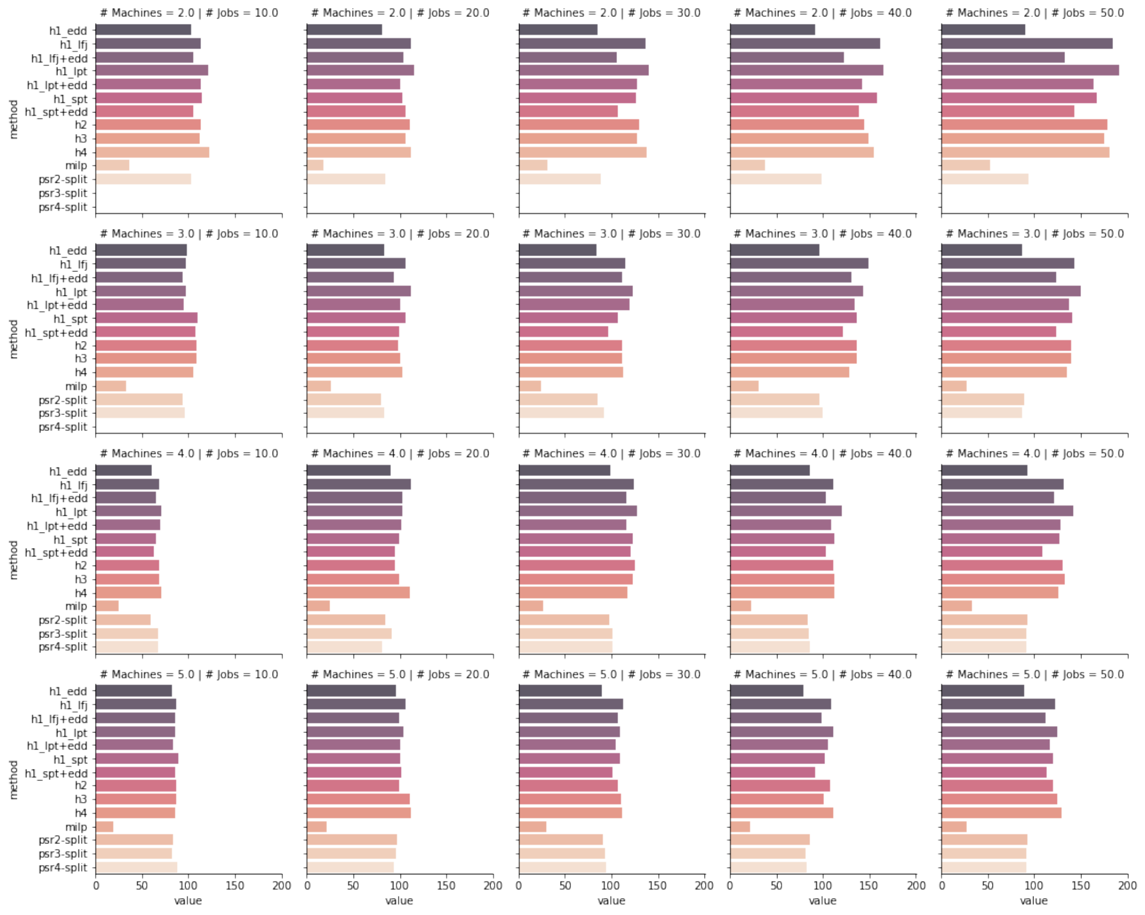

Figure 4.5: Lateness small datasets values

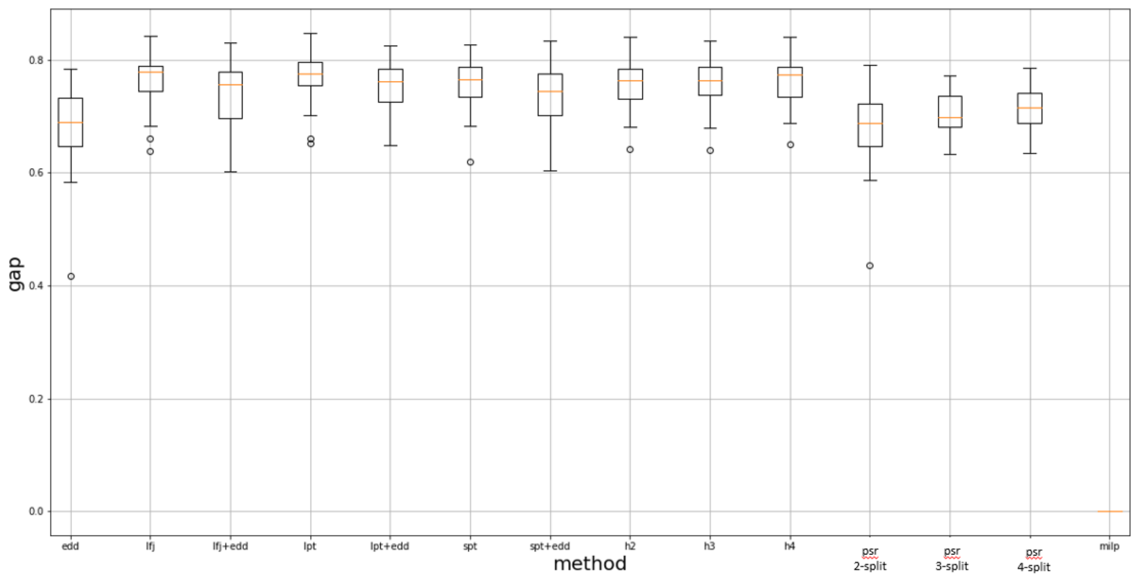

Figure 4.6: Lateness small datasets box-plots for gap

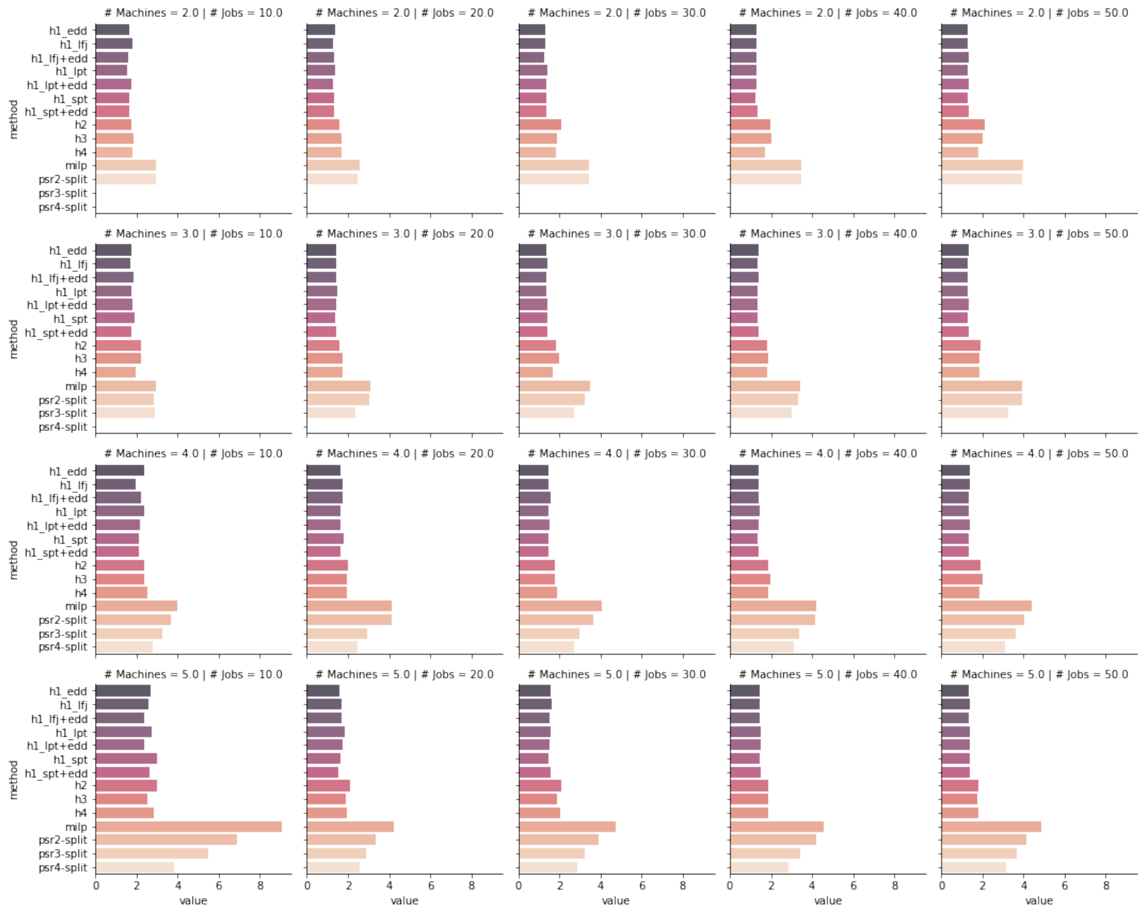

Figure 4.7: Setup ratio small datasets values

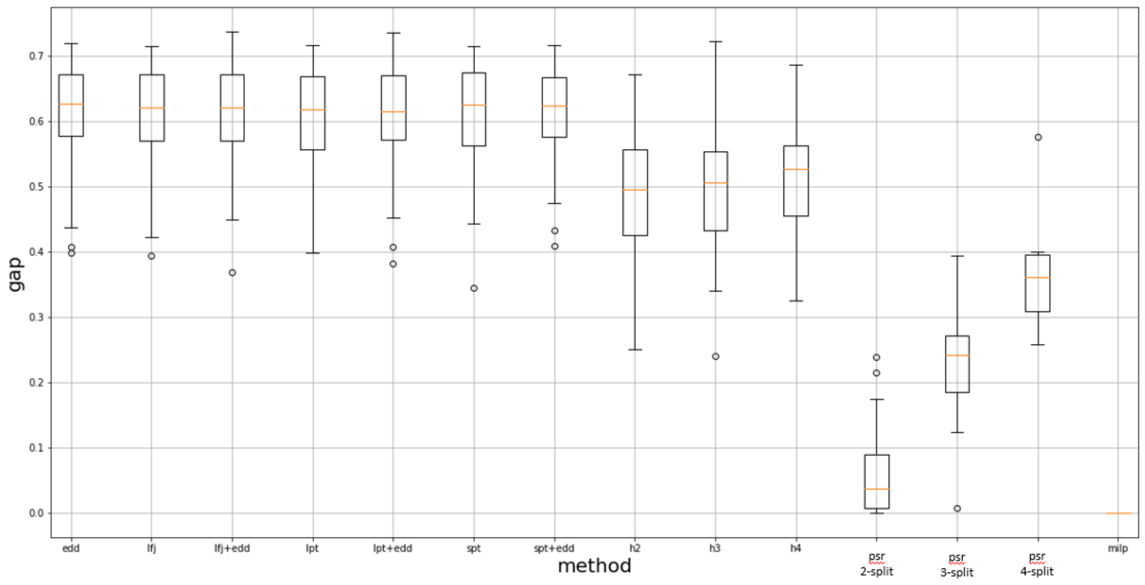

Figure 4.8: Setup ratio small datasets box-plots for gap

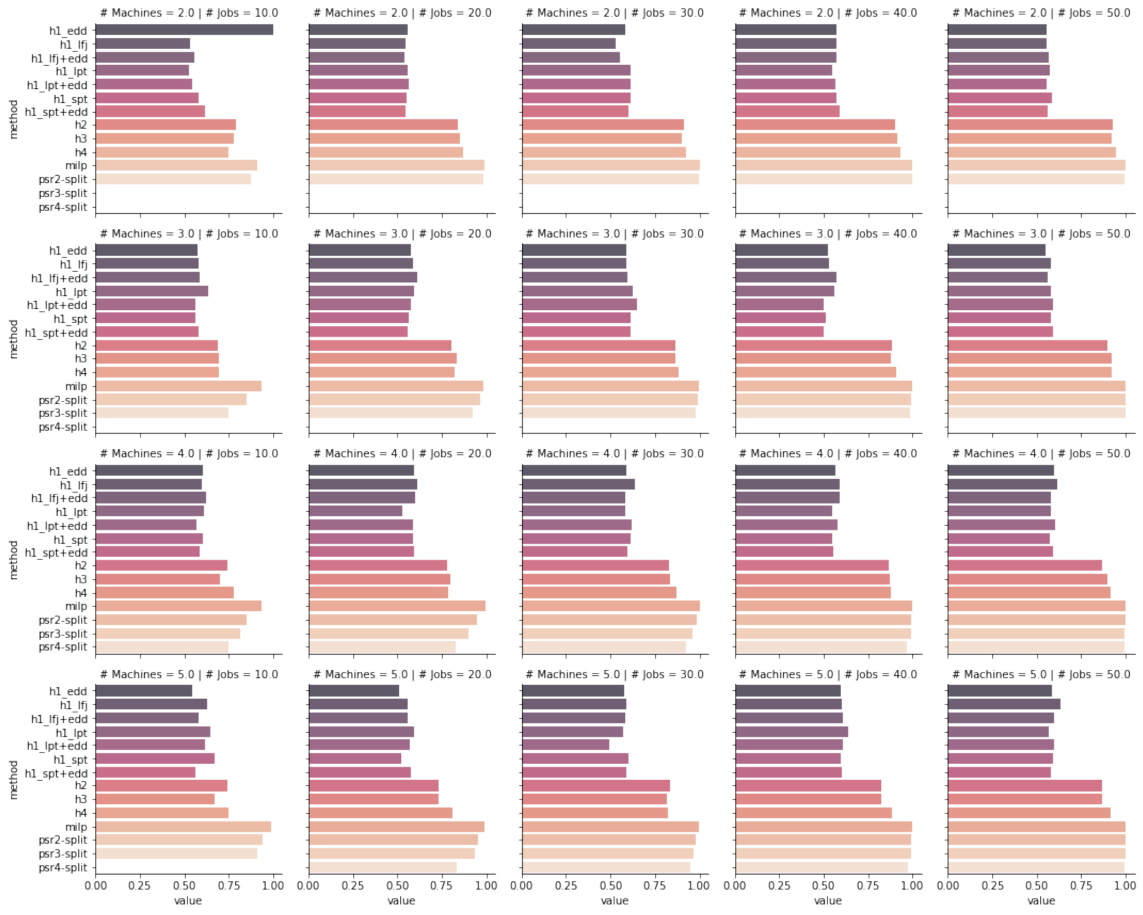

Figure 4.9: Color preference small datasets values

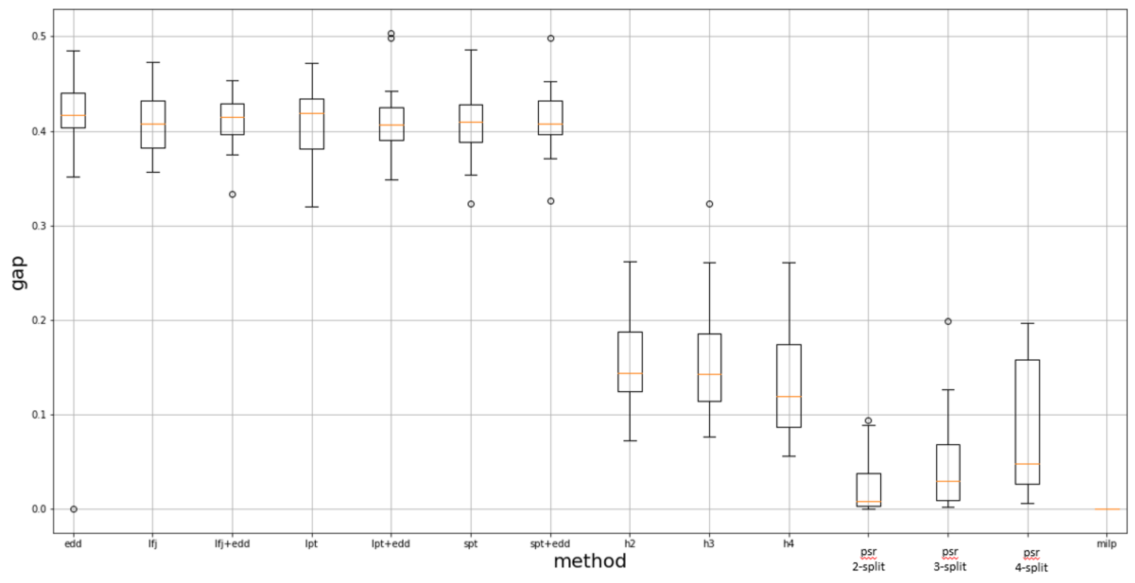

Figure 4.10: Color preference small datasets box-plots for gap

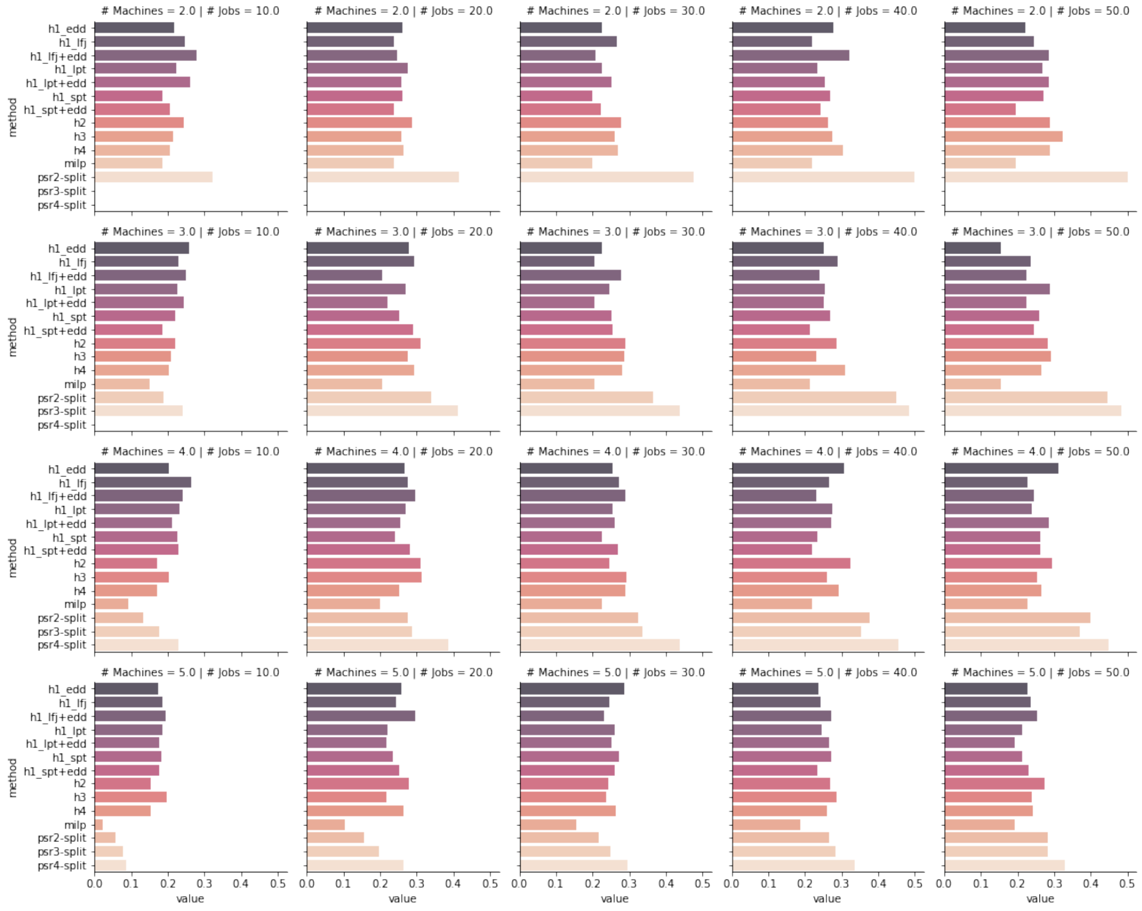

Figure 4.11: Shade consistency small datasets values

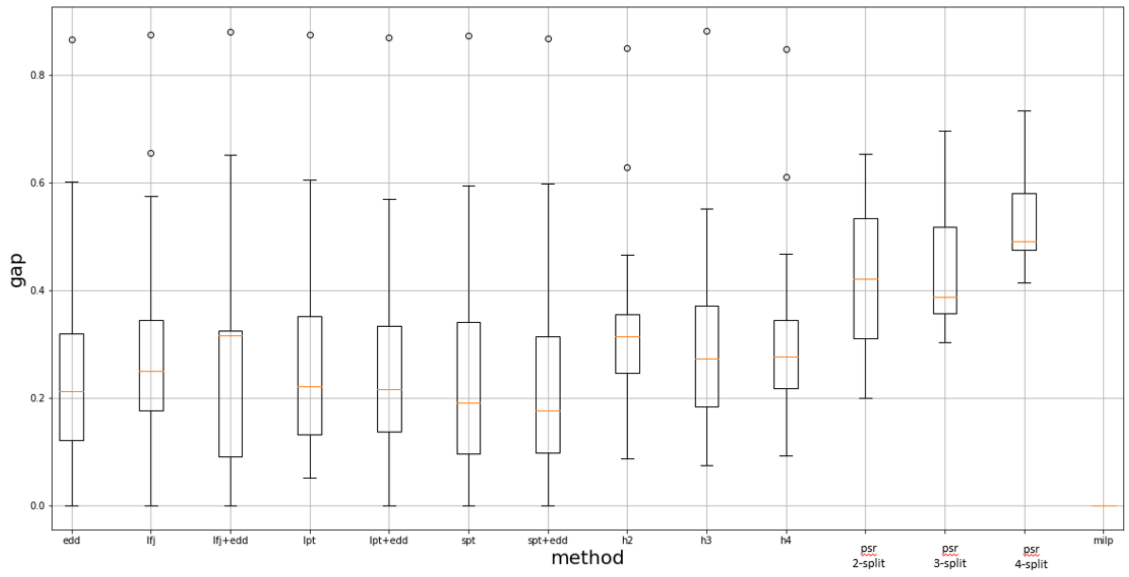

Figure 4.12: Shade consistency small datasets box-plots for gap

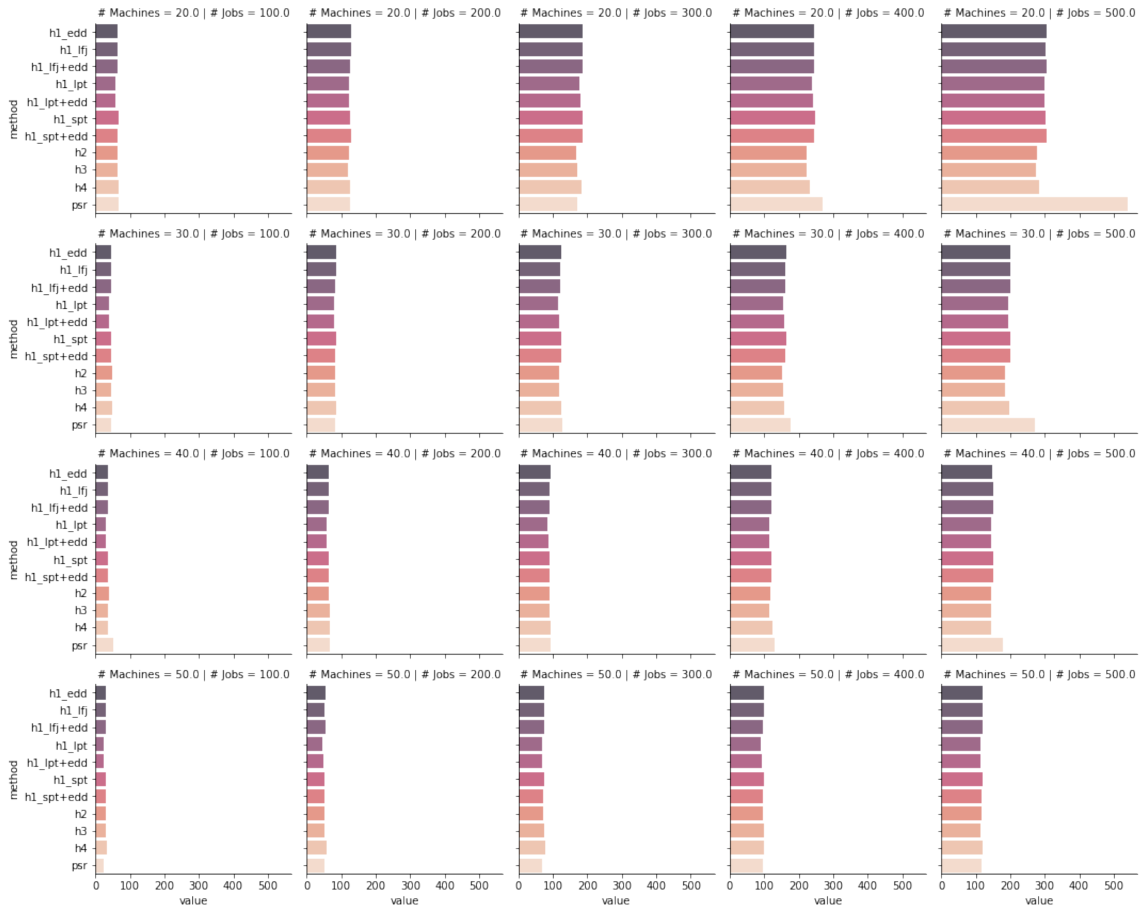

Figure 4.13: Makespan large datasets values

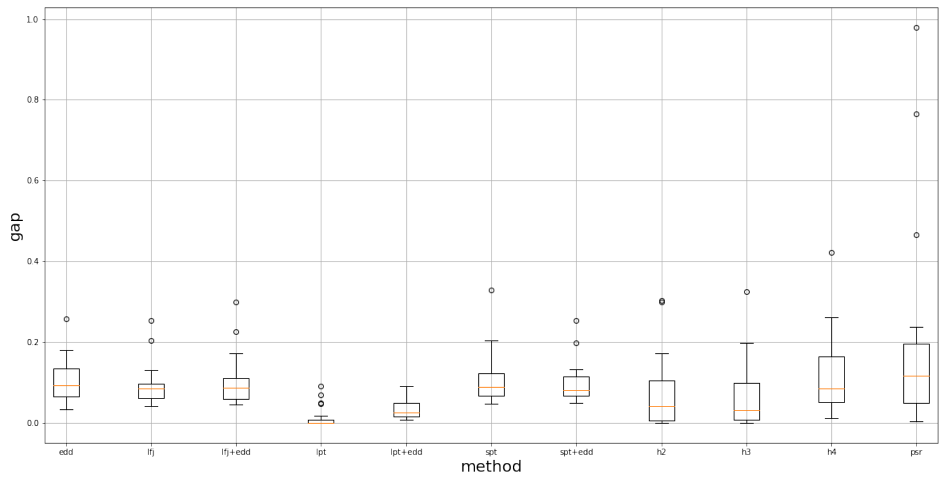

Figure 4.14: Makespan large datasets box-plots for gap

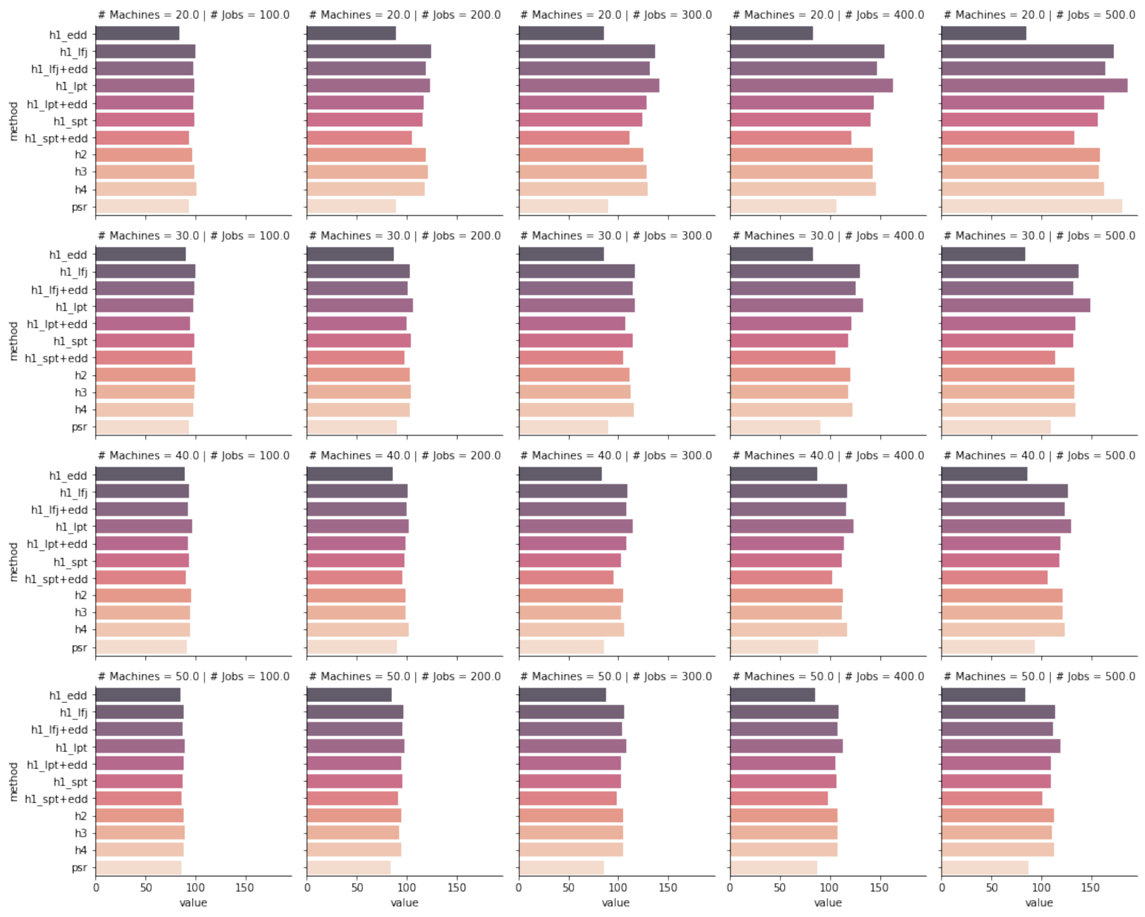

Figure 4.15: Lateness large datasets values

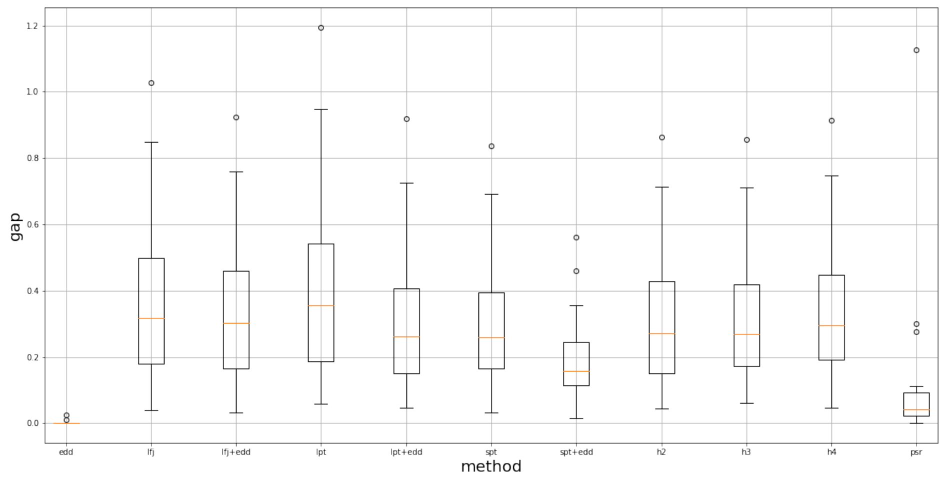

Figure 4.16: Lateness large datasets box-plots for gap

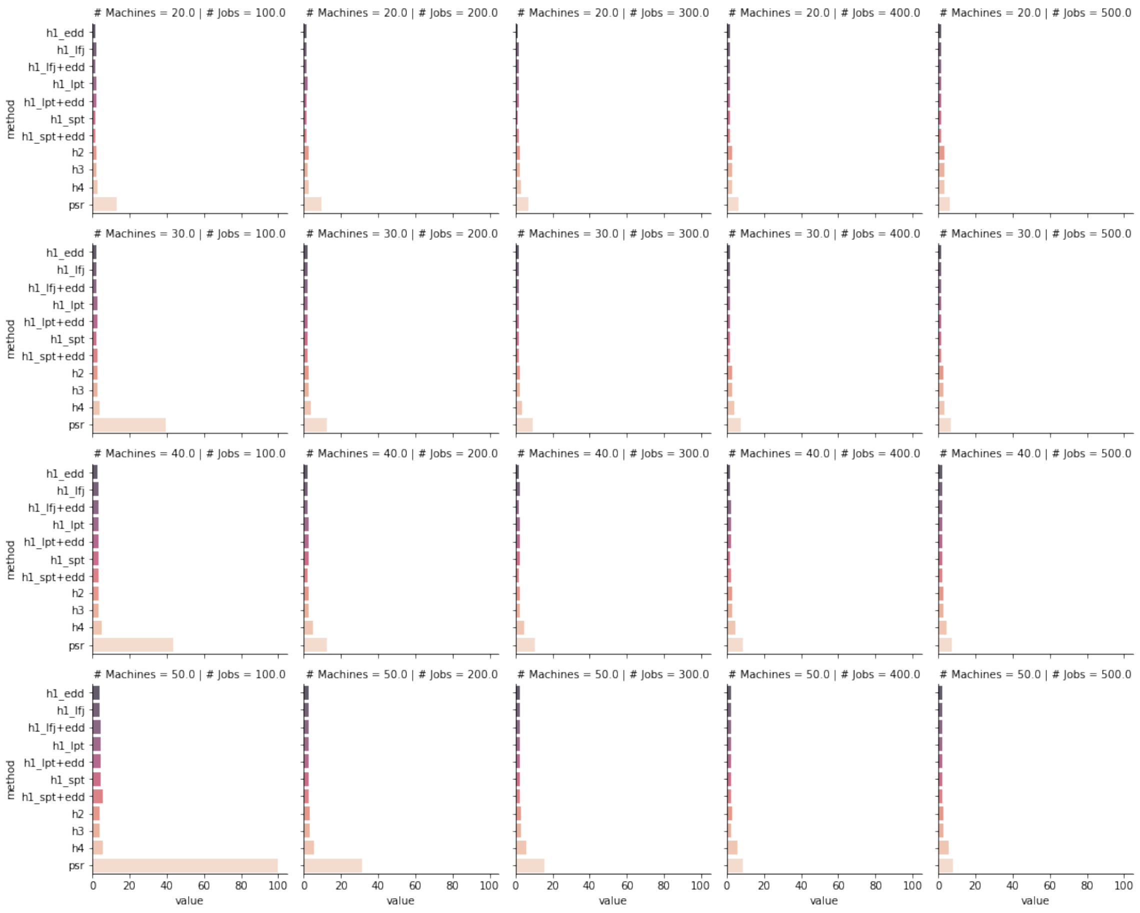

Figure 4.17: Setup ratio large datasets values

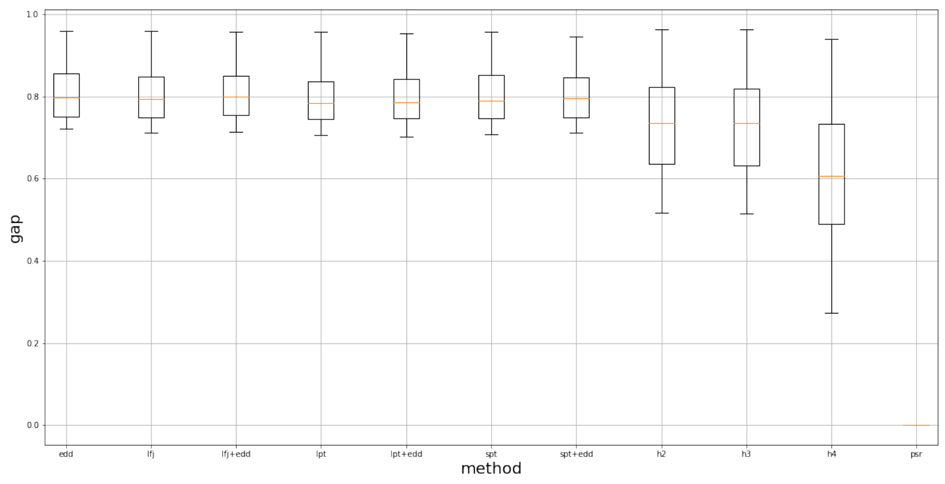

Figure 4.18: Setup ratio large datasets box-plots for gap

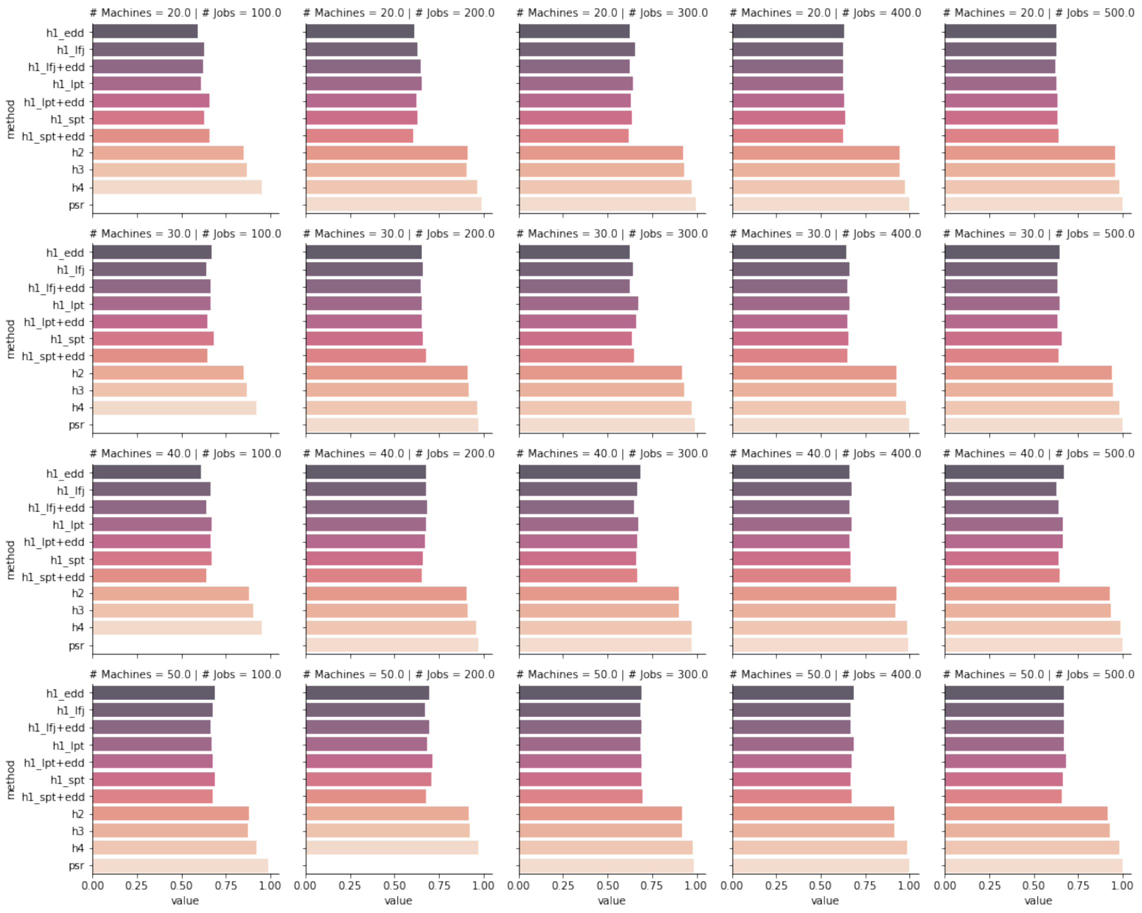

Figure 4.19: Color preference large datasets values

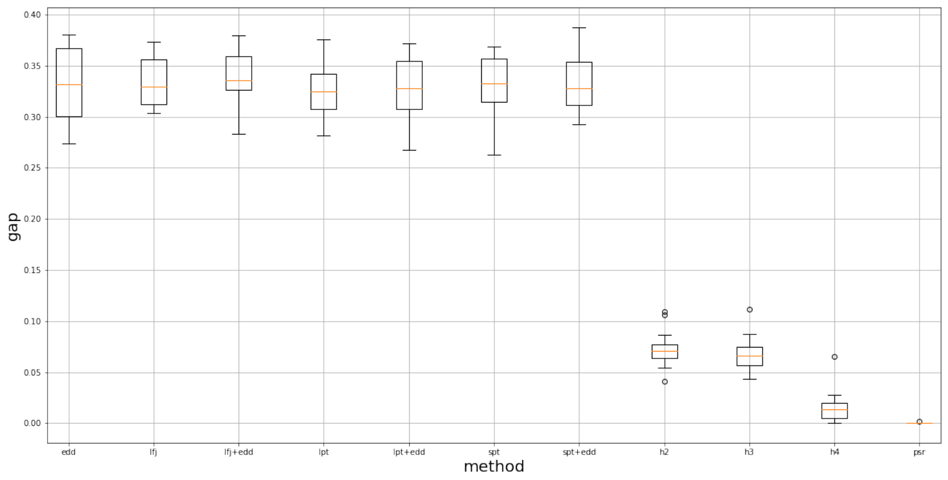

Figure 4.20: Color preference large datasets box-plots for gap

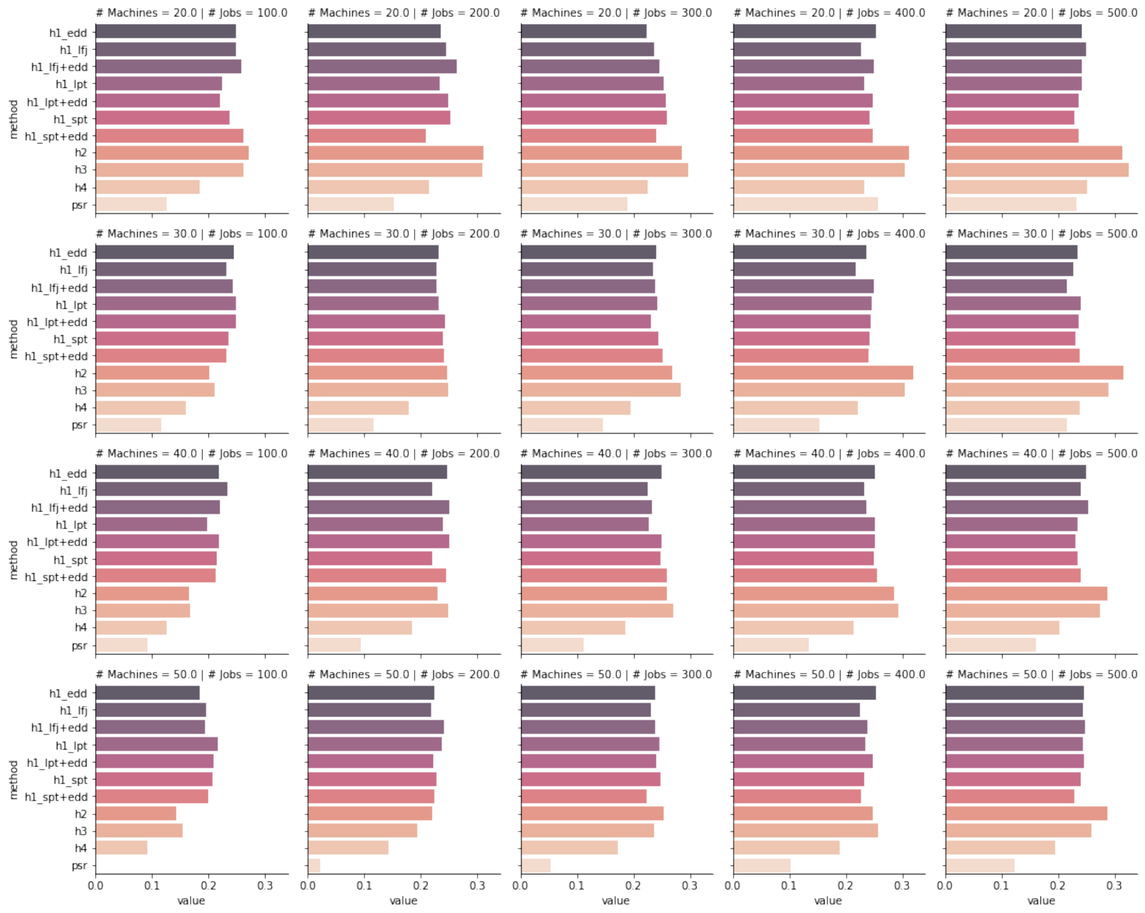

Figure 4.21: Shade consistency large datasets values

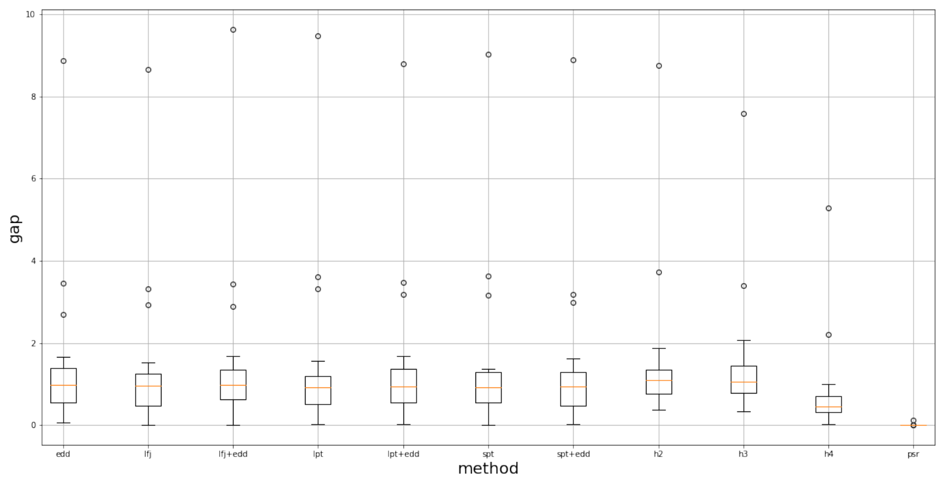

Figure 4.22: Shade consistency large datasets box-plots for gap

Chapter 5

Meta-heuristics

Up until this point in the dissertation, we have addressed the multi-objective problem on our hands by treating each objective individually and not in a strictly multi-objective sense. But, it is of interest to us and the industrial sponsor for this project to obtain schedules that perform well on all objectives at the same time, even if they are not optimal for any single objective. This chapter describes the adaptation of two multi-objective meta-heuristic methods - NSGA-II and MOEA-D for our problem. These are very popular multi-objective genetic algorithms. However, these methods cannot directly be applied to our problem due to unique sequence-dependent restrictions while placing the jobs. So, an additional layer is introduced into the methods. This layer handles the job placement using the information from the chromosome. In this chapter, we first try a construction heuristic-based approach to job placement. Then, we develop a hybrid-optimal method where the information from a chromosome is fed to a MILP which provides the best possible job placement for that chromosome. These two methods, along with the two approaches to job placement, are compared in a multi-objective perspective to obtain the best performing meta-heuristic method across the five objectives.

5.1 The Story of the chapter

At the beginning of the project with the industrial sponsor, we learned about their requirements, constraints, etc., and developed a base MILP formulation. Since this formulation did not work well to solve problems at scale, we then developed heuristic methods that could generate solutions quickly. These solutions were acceptable to the industrial sponsor (mainly H3 and H4 solutions). But, there was one restriction to these methods. For any set of jobs and machines, they provided a single solution. They were construction heuristics. It was of interest to the industrial sponsor to have options of looking at multiple different schedules to evaluate trade-offs and then pick one schedule to implement in their manufacturing plant for that day or week. This was necessary because their priorities used to change from day to day. Sometimes, when there was a backlog of jobs, it was necessary to prioritize the minimization of delays. At other times, when they wanted to reduce the usage of cleaning chemicals, it was essential to prioritize minimization of setups. At other times, they wanted to concentrate on makespan or quality. With the H3 and H4 generated schedules, there was no way to evaluate these tradeoffs because there was only one schedule available. So, for the first time in our project, we needed to evaluate our problem with multiple objectives from a multi-objective perspective. If we had two objectives and our MILP was easy to solve, we could have implemented weighted sum or epsilon constraint methods and obtained the Pareto front of solutions and visualized them. However, with five objectives, a usable MILP formulation on 2 of the objectives, and our industrial sponsor's reluctance towards using MILP based methods, we chose to implement a multi-objective meta-heuristic framework that could provide multiple solutions with varying levels of performance on each objective, and also provide these solutions in a reasonable time frame (if not in a few seconds like the construction heuristics). This method was going to be a backup for the planners at the company. If the H3 and H4 schedules failed to meet their requirements for that day/ week, they were willing to spend additional time exploring more schedules and select one which fits. Within the scheduling literature for multi-objective meta-heuristics, NSGA-II has been extremely popular. So, as a first step, we decided to implement the NSGA-II meta-heuristic method for our problem. But, it was impossible to create any chromosome type that could translate directly to a feasible schedule for our problem. So, we adapted the framework to our problem by introducing a new layer that could create feasible schedules from the chromosomes given to it. We used the H1 construction heuristic as the new layer within the NSGA-II framework to obtain feasible schedules and then evaluated them. Then, we found a problem similar to ours from the literature, and they had proposed another meta-heuristic framework that they claimed worked well for their problem. This was the MOEA-D framework. So, next, we implemented the MOEA-D framework with the same adaptations as we had done for the NSGA-II framework. Next, we tested both the MOEA-D and NSGA-II methods on the company's order data for several weeks. At the end of the tests, there was no conclusive evidence of which method was better. However, the company was of the opinion that NSGA-II was simpler and easier to maintain. So, they implemented the NSGA-II method with the H1 construction layer, and they were satisfied with the results. However, as a research exercise, we wanted to evaluate these methods rigorously using single objective as well as multi-objective evaluation techniques. So, we created extensive testing data and obtained optimal solutions from our now-improved MILP methods for small problems. We rigorously tested both the meta-heuristic methods. After testing and analyzing the methods, we found the advantages and disadvantages of using both methods. We also found that both methods could not find optimal solutions consistently on any of the objectives, i.e., there was scope for improvement in the quality of the solutions. So, as a research exercise, we developed a hybrid-optimal method of creating the schedules. We chose a chromosome in such a way that the meta-heuristic framework made the scheduling decisions; however, the sequencing decisions were made using our improved MILP methods. With the help of this method, we were able to produce significantly improved solutions to the same problems over all previously tested methods.

In the following sections of this chapter, we first introduce the multi-objective frameworks we tested (NSGA-II and MOEA-D). Then, we discuss the types of chromosomes and genetic operators that we used. Then, we describe the scheduling layers added to the frameworks. Next, we describe how all the pieces work together as step-by-step algorithms. Next, we describe the design of experiments and evaluation metrics used to analyze the performance of the meta-heuristic methods from both single and multi-objective perspectives. Then, we discuss the results obtained. As a final exercise, we pick the best method among all those tested and use it to provide schedules for the largest datasets we used to test the construction heuristics. We then compare the best meta-heuristic to the construction heuristics for all the objectives.

5.2 Problem Definition and Adaptation of Metaheuristics

5.2.1 Definition and Formulation

This study addresses a multi-objective parallel machine scheduling problem with the objectives of minimizing makespan, minimizing lateness, minimizing setup times, minimizing shade inconsistency, and maximizing color preferences. The machines are the fabric dyeing machines in a textile dyeing facility, and the jobs are the fabrics to be dyed. The machines perform the dyeing process depending on the fabric characteristics. The setups are the cost-intensive cleaning process of machines which is required depending on several factors. The other specific features of the problem are as follows:

- There are m nonidentical machines.
- There are n independent jobs that are available at time 0.
- Each job is to be processed on only one machine and each machine can process at most one job at a time.
- Each machine can only process a subset of jobs.
- Each job has a deterministic processing time p_i.

- There are sequence dependent machine setup times β_{i_1,i_2} between job i_1 and job i_2 depending on the color and shades of the jobs where $\beta_{i_1,i_2} \neq \beta_{i_2,i_1}$.
- There are sequence dependent restriction constraints.
- Each job has a shade value s_i and a shade group s_i^g. The shade value ranges from 0 to 100 and the shade groups are very dark (1-25), dark (26-50), light (51-75), and very light (76-100).
- There are preferences between jobs depending on the colors γ_{i_1,i_2}.

As aforementioned, this study is conducted to solve the same problem given in [90] using a Pareto optimality framework which is more convenient for multi-objective problems. For the sake of clarity and to avoid repetition, the indices, input parameters, and decision variables used in the model are summarized in Chapter 3. The problem requires the assignment of n jobs to m machines and determining the order of jobs on the assigned machines. According to the above problem characteristics and notations, the formulation of the corresponding parallel machine scheduling problem is given in 3.

The objectives in a multi-objective environment will often conflict with each other. Thus, there is not any solution that optimizes all objectives at once. The term *Pareto optimality* is defined to show the tradeoffs among objectives and to balance them. Define the objective set as $F(x) = f_1(x), .., f_m(x)$ where m is the number of objectives. A Pareto optimal solution is a solution that is not dominated by any other solution, i.e., a point x^* is Pareto optimal if there is not any other point x such that x dominates x^*. And x^* dominates x if $x_i^* \leq x_i$ for all $i \in \{1, .., m\}$ and $x_o^* < x_o$ for at least one o, where all objectives are minimization. Multi-objective evolutionary algorithms (MOEAs) aim to find a set of representative Pareto optimal solutions. Experimental studies by [112] for the behaviors and performances of MOEAs show that they solve multi-objective problems fast and efficiently. The main advantage of MOEAs is that they can find a set of Pareto optimal solutions in a single run. Based on the multi-objective nature of our scheduling problem and the success of MOEAs, we employed the framework of two well-known MOEAs, namely

Non-dominated Sorting Genetic Algorithm-II (NSGA-II) and Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D), which are introduced in the remainder of this section.

5.2.2 The Non-dominated Sorting Genetic Algorithm-II

NSGA (Non-dominated sorting genetic algorithm) is a multi-objective evolution-based meta-heuristic method introduced by [91]. NSGA-II [34] by is an improvement over NSGA in terms of computational complexity and elitism. NSGA-II is one of the most popular multi-objective meta-heuristic methods used in literature. NSGA-II has been used by [103] to reduce errors in a multi-objective wireless sensor network positioning problem. A bi-objective model to optimize water resources in a reservoir is solved using NSGA-II by [74] to maximize the minimum water level and maximize the power output. They use a lion-pride-based modification NSGA-II to improve the diversity of solutions. [105] have proposed a modified NSGA-II algorithm based on reinforcement learning to solve a multi-objective problem in an off-grid wind energy system. They have three objectives: minimizing energy cost, minimizing loss of power, and minimizing power abandonment. Stochastic multi-objective problems are solved using NSGA -II combined with simulation modeling in a hazardous waste management environment by [82]. They aim to minimize the cost of routing waste as well as minimizing the environmental risk involved in routing the hazardous wastes. A multi-objective task scheduling problem of minimizing costs and minimizing the makespan of the systems is solved using NSGA-II by [89]. An objective-specific variation of NSGA-II is used by [92] for a vehicle routing problem with time windows with objectives of minimizing the number of vehicles, makespan, delays, etc. A healthcare scheduling problem with objectives of minimizing wait times and improving customer satisfaction is solved using NSGA-II by [5]. In a job shop scheduling environment, a multi-objective problem of minimizing worker fatigue and minimizing the schedule makespan is solved using an enhanced NSGA-II, which is fed a starting population consisting of high-quality solutions by [93].

The general framework for this meta-heuristic is described in algorithm 6. The method starts with an initial parent population of Q feasible solutions. If feasibility is not confirmed, then feasibility is included as one of the objectives, i.e., a feasible solution is better than (dominates) an infeasible solution. Next, this parent population is used to create an equal-sized (Q) child population using genetic operators and mutations modeled by the user. This initial parent and child population is the basis for the first iteration of the algorithm. The entire population is subject to the creation of Pareto fronts, i.e., the set of non-dominated solutions forms the first Pareto front. Apart from the set of solutions in the first Pareto front, the set of non-dominated solutions in the remaining solution set forms the second Pareto front and so on. Next, the top Q solutions are used as the parent population for the next iteration. If solutions have to be picked from a single Pareto front to reach the Q limit, then the crowding distance measure by [34] is used to sort the solutions. This operator ensures an even spread among the solutions selected. Using the new parent population, a new child population is created for the second iteration. This process is continued until convergence is reached, an iteration limit or time limit is reached, or a certain objective value is achieved, as determined by the user.

Algorithm 6 NSGA - II general framework

```
Obtain initial parent population of size $Q$
Create a child population of size $Q$ using initial parent population
while Solutions converge/ User limit is not reached do
    Evaluate each member of the entire population
    Create Pareto fronts from all the members of the population
    Sort each Pareto front using the crowding distance measure
    Select the top $Q$ members using the Pareto front and crowding distance measures
    Create a new child population of size $Q$ for the next iteration
end while
```

For meta-heuristic algorithms, one of the measures of performance is how many solutions it can generate in the first Pareto front. A drawback of NSGA-II is that the number of solutions available at the end of the test run (unless convergence is reached) is limited in size to Q because only Q solutions are carried over at the end of each iteration. This drawback is also mentioned in literature by [39, 77], etc. A fix for this drawback is presented by both [39, 77]. Their fix is to collect solutions in an external population set. This set accumulates non-dominated solutions along the run of the algorithm. We also implemented this fix as an additional algorithmic framework during our exploration of methodologies. We also have used a modified NSGA-II algorithm called NSGA-II-EP, which creates an external population of solutions. At the end of every iteration of the NSGA-II algorithm, this modified algorithm adds all the non-dominated solutions found to the external population. This helps identify all the non-dominated solutions found during the entire run of the algorithm.

Algorithm 7 NSGA - II - EP general framework

```
Obtain initial parent population of size $Q$
Create a child population of size $Q$ using initial parent population
Create an empty set of solutions called EP (external population)
while Solutions converge/ User limit is not reached do
    Evaluate each member of the entire population
    Create Pareto fronts from all the members of the population
    Sort each Pareto front using the crowding distance measure
    Select the top $Q$ members using the Pareto front and crowding distance measures
    Add all non-dominated solutions to the EP set
    Remove all dominated solutions from the EP set
    Create a new child population of size $Q$ for the next iteration
end while
```

5.2.3 Multi-objective Evolutionary Algorithm Based on Decomposition

MOEA/D is a decomposition-based multi-objective evolutionary algorithm proposed by [109]. Since the MOEA/D was first introduced by [109], several versions were developed to solve combinatorial optimization problems and compared with other best-known MOEAs. A comparison of NSGA-II and MOEA/D is presented by [66] for complicated Pareto sets and shows that the decomposition-based meta-heuristics can deal with complex problems. [57] develops a hybrid algorithm by combining the decomposition method of MOEA/D and the pheromone matrix of ant colony optimization (ACO). [65] adapt a neural network algorithm as a local search within MOEA/D.

MOEA/D decomposes a multi-objective problem into Q single-objective optimization problems and solves scalar problems simultaneously by only using the information of neighbor sub-problems. An aggregation coefficient vector (λ_i) is employed to define the neighborhood (B_i) of each sub-problem i. The neighborhood of a sub-problem includes the T closest sub-problems in terms of the Hamiltonian distance among the coefficient vectors, where T is an integer constant that determines the size of the neighborhood. In this study, the MOEA/D framework is used as given by [109], yet the decomposition method is modified, see algorithm 8.

There are several methods to decompose a multi-objective problem into scalar sub-problems, e.g., Weighted Sum Approach by [78], Tchebycheff Approach by [78], Boundary Intersection Approach by [32]. The range of the objective values for our problem varies (i.e., not all objective values are in the range of [0,1]), which induces different impacts of objectives while solving the scalar problems. Accordingly, we employ the Tchebycheff approach since it allows us to normalize objective values. The modified Tchebycheff approach with the normalized distances is as follows:

$$minimize\ g(x|\lambda, z^*, z^-) = \max_{1\leq i\leq m} \left\{ \frac{\mid f_i(x) - z_i^* \mid \lambda_i}{\mid z_i^* - z_i^- \mid} \right\} \tag{5.1}$$

where z^* and z^- are the reference point for the best and worst found objective values, respectively.

Algorithm 8 Pseudo-code of MOEA/D

```
Step 1) Initialization
Set external population: $EP = \emptyset$
Calculate Euclidean distances between weight vectors for each pair and determine the neighborhoods
Generate an initial population of size $Q$
Calculate objective values of each solution
Determine the best $(z^*)$ and the worst $(z^-)$ found values for each objective
while Stopping criteria is not met do
  Step 2) Update
  for $i = 1, .., Q$ do
    Apply a genetic operator to generate a new solution $y$ using two neighbors of solution $i$
    Update of $z$: if $z_j < f_j(y)$, then set $z_j = f_j(y)$ for each $j = 1, .., m$
    Update of neighboring solutions: for each $j$ in $B(i)$, if $g(y|\lambda^j, z) \leq g(x^j|\lambda^j, z)$, then set $x^j = y$.
    Update of EP: Add $F(y)$ to the EP if $y$ is not dominated by any solutions in EP and remove all the solutions dominated by $y$ from EP.
  end for
end while
Output EP.
```

5.3 Algorithm Design

To fairly compare the performance of the two MOEAs explored in this paper, many algorithm design details are used in common. This section describes those common aspects, including chromosome designs, initialization techniques, feasible schedule generation methods, and genetic operators. To summarize, we present two feasible schedule generation methods integrated into the MOEA frameworks. Each method adopts a unique initialization technique, chromosome design, and genetic operator.

5.3.1 Chromosome design

In this subsection, we discuss the creation of the initial population using two different methods. The first method is by creating every member of the initial parent population as a random order of jobs. For example, consider the first chromosome design shown in Figure 5.1. In this sample chromosome, we have ten jobs. The first job that will be assigned to a machine is job 5. Then job 4 is assigned to a machine and so on. Similarly, by changing the order of jobs in the chromosome, we can create different schedules.

We create Q chromosomes to constitute the initial parent population. The genes (jobs) in seven chromosomes are ordered according to ascending order of processing times (SPT), descending order of processing times (LPT), ascending order of due dates (EDD), ascending order of machine flexibility (LFJ), and four combinations detailed in the H1 heuristics section of the study by [90], i.e., SPT+EDD, LPT+EDD, LFJ+EDD. The other $Q - 7$ chromosomes are created at random. There has to be an additional step for this method to ensure that each chromosome represents a feasible solution. In the following subsection, we explain how a chromosome can be used to create a feasible solution.

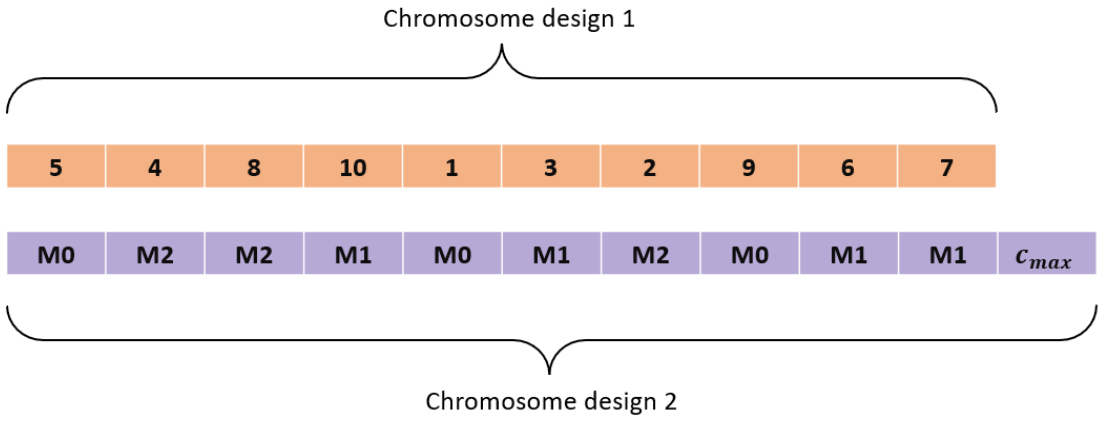

Figure 5.1: Chromosome designs

The second method of creating the initial set of chromosomes is by assigning machines to jobs based on their machine eligibility information. Consider an example of 10 jobs. A sample chromosome for these ten jobs is shown in chromosome design 2 in Figure 5.1. In this chromosome, the first job is assigned machine M0. The second job is assigned machine M2 and so on. The last gene in this chromosome contains the objective on which this solution will be evaluated. In the given example, the algorithm will minimize the makespan for this solution. While this chromosome has machine information for each job, and it is feasible from the eligibility perspective, we still do not know the sequence of jobs on each machine. To obtain feasible (or infeasible) schedules, we use a hybrid optimal method of treating each machine as a single machine problem and create sequences for each machine using a MILP with the objective specified by the last gene of the chromosome, which is explained in subsection 5.3.2.2.

5.3.2 Solution creation

5.3.2.1 Construction heuristic for job placement

The chromosome must be decoded into a schedule that can be evaluated based on the five objectives. In algorithm 9, we present a construction heuristic, an adaptation of the H1 heuristic explained in [90] to attempt to create a feasible schedule from a chromosome. When a chromosome cannot produce a feasible solution, we discard that chromosome and obtain another random chromosome until all chromosomes represent a feasible schedule.

Algorithm 9 Construction Heuristic for job placement

```
Obtain list of jobs
for every job $i$ in list of jobs do
  Obtain list of eligible machines
  for every machine $j$ in list of eligible machines do
    Finish time for job $i$ on machine $j$ = $\begin{cases} \text{finish time of current job on machine } j + \\ \text{setup time between current job and job } i + \\ \text{processing time for job } i \end{cases}$
  end for
  Place job $i$ on the machine where it finishes first
  Update current job and finish time on the chosen machine
end for
```

5.3.2.2 Hybrid-optimal heuristic for job placement

In this subsection, we first describe a single machine sequencing MILP which ensures a feasible sequence of jobs on each machine. This MILP uses the same parameters as the MILPs described in Chapter 3.

The base variables for this formulation include the start variables y^{strt}, the end variables y^{end}, and the sequencing variables y. Additionally, we have the c variables to capture completion times.

$$y_i^{strt} = \begin{cases} 1, & \text{if job } i \text{ is the first job in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

$$y_i^{end} = \begin{cases} 1, & \text{if job } i \text{ is the final job in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

$$y_{i_1,i_2} = \begin{cases} 1, & \text{if job } i_1 \text{ is processed immediately before } i_2 \\ 0, & \text{otherwise} \end{cases}$$

c_i = finish time for job i

We define the following constraints to generate feasible sequences for each machine. Constraint set (5.2) ensures that if a job is the beginning of the sequence, then it has no predecessors. Constraint set (5.3) ensures that if a job is the last job in a sequence, then it has no job following it. Constraint set (5.4) ensures that no job follows or precedes itself. Constraint set (5.5) ensures that the completion time for a job is not less than its processing time. Constraint sets (5.6) and (5.7) ensure that every sequence has a first and a last job. Constraint set (5.8) ensures that for every pair of jobs, either the first job can follow the second, or the second can follow the first, but both cases cannot happen at the same time. Constraint set (5.9) ensures that there is no overlap of jobs in sequence. Constraint sets (5.10), (5.11), (5.12) ensure binary restrictions on the variables. Constraint sets (5.13), (5.14) ensure that the shade difference between any two jobs that immediately precede or succeed each other is not too drastic (limited to 60 for this specific study).

$$y_i^{strt} = 1 - \sum_{i_1 \in I} y_{i_1,i} \quad \forall i \tag{5.2}$$

$$x_i^{end} = 1 - \sum_{i_1 \in I} y_{i,i_1} \quad \forall i \tag{5.3}$$

$$y_{i,i} = 0 \quad \forall i \tag{5.4}$$

$$c_i \geq p_i \quad \forall i \tag{5.5}$$

$$\sum_i y_i^{strt} = 1 \tag{5.6}$$

$$\sum_i y_i^{end} = 1 \tag{5.7}$$

$$y_{i_1,i_2} + y_{i_2,i_1} \leq 1 \quad \forall i_1, i_2 \tag{5.8}$$

$$c_{i_1} \leq c_{i_2} - p_2 - \beta_{i_1,i_2} + M(1 - y_{i_1,i_2}) \quad \forall i_1, i_2, j \tag{5.9}$$

$$y_{i_1,i_2} \in \{\ 0, 1\} \quad \forall i_1, i_2 \tag{5.10}$$

$$y_i^{strt} \in \{\ 0, 1\} \quad \forall i, j \tag{5.11}$$

$$y_i^{end} \in \{\ 0, 1\} \quad \forall i, j \tag{5.12}$$

$$(s_{i_1} - s_{i_2}) y_{i_1,i_2} \leq 60 \quad \forall i_1, i_2 \tag{5.13}$$

$$(s_{i_2} - s_{i_1}) y_{i_1,i_2} \leq 60 \quad \forall i_1, i_2 \tag{5.14}$$

There are five objectives in this problem: minimizing makespan, minimizing lateness, minimizing setups, maximizing color preferences, and minimizing shade inconsistency.

For both the makespan and setup time objectives, we use the objective function that minimizes setups (5.15). We can use the same objective because for a single machine sequencing problem, minimizing setup times is the same as minimizing the makespan.

$$\text{Setup times:} \quad \min \sum_{i_1,i_2} \beta_{i_1,i_2} y_{i_1,i_2} / (n - m) \tag{5.15}$$

The lateness objective is used to minimize delays. To record the delay for each job, we introduce the lateness variables as defined in (5.16). The objective (5.17) is to minimize the sum of delays for all the jobs. Constraint set (5.18) ensures that the delay is greater than or equal to 0. This ensures that we do not provide an incentive to jobs that are finished ahead of their due dates. Constraint set (5.19) ensures that the delay for a job is greater than the difference between the completion date and due date of the job.

$$u_i = \text{lateness for job i} \tag{5.16}$$

$$\text{Lateness:} \quad \min \sum_i u_i \tag{5.17}$$

$$u_i \geq 0 \quad \forall i \tag{5.18}$$

$$u_i \geq c_i - d_i \quad \forall i \tag{5.19}$$

To maximize color preferences, we use the same objective (5.20) as in [90].

$$\text{Average Color Preference:} \quad \max \sum_{i_1,i_2} \gamma_{i_1,i_2} y_{i_1,i_2} \tag{5.20}$$

The objective of reducing shade inconsistency is also handled by the addition of 2 sets of variables defined as (5.21) and (5.22).

$$s^d_{i_1,i_2} = \text{shade difference between successive jobs } i_1 \text{ and } i_2 \tag{5.21}$$

$$s^{gd} = \text{ shade group difference from first to last job} \tag{5.22}$$

The objective as described in (5.23) is a minimization of a sum of two parts - the first part aims to reduce differences in shades for jobs in sequence, and the second aims to reduce differences between shade groups between the first and last jobs.

$$\text{Average Shade Consistency:} \quad \min \sum_{i_1,i_2} s^d_{i_1,i_2}/200(n-1) + s^{gd}_j/6 \tag{5.23}$$

To ensure that the values for the shade difference and shade group difference variables are feasible, we introduce constraints (5.24) to (5.27). Constraint set (5.24) ensures that the shade difference between a pair of jobs in immediate succession on the same machine is recorded by the shade difference variable, and constraint set (5.25) ensures that the difference is greater than zero. Constraint sets (5.26) and (5.27) ensure that the shade group difference variable captures the shade group difference between the first and last job.

$$s^d_{i_1,i_2} \geq (s_{i_2} - s_{i_1})y_{i_1,i_2} \quad \forall i_1, i_2 \tag{5.24}$$

$$s^d_{i_1,i_2} \geq 0 \quad \forall i_1, i_2 \tag{5.25}$$

$$s^{gd} \geq \sum_i s^g_i y^{strt}_i - \sum_i s^g_i y^{end}_i \tag{5.26}$$

$$s^{gd} \geq \sum_i s^g_i y^{end}_i - \sum_i s^g_i y^{strt}_i \tag{5.27}$$

This MILP receives a set of jobs along with the job information and creates feasible sequences among the jobs, and optimizes them based on each of the five objectives. If there is no feasible solution for a certain machine, the chromosome is discarded as infeasible, and a new random chromosome replaces it in the population.

Algorithm 10 uses this single machine MILP to create feasible solutions for the global problem of parallel machines. In this algorithm, we first obtain the chromosome from the meta-heuristic framework during either initialization or when evaluating new solutions for every iteration. This chromosome has a machine assignment for every job as well as an objective gene. We compute a feasible sequence of jobs for every machine using the single machine MILP. If the MILP cannot find any feasible solution, this chromosome is discarded and replaced by a new chromosome. After each machine has a feasible sequence, the entire schedule is compiled. Note that even though we use a MILP to solve the objective for each machine, the compiled schedule is not necessarily optimal for the entire problem. The meta-heuristic framework looks to find the best machine placement for each job, while the single machine MILP ensures that each machine has an optimal solution relative to a single objective given the set of jobs assigned to it or the best solution found within a pre-specified time frame.

Algorithm 10 Hybrid optimal heuristic for job placement

```
Obtain chromosome
Initialize an empty schedule for each machine
for every machine $i$ in set of machines do
    Obtain the list of jobs in the chromosome for machine $i$ and the objective for the chromosome
    Use single machine MILP to obtain an optimal/feasible sequence for jobs on machine $i$
    if sequence not infeasible then
        Add feasible/optimal machine sequence to the final schedule for machine $i$
    else
        Discard chromosome and replace by new chromosome
        Run hybrid algorithm again from the beginning
    end if
end for
```

5.3.3 Genetic operators and crossover

In the previous section, we described two methods for population initialization. We also used two different types of chromosomes. In this section, we explain two types of genetic operators to obtain new members of the population. Both these operators take two parent chromosomes and provide two-child chromosomes.

5.3.3.1 Partially matched crossover

For the first type of chromosome, we use a partially matched crossover. This crossover method takes two chromosomes and a random point of the chromosome. Then, it crosses the information from the parent chromosomes to create two new child chromosomes. Consider an example shown in Figure 5.2. In this example, there are two parent chromosomes, P1 and P2, created using chromosome design 1 5.1. For the crossover, a random point is selected. Let this random point be 4, which means there will be a crossover at the fourth job. This results in two children, C1 and C2, retaining the genes in the same order as P1 and P2, respectively, until their fourth jobs. For the slots after the fourth position, C1 obtains jobs in the order defined by P2, whereas C2 obtains jobs in the order defined by P1. According to this crossover, for C1, the first four jobs are obtained from P1 - jobs 5,4,8, and 10. The next six jobs, 1,6,3,2,9 and 7, are obtained from P2. Similarly, C2 is created using P2 as the starting 4 and P1 as the remaining 6.

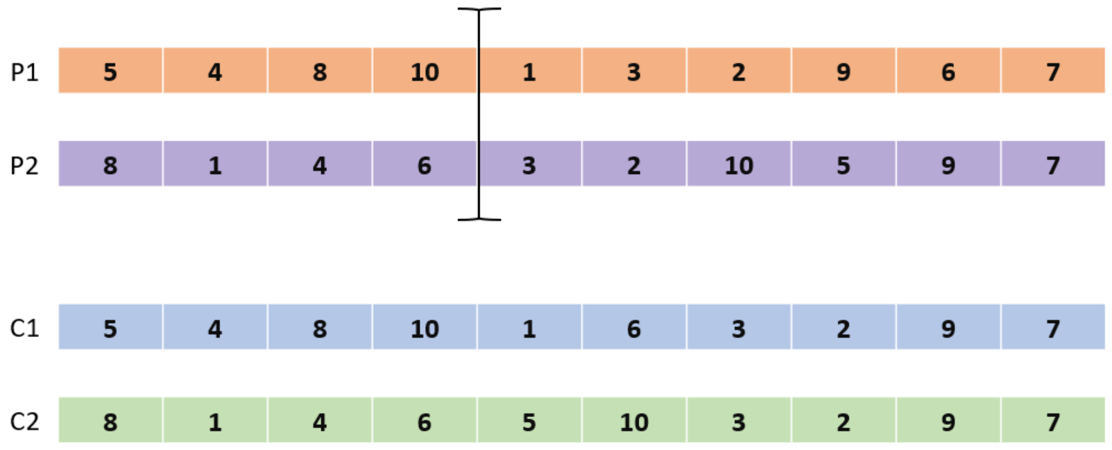

Figure 5.2: Partially matched crossover example

5.3.3.2 Uniform crossover

For the second type of chromosome, we use a uniform crossover. This crossover method takes two-parent chromosomes, then crosses them over at every gene, using a random selection. This crossover also obtains two new child chromosomes. Consider an example shown in Figure 5.3. In this example, we have two parent chromosomes, P1 and P2, created using chromosome design 2 5.1. Then, for crossover, we have a set of random numbers shown in grey. We have the same number of random numbers as the set of genes (jobs and objective). If a random number for a job is greater than or equal to 0.5, we swap the machines for that job. Otherwise, we do not swap. In Figure 5.3, in P1, job 1 is assigned to M0. In P2, job 1 is assigned to M1. The random number for this job is 0.07. Since 0.07 is less than 0.5, we do not swap the machines. For job 2, P1 assigns M2, whereas P2 assigns M0. The random number for this job is 0.89, so we swap the machines. Similarly, we assign machines to each job in the child chromosomes based on the job placements in the parent chromosomes and the random number for each job. As for the last objective chromosome, we again use a random number to determine which objectives will be optimized for the solutions generated from the child genes.

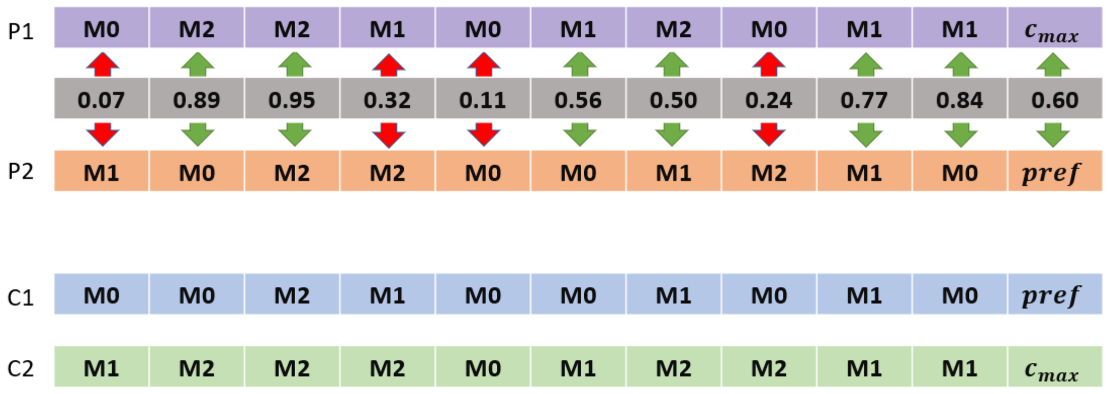

Figure 5.3: Uniform crossover example

5.3.4 Summary of methods tested

In the above sections, we have listed different types of techniques for the creation of the chromosomes, different techniques to obtain feasible solutions, as well as two different meta-heuristic frameworks. In Table 5.1, we summarize the methods as we have tested them.

Method	Meta-heuristic	chromosome design	crossover	solution creation
N-CH	NSGA-II	Design 1	PMX	Construction heuristic
N-CH-EP	NSGA-II with external population	Design 1	PMX	Construction heuristic
N-HO-EP	NSGA-II with external population	Design 2	UFX	MILP-based
M-CH	MOEA-D	Design 1	PMX	Construction heuristic
M-HO	MOEA-D	Design 2	UFX	MILP-based

Table 5.1: Summary of methods

5.4 Design of experiments

We use the same datasets first introduced in [90], i.e., the number of machines ranging from 2 to 5, the number of jobs ranging from 10 to 50, and each configuration of machines and jobs are used to create 10 different datasets to ensure that results are not skewed based on the test parameters such as job colors, due dates, etc. Also, since there is inherent randomness in the meta-heuristic methods, we test each dataset 30 times. The methods we test are: NSGA-II with the construction heuristic (N-CH); NSGA-II-EP with the construction heuristic (N-CH-EP); NSGA-II-EP with the hybrid optimal method for job placement (N-HO-EP); MOEA/D with the construction heuristic (M-CH); and MOEA/D with the hybrid optimal method for job placement (M-HO). We do not test the NSGA-II method with the hybrid optimal approach because this method can never be better than the NSGA-II-EP method because all the solutions obtained from NSGA-II at the end of the final iteration will also be in the EP set at the end of the algorithm run. We also test both the methods for the construction heuristic approach to observe the extent of improvement provided by including the EP methodology. For each method, we limit the number of iterations to 100 without any time limit. We also set the number of solutions explored in each iteration to 50, i.e., $Q = 50$. The MOEAs are implemented in Python 3, and the experiments are performed using the computing nodes of the Clemson University supercomputer with eight cores and 15 GB of memory.

In summary, we have five methods tested across 200 datasets, each 30 times, for a total of 30,000 algorithm simulations. In each simulation, we explore 5,050 solutions.

For the comparison of the meta-heuristic methods with respect to the construction heuristics from [90] we take the best value found by the meta-heuristic for each objective at the end of its simulation and compare it to the solutions obtained by each construction heuristic. We also have the optimal solutions for each objective for the small datasets. We also test the meta-heuristic methods against these optimal solutions for each objective separately. Next, to compare the meta-heuristics among themselves from a multi-objective perspective, we use commonly used performance metrics as explained in the following subsection.

5.4.1 Performance measures

The two evolutionary algorithms are compared in the multi-objective optimization environment in which we deal with Pareto solutions. There are several performance metrics focusing on the following concepts to evaluate the quality of provided solutions as in [111, 6, 86, 51, 29]: (1) The number of non-dominated solutions, (2) The distribution of the solutions in the non-dominated set, (3) The distance among the non-dominated solutions and Pareto-optimal front (if available).

In this study, we used the following performance metrics to measure the criteria mentioned above:

- **The number of non-dominated solutions (NNS):** NNS determines the number of solutions in Pareto front provided by a algorithm, i.e., the number of non-dominated solutions in a set $X^{'}$ of decision vectors.

$$NNS = |a^{'} \in X^{'}; \nexists \overline{a} \leq a^{'}| \tag{5.28}$$

- **Covered size of space (CSS):** CSS (also called *C metric*) is presented by [111]. CSS compares a pair of non-dominated sets by calculating the ratio between the number of solutions in a set dominated by any solutions in the other set and the total number of non-dominated solutions in the set. Let's assume that $X^{'}$ and $X^{"}$ are two sets of decision vectors provided. From the equation 5.29, $C(X^{'}, X^{"}) = 1$ means that all the solutions in $X^{"}$ are dominated or equal to (weakly dominated) the solutions in $X^{'}$.

$$C(X^{'}, X^{"}) = \frac{\mid a^{"} \in X^{"}; \exists a^{'} \leq a^{"} \mid}{X^{"}} \tag{5.29}$$

- **Mean ideal distance (MID):** This performance measure evaluates the distance between the ideal point z^{*} and the solutions in the Pareto front f as given in equation 5.30. The ideal point is determined by solving the problem as a single-objective problem for each objective m optimally, where n is the number of non-dominated solutions. Accordingly, a smaller MID value is better.

$$MID = \frac{\sum_{i=1}^{n} \sqrt{\sum_{j \in m} (f_{ij} - z_{m}^{*})^2}}{n} \tag{5.30}$$

- **Spread of non-dominance solutions (SNS):** SNS measures the distribution of the non-dominated solutions depending on the MID. SNS is the standard deviation of the distance between each solution and the ideal point. Larger SNS value is better since we want to discover solutions distributed over the Pareto front uniformly.

$$SNS = \sqrt{\frac{\sum_{i=1}^{n} \left(MID - \sqrt{\sum_{j \in m} (f_{ij} - z_{m}^{*})^2} \right)^2}{(n-1)}} \tag{5.31}$$

- **Mean ideal gap (MIG):** This performance measure evaluates the optimality gap between the ideal point z^{*} and the solutions in the Pareto front f as given in equation 5.32. A smaller MIG value is better. The benefit of using this metric instead of the MID metric is that this metric also takes into account the different scales associated with each objective, whereas MID only looks at the distance between the solutions and the ideal point. This is a normalized metric.

$$MIG = \frac{\sum_{i=1}^{n} \sum_{j \in m} (f_{ij} \ z_m^*)/ \max(f_{ij}, z_m^*)}{n} \tag{5.32}$$

- **Pareto average gap (PAG):** This performance measure evaluates the optimality gap between the ideal point z^* and the average values of the objectives obtained from the solutions in the Pareto front f as given in equation 5.33. A smaller PAG value is better. This is also a normalized metric. Either PAG or MIG will work towards providing the same information.

$$PAG = \frac{\sum_{j=1}^{m} \frac{\sum_{i=1}^{n} f_{ij}}{n} - z_j^*}{m} \tag{5.33}$$

- **Spread of non-dominance solutions - Gap based (SNSG):** SNSG measures the distribution of the non-dominated solutions depending on the MIG. SNSG is the standard deviation of the optimality gap of each solution and ideal point. Larger SNSG value is better since we want to discover solutions distributed over the Pareto front uniformly.

$$SNSG = \sqrt{\frac{\sum_{i=1}^{n} \Big(MIG - \sum_{j \in m} (f_{ij} \ z_m^*)/ \max(f_{ij}, z_m^*)}{(n-1)}} \tag{5.34}$$

5.5 Results and discussion

In this section, we evaluate the five meta-heuristic methods tested on all objectives separately as well as analyze them from a multi-objective perspective. We also compare the meta-heuristic performance against the optimal solutions as well as the best-performing heuristics from our previous work by [90]. Then, we summarize the discussion by providing a recommendation for the best-performing meta-heuristic based on our tests.

5.5.1 Single objective analysis

In this subsection, we analyze the solutions obtained from the meta-heuristic simulations with respect to each objective. At the end of each meta-heuristic simulation run, we obtain a set of points as the best-found solutions. For NSGA-II, we consider the first Pareto front at the end of the last generation. For NSGA-II with external population and MOEA/D, we consider the external population at the end of the last generation. We compare this best-found set of solutions to the ideal point - the optimal value for each objective obtained from our previous work in MILPs.

In Table 5.2, we report the average gap between the best values found for each objective and the ideal point (optimal value obtained from a MILP) across all 200 instances from every method. The Best Heuristic column shows the best heuristic results provided in [90]. But, we understand that comparing averages aggregated over a large number of instances is not ideal. Hence, we have broken down our analyses based on the size of the problems and conducted Kruskal-Wallis tests to test if there is a significant difference between the methods for each objective and problem sizes. We found that there was a significant difference among the methods for each configuration of the problem and for each objective (p-values < 0.05). Since this test does not provide information about which is the best or the worst performing method, we performed pairwise Kruskal-Wallis tests for each configuration of the problem and each objective. When a pairwise comparison is made, we can rank the methods based on whether there are significant differences or not in the pairwise tests. We report these results also in the following discussion. The configurations of the problems are based on how many machines (nmach) and how many jobs (norders) there were in the problems. We tested a total of 20 configurations.

Objective	Best Heuristic	N-CH	N-CH-EP	N-HO-EP	M-CH	M-HO
Makespan	0.17	0.11	0.11	**0.02**	0.08	0.05
Lateness	0.65	0.22	0.22	**0.20**	0.24	0.22
Setups	**0.05**	0.47	0.43	0.08	0.39	0.14
Color Preference	0.02	0.20	0.20	**0.01**	0.19	0.02
Shade Consistency	0.15	0.14	0.14	**0.05**	0.07	0.06

Table 5.2: Average minimum gap for each objective

For the makespan objective, the best performing meta-heuristic was NSGA-II with the hybrid optimal approach for job placement (N-HO-EP) as seen in Figure 5.4, where we present the gap between the best solutions from each method and the optimal solution. The gap is computed between the best-found solution by the meta-heuristic method and the optimal value for this objective and dataset. N-HO-EP was statistically different (p-value < 0.05) from the other methods in 15 of the 20 combinations tested, according to pairwise Kruskal Wallis tests, and its mean gap was smaller. This was closely followed by MOEA/D with the hybrid optimal approach for job placement (M-HO). NSGA-II using the hybrid optimal approach is only 2% worse than the optimal solution on average, as seen in Table 5.2, followed by 5% for M-HO. From Srinath et al. (2021), the best performing construction heuristic had an average gap of 17%, which is considerably worse than the average 2% gap obtained from using the best-found meta-heuristic method.

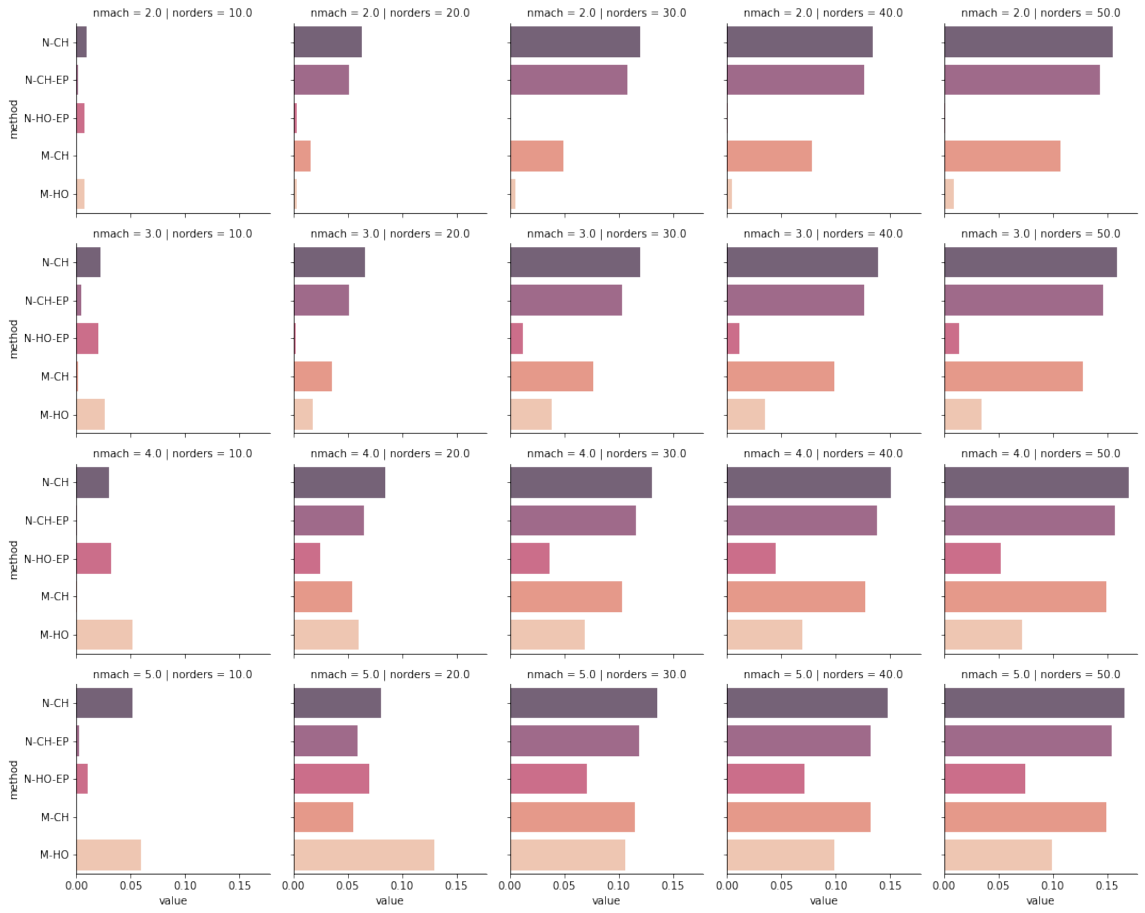

Figure 5.4: Makespan gap - results by number of machines and jobs

For the lateness objective, we see from Table 5.2 that the best performing construction heuristic from Srinath et al. (2021) is also 65% worse than the optimal solution on average. The usage of the meta-heuristic methods provides a vast improvement of about 45% compared to a construction heuristic. This improvement is also evident from Figure 5.5, where all the meta-heuristic methods perform only about 25% worse than the optimal solution. When compared to the optimal values, N-HO-EP is again the best performing meta-heuristic with an average gap of 20%. According to Kruskal Wallis tests, it is also better (p-value > 0.05) than the rest for 16 of the 20 combinations tested. However, the difference between N-HO-EP and the other meta-heuristic methods is small. Among all the objectives, the meta-heuristic methods performed the worst on this objective. Our previous work in MILPs indicates that this is a hard objective to solve in our problem. Considering that we also use a similar MILP in the meta-heuristic methods with the hybrid-optimal job placement approach, it is expected that this objective would be harder to solve than the others.

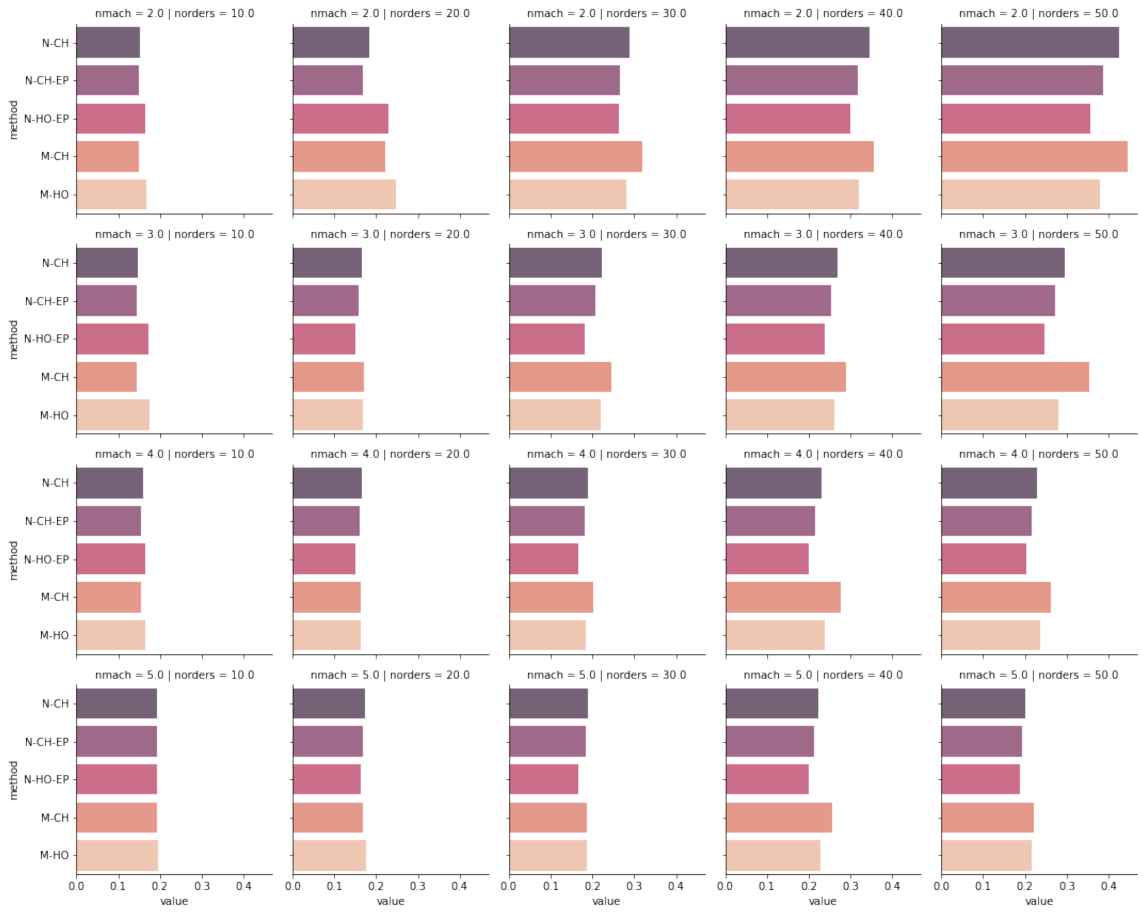

Figure 5.5: Lateness gap - results by number of machines and jobs

For the setup objective, the best performing construction heuristic is the PSR method described in our previous work [90]. The PSR method splits the jobs into groups of machines and then employs a MILP to solve the groups to optimality, providing the best solutions obtained. PSR provided an average gap of 2% as seen in Table 5.2. Even the best performing meta-heuristic (N-HO-EP once again) is slightly worse than PSR on this objective. Among the meta-heuristics, using the HO approach provides a significant boost on this objective, as seen in Figure 5.6, with N-HO-EP being able to find close-to-optimal solutions on all small-sized problems and on the larger problems also, the solutions from N-HO-EP and M-HO are better than the solutions obtained from the CH approaches. This can be attributed to the fact that the construction heuristic employed in these methods has no feature that strives to improve this objective. As part of future work in this direction, construction heuristics can be built differently for each objective, thus giving a chance for improvement.

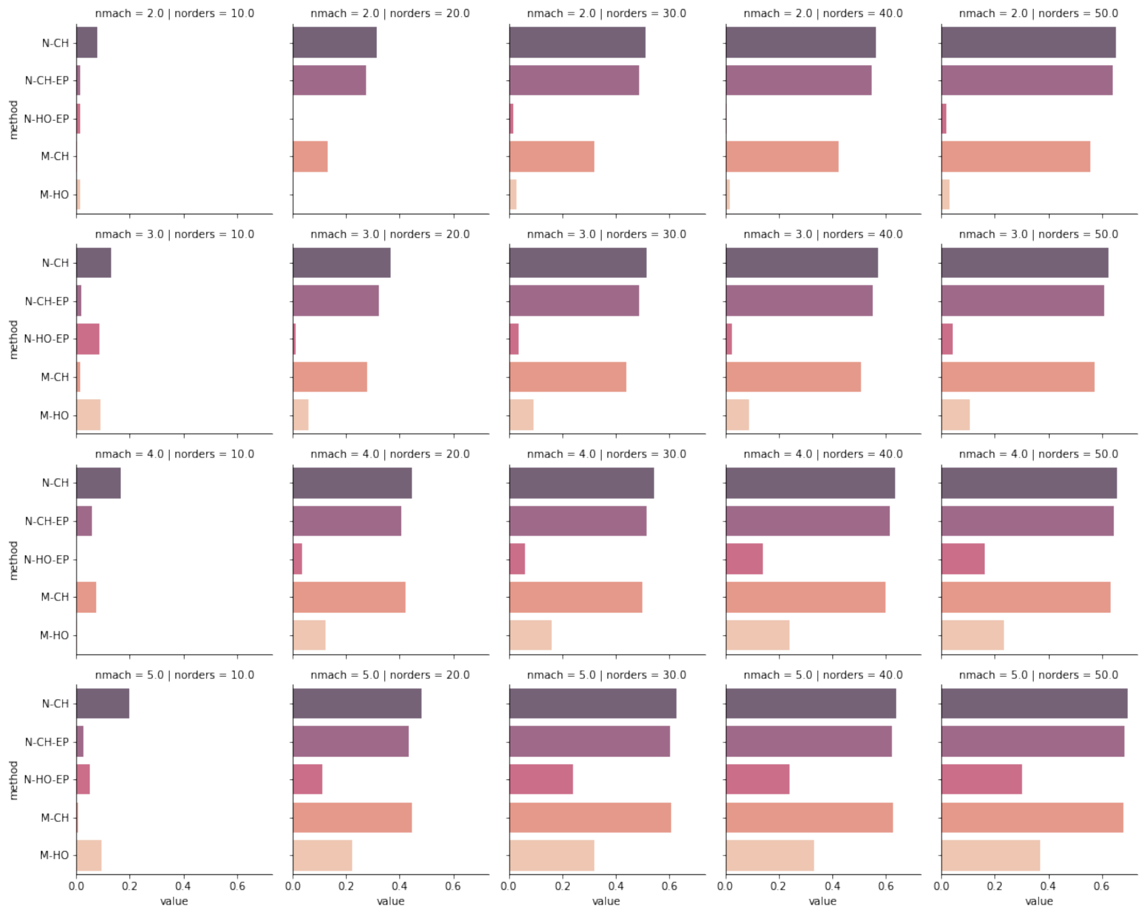

Figure 5.6: Setup times gap - results by number of machines and jobs

On the color preference objective, the N-HO-EP method almost solves all instances to optimality by the end of 100 iterations. This can be seen from the very small average gap across all instances in Figure 5.7. But, for this objective also, the PSR method provides very good solutions with an average gap of only 2% as seen in Table 5.2. Using the hybrid optimal approach works well for this objective too. Considering that this objective is very similar to the setup objective, a similar performance is expected. The meta-heuristic methods with the construction heuristic again perform worse on this objective. Again, this is because the method does not strive to improve this objective. Another factor that makes the MILP-based methods perform so well on the setup and color preference objectives is that the base MILP that partially solves these problems is easier to solve for these objectives than the lateness or makespan objectives.

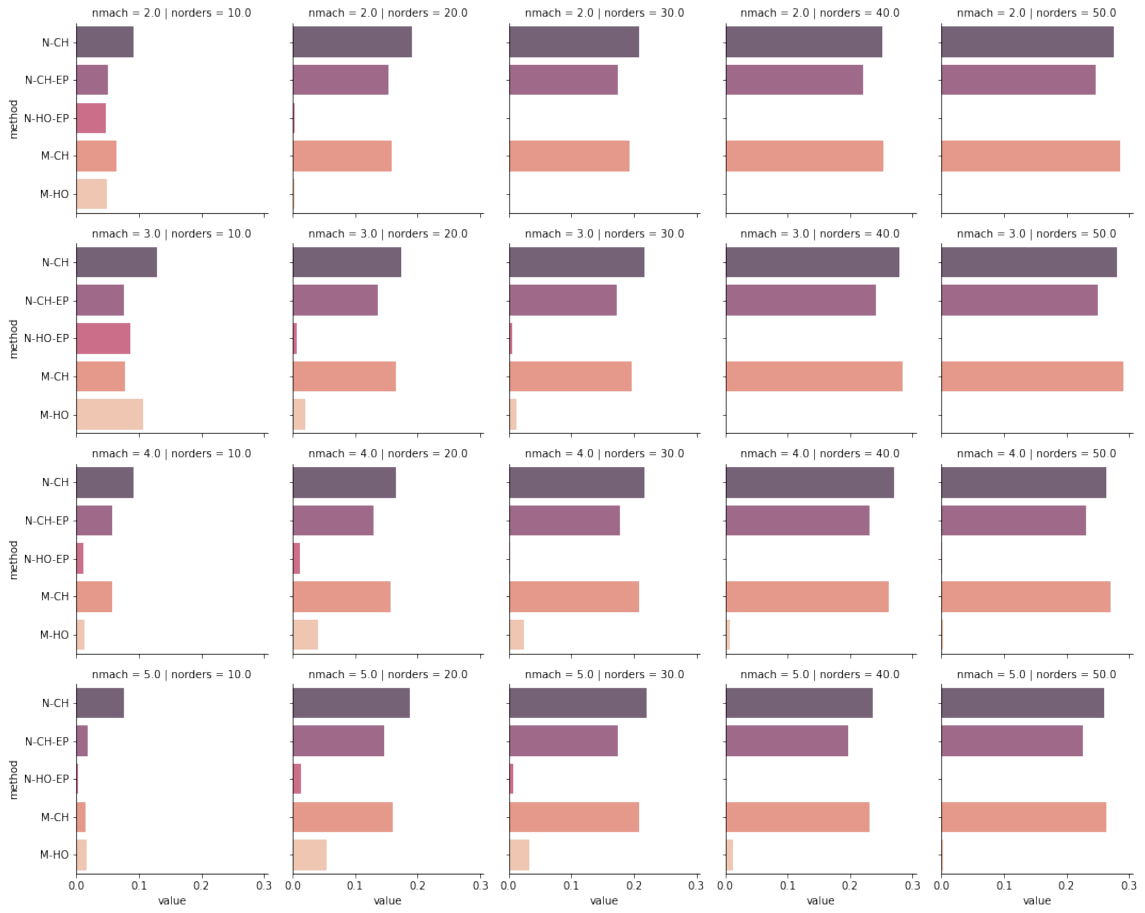

Figure 5.7: Color preference gap - results by number of machines and jobs

On the shade consistency objective also, N-HO-EP and M-HO perform better than the other methods, as seen in Figure 5.2, but the difference is small. N-CH-EP also performs almost as well as the HO methods. It is interesting to note that this objective seems to be harder to solve when there are fewer machines. This is because, with fewer machines, the transitions of shades will be drastic when the jobs are evenly spread. So, reaching a small value of less than 0.05 is difficult. Any value less than 0.1 is excellent on this objective from the perspective of the industrial sponsor. We also consider any value of 0.05 or lower as optimal for this objective for our tests. With this in mind, any of the methods can be used to obtain high-quality solutions for this objective. But, purely based on numbers, the best performing meta-heuristic is about 5% worse than optimal (Table 5.2. The best performing construction heuristic from [90] is about 15% worse than optimal. Thus, using the meta-heuristic provides an average improvement of 10%.

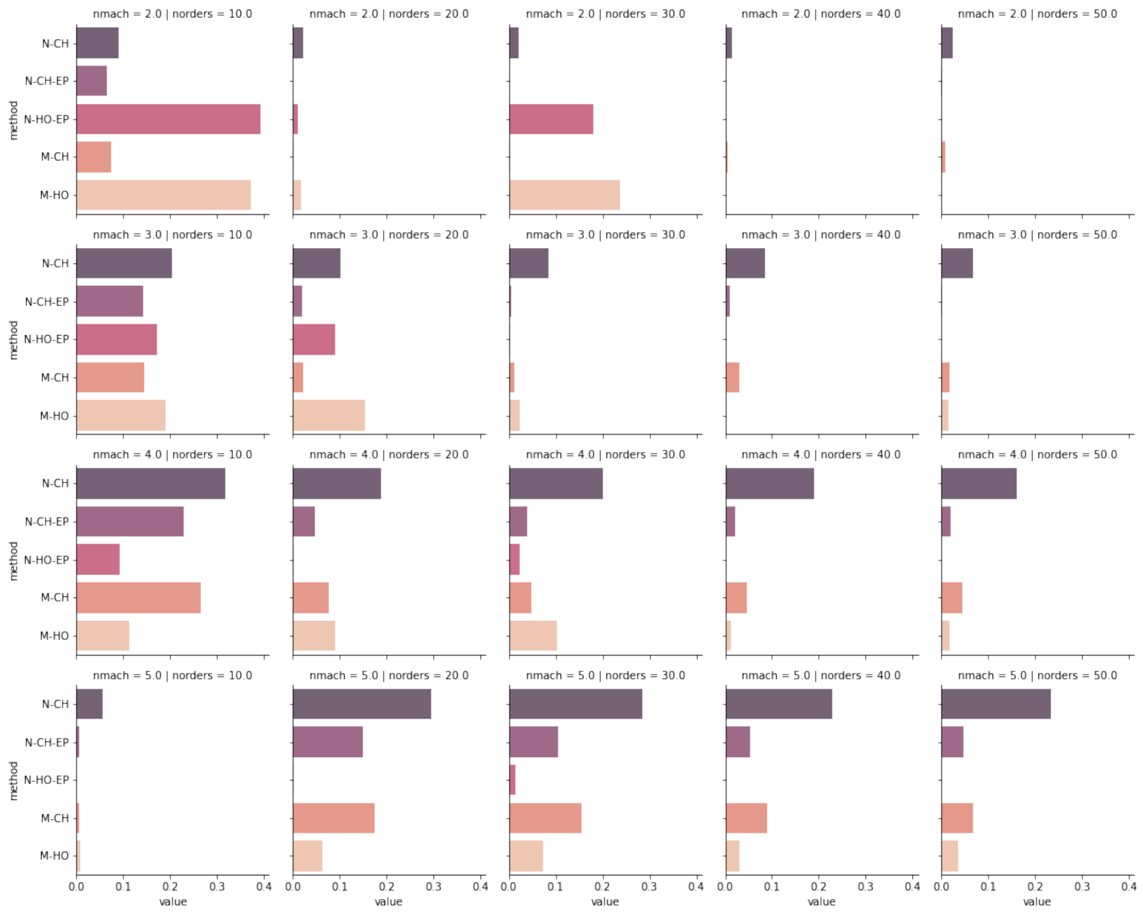

Figure 5.8: Shade consistency gap - results by number of machines and jobs

5.5.2 Multi-objective analysis

While it is important to analyze the solutions obtained from the different methods for each objective separately and understand how close they can reach the optimal value, it is also important to analyze the set of solutions from a multi-objective perspective. In Table 5.3, we summarize the metrics by averaging across all tested instances and runs. However, just as with the single objective analyses, we have also broken down our discussions based on problem configurations. For all multi-objective metrics also, we found a significant difference (p-value < 0.05) between the methods for each configuration and each metric from the Kruskal Wallis tests conducted on all methods together. We again did a pairwise Kruskal Wallis test to determine the better methods.

Measure	N-CH	NCH-EP	N-HO-EP	M-CH	M-HO
NNS	28.05	**122.73**	114.94	99.77	102.05
MIG	0.38	0.38	**0.37**	0.41	0.38
SNSG	0.04	0.04	0.04	0.04	0.04
Comptime	4665	4933	7683	**3064**	4725

Table 5.3: Average values of multi-objective metrics

The first metric we discuss is the number of non-dominated solutions provided by each method at the end of a simulation, in which more is better.

In Figure 5.9, we show the average number of non-dominated solutions provided by each method for each combination of jobs and machines. From the figure, we can see that N-CH-EP consistently generates more non-dominated solutions. It is closely followed by N-HO-EP and M-HO. When a Kruskal Wallis test was performed as a two-way comparison between each pair of methods, N-CH-EP was better (p-value¡0.05) than the rest in 8 of the 20 combinations. From Table 5.3, we see that N-CH-EP provides 123 solutions on average. N-HO-EP closely followed behind with 7 of the 20 combinations, followed by M-HO, which was better than the rest in 5 of the 20 combinations. Interestingly, when the number of machines is less, the construction heuristic-based approach performs better on this metric. When there are more machines, using the hybrid-optimal method generates more non-dominated solutions. Between NSGA-II with EP and MOEA-D, NSGA-II consistently generated slightly more solutions. The difference was not large, though. When an EP was not used, NSGA-II was worse than the other methods in all scenarios. This is because of the limit enforced by the population size of each iteration. In our case, we had a limit of 50 on the parent population for each iteration, so there could at most be 50 non-dominated solutions.

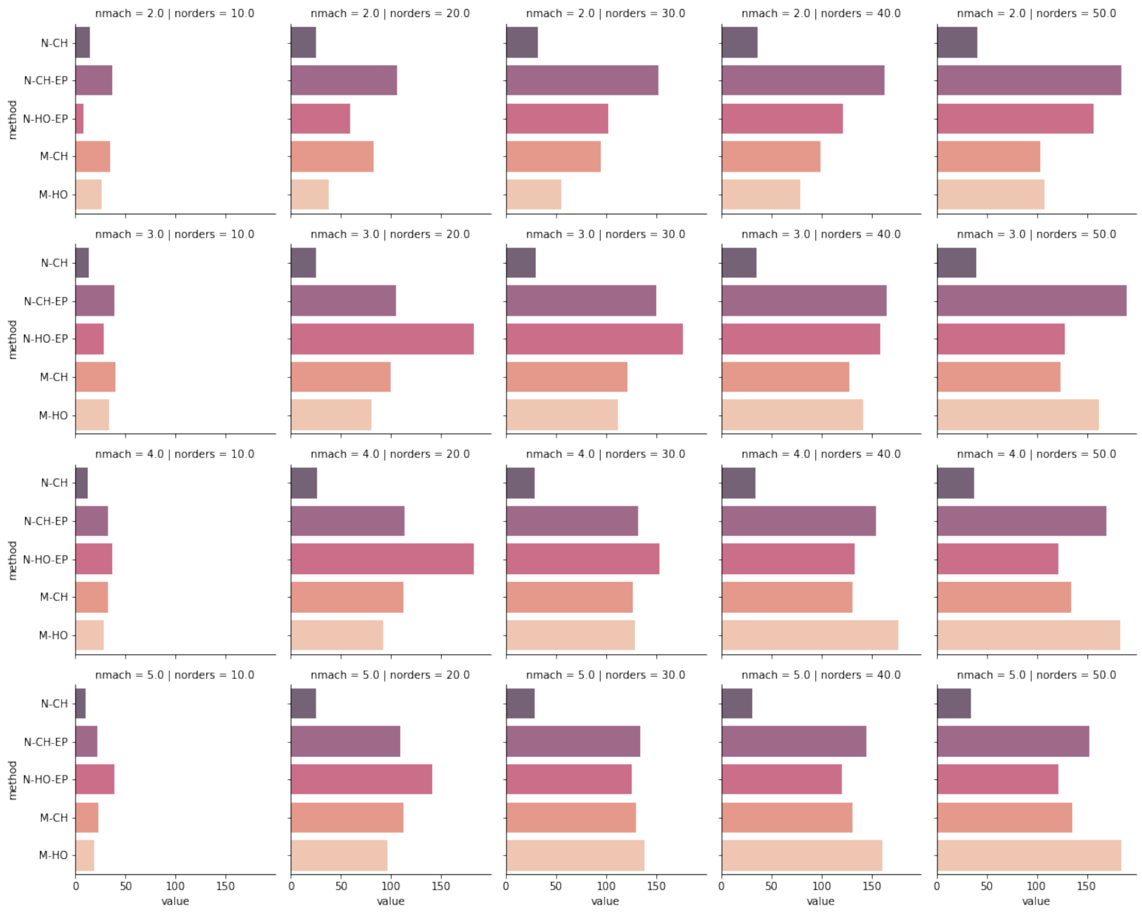

Figure 5.9: NNS - results by number of machines and jobs

The next metrics we discuss are the mean ideal gap (MIG) and the mean ideal distance (MID) metrics. MID is the standard metric used, but, in our case, the scales of the objectives are different, and this difference can result in skewed results on the MID metric. To negate the effect of the scales, we use the MIG metric, which is based on the gap rather than the distance. The mean ideal gap evaluates how far each solution is from the ideal point and averages the distances of all solutions. For this metric, a lower value means closer the mean gap between the solutions and the ideal point (obtained from solving each objective to optimality), which is better.

For the MIG metric, from Table 5.3, we see that there is no significant difference between the averages. When the results are analyzed for each size of problem also, we see a very similar performance across all the methods tested, as seen in Figure 5.10. However, there is a slight advantage to using the hybrid optimal approaches for larger problems, as evidenced by the KW tests. N-HO-EP and M-HO both were better than the rest of the methods for 14 of the 20 combinations tested, and they were better than the rest for all problems with greater than 3 machines and greater than 30 jobs. N-CH-EP was better than the other methods when the problem sizes were small. But, in many cases, the other methods were equally good.

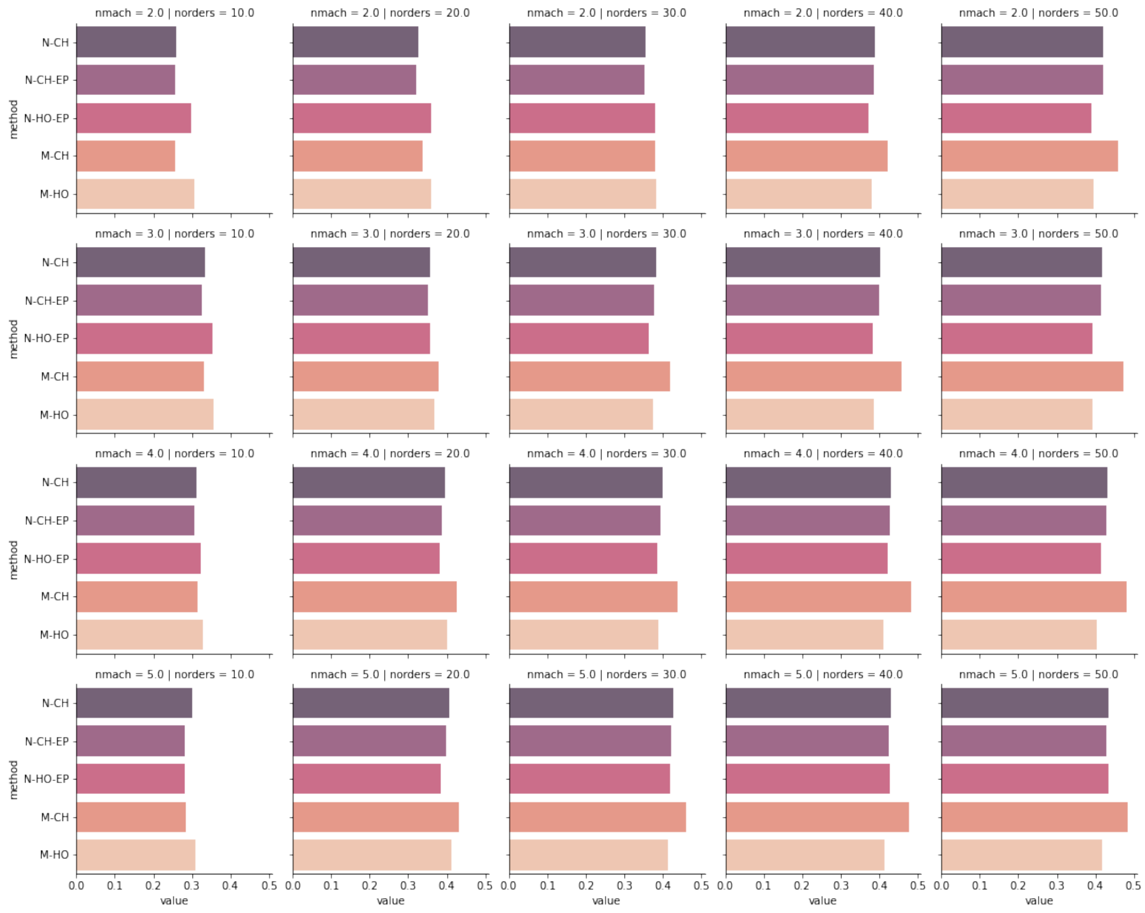

Figure 5.10: MIG - results by number of machines and jobs

This is a strange conclusion, considering that the performance of the HO-based methods relative to the construction heuristics-based methods perform better on all objectives than the construction heuristic-based approaches. However, this can be explained. When we consider multiple objectives, usually, these objectives conflict with each other. Consider a point that is optimal for one of the objectives, say makespan; With this sample point that is optimal for makespan, its performance on another objective, such as the color preference, might be terrible. This might make the average gap for this point from the ideal point large because it is far from optimal for the other objectives. This can explain why, even though the HO methods perform better than the CH methods for each objective, the

MIG values for all methods are similar.

When we evaluate the set of solutions based on their spread, we turn to the SNS and SNSG metrics. The SNS metrics also suffer from different scales on the solutions. When we use the gaps of solutions from the ideal point and observe the standard deviation of the gaps, i.e., using the SNSG metrics, the results are once again very similar, as seen in Figure 5.11. This is because the gaps range from 0 to 1. The small scale of possible values makes the differences between the methods very small. However, for this metric too, N-CH-EP, M-HO, and N-HO-EP are the slightly better performing metrics. Once again, the hybrid-optimal approaches (especially M-HO) perform well on the larger problems with 4 or more machines. N-CH-EP performs well on the smaller problems.

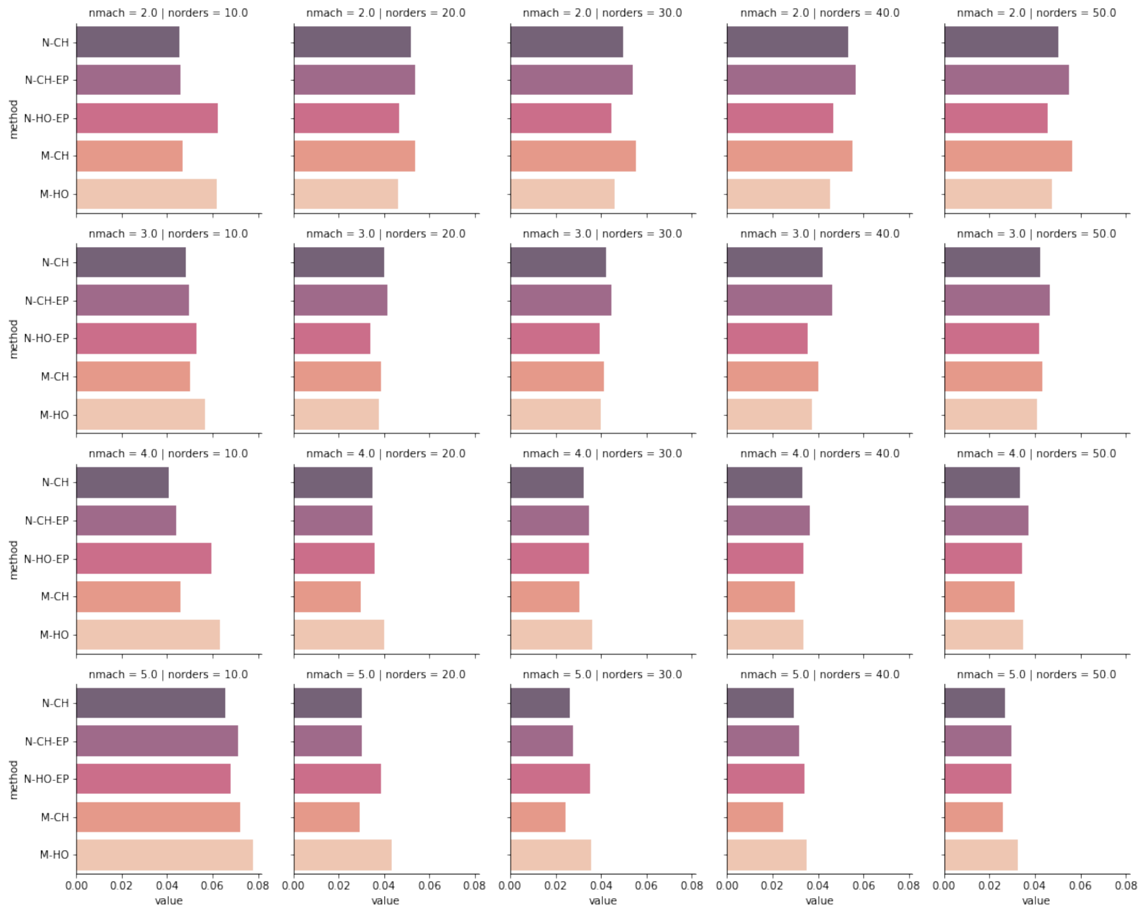

Figure 5.11: SNSG - results by number of machines and jobs

The CSS metric provides a reliable pair-wise comparison among the MOEAs. When N-CH and N-CH-EP are compared, we see that N-CH is completely weakly dominated by N-CH-EP, as seen in Figure 5.12. This is expected because all the solutions of N-CH are in the larger N-CH-EP solution set. Across all instances, the average NNS value for N-CH was about 22% of the value for N-CH-EP (Table 5.3). On the CSS metric between these two methods, N-CH weakly dominated N-CH-EP 19% of the time on average (Table 5.4). Since N-CH-EP contains all solutions of N-CH, we will continue the comparison between the methods by using only N-CH-EP.

Method	N-CH	N-CH-EP	N-HO-EP	M-CH	M-HO
N-CH	–	1.00	0.29	0.16	0.19
N-CH-EP	0.19	–	0.27	0.15	0.18
N-HO-EP	0.09	0.16	–	0.13	0.31
M-CH	0.36	0.57	0.55	–	0.46
M-HO	0.11	0.19	0.63	0.16	–

Table 5.4: Average CSS values

From Figure 5.13, we see N-CH-EP clearly being better than M-CH on 19 of the 20 combinations tested. The difference is more significant when the problems are larger also, with almost no solution of N-CH-EP being weakly dominated by M-CH for the largest problems with 4 or 5 machines. N-CH also performs better than M-CH, with only 16% of its solutions weakly dominated, while 36% of M-CH solutions are weakly dominated (Table 5.4). N-CH-EP is the best CH-based metric; however, it is worse than the HO-based methods.

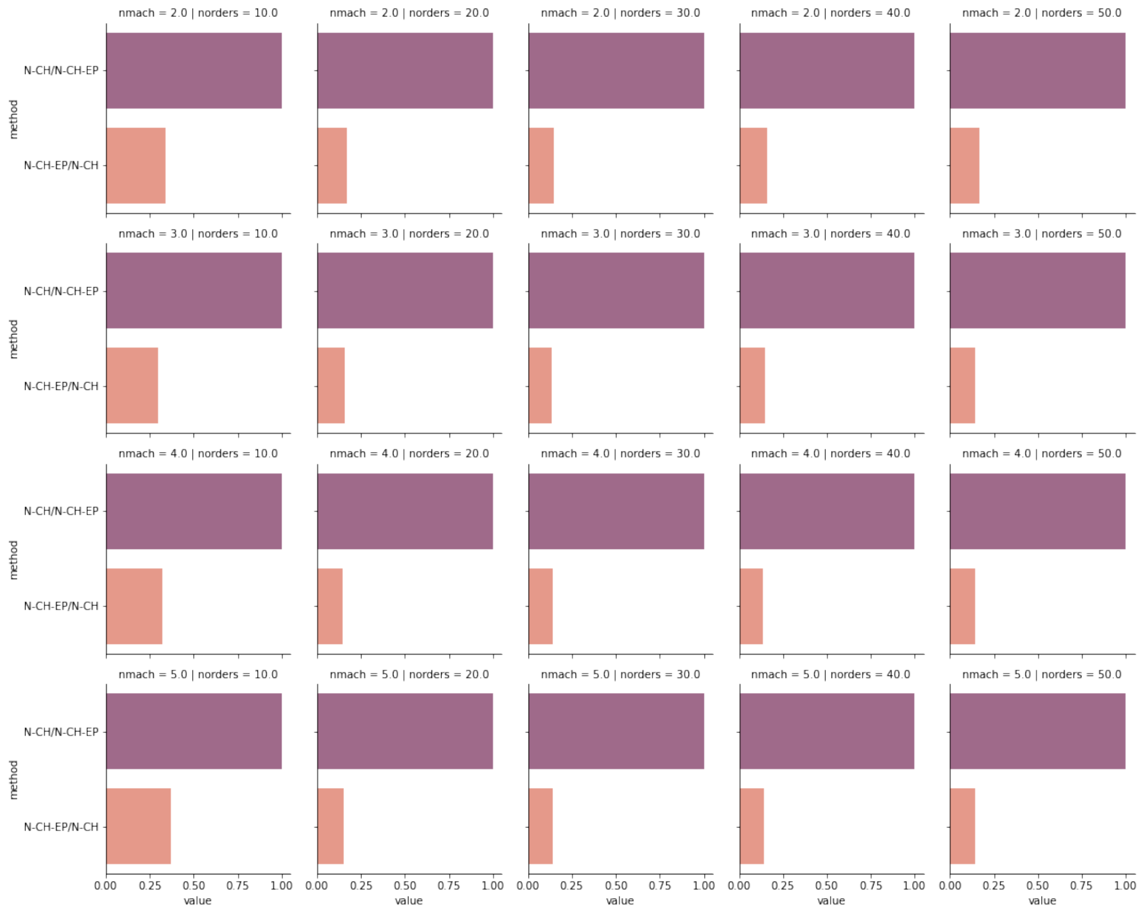

Figure 5.12: CSS - N-CH / N-CH-EP - results by number of machines and jobs

In Figure 5.14, we see the advantage of using the hybrid optimal approach, especially for large problems. In this figure, we compare the CSS metric between N-CH-EP and N-HO-EP. N-HO-EP performs better for all problems with more than 20 jobs. But, the construction heuristic can find good solutions quickly for the small problems with fewer solutions.

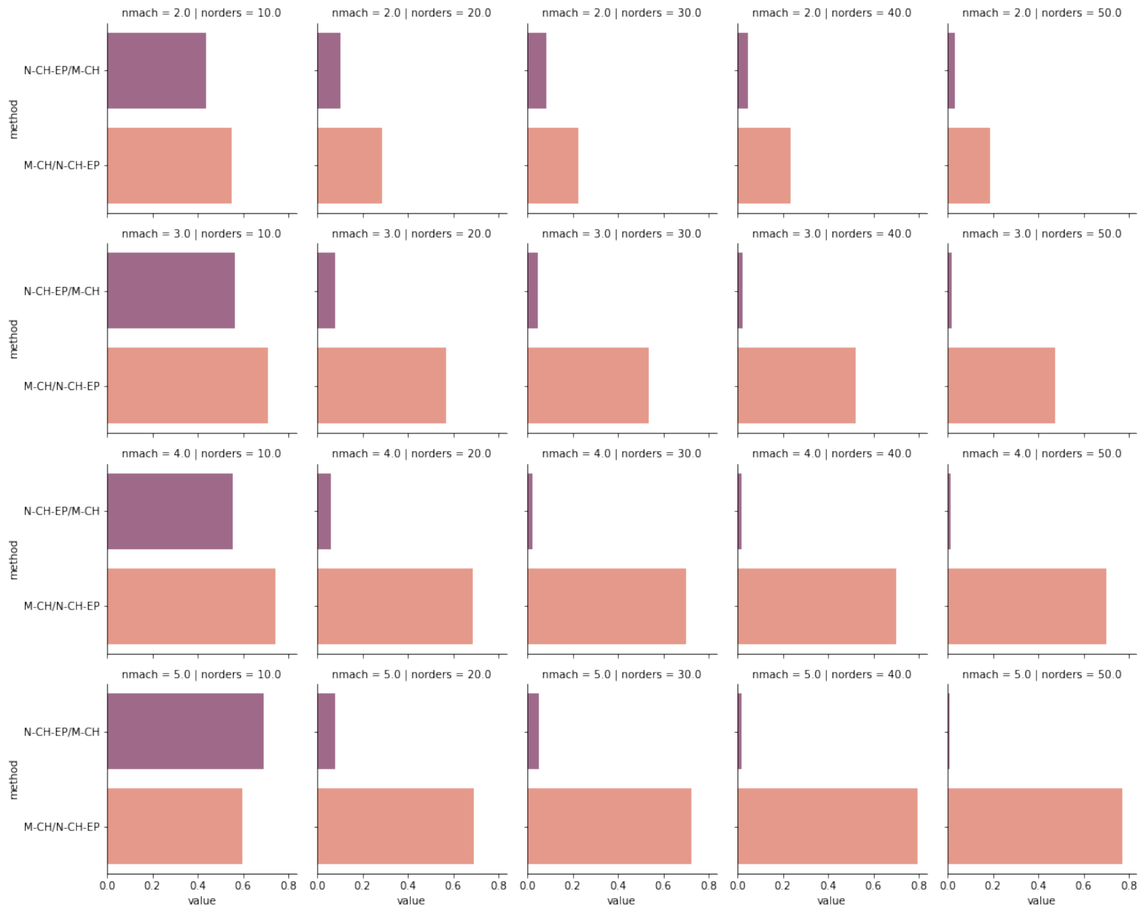

Figure 5.13: CSS - N-CH-EP / M-CH - results by number of machines and jobs

The comparison between N-CH-EP and M-HO is also similar, with M-HO performing better than N-CH-EP on all problems with more than 20 jobs. This can be seen in Figure 5.15.

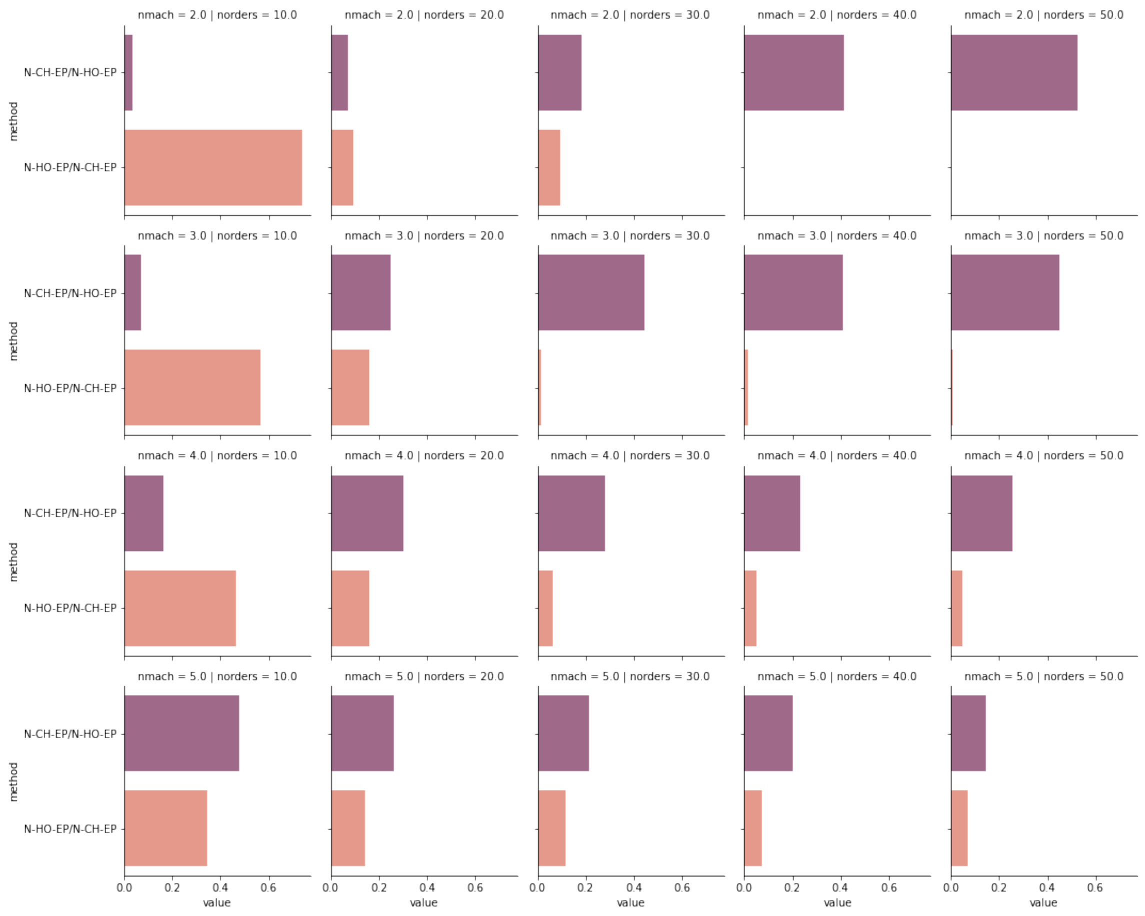

Figure 5.14: CSS - N-CH-EP / N-HO-EP - results by number of machines and jobs

From the tests we conducted for our problem, the usage of the NSGA-II framework proved to be better than the MOEA/D framework to obtain higher-quality solutions. This is evident from the CSS values for N-CH and N-CH-EP being better than M-CH (Table 5.4). This is also seen in comparing the two frameworks when using the hybrid optimal approach. In Figure 5.16, we compare N-HO-EP and M-HO. We see N-HO-EP perform better than M-HO on 16 of the 20 combinations and slightly worse than M-HO on the 4 other combinations. It is interesting to note that M-HO performs somewhat better on the largest problems with over 4 machines and 40 jobs.

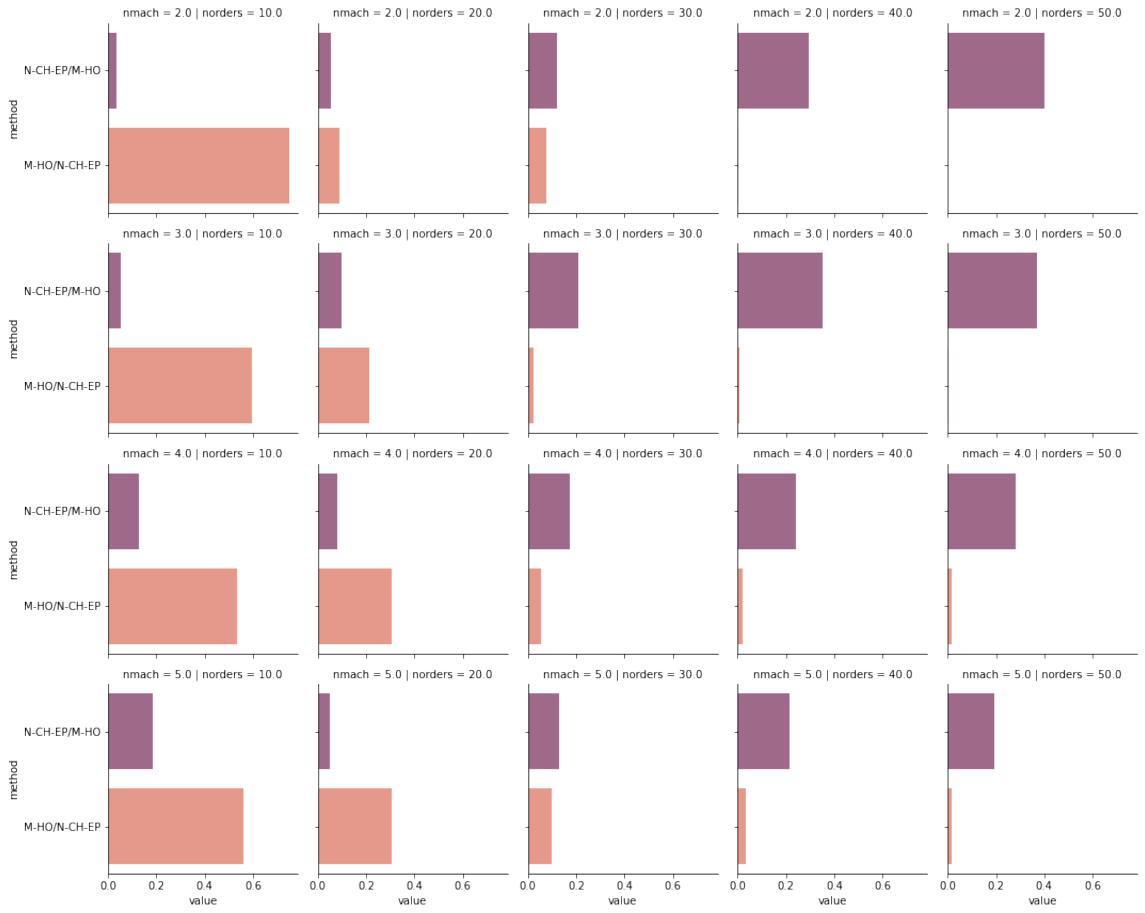

Figure 5.15: CSS - N-CH-EP / M-HO - results by number of machines and jobs

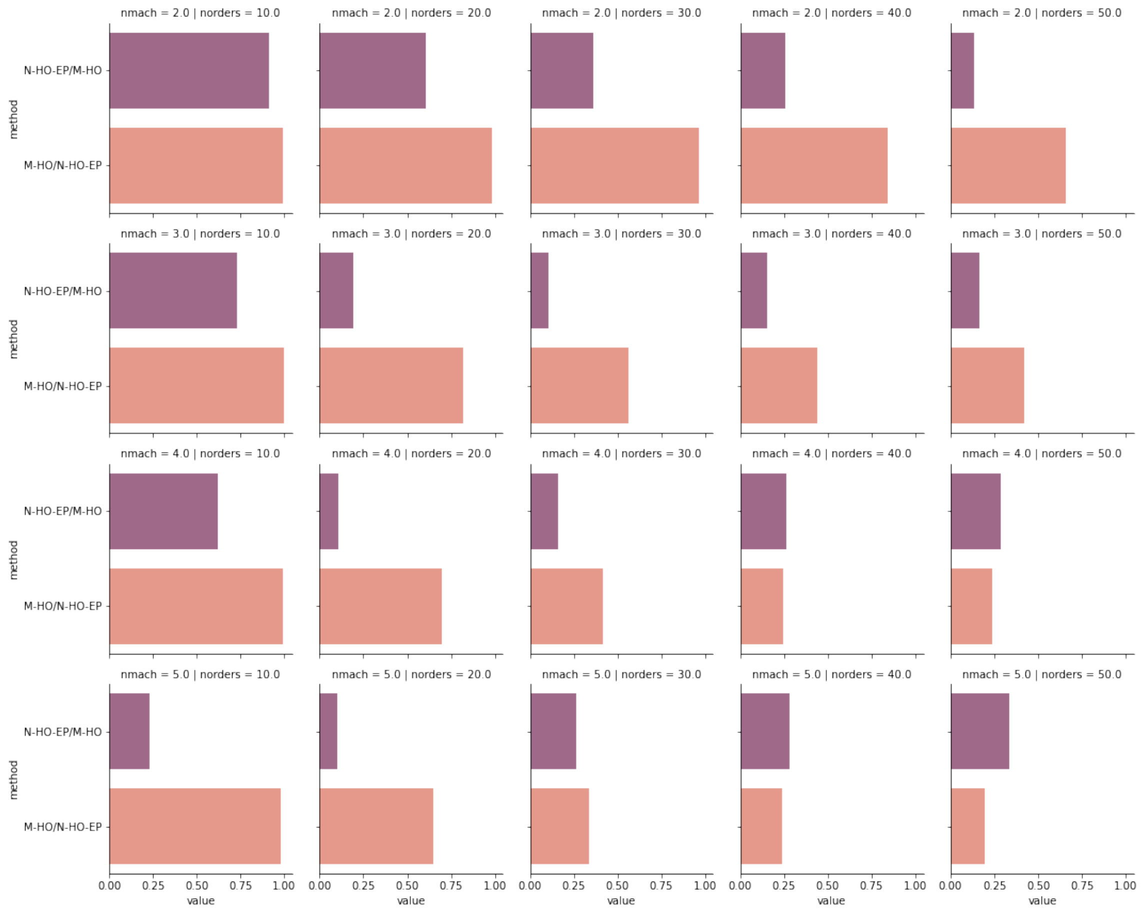

Figure 5.16: CSS - N-HO-EP / M-HO - results by number of machines and jobs

Overall, our tests conclude that N-HO-EP is the best performing method, on average, with its solutions dominating solutions from other methods more often than vice-versa. N-HO-EP also performed better than the rest on all the objectives when evaluated individually. If the methods were ranked based on averages of CSS values, then N-HO-EP $>$ M-HO $=$ N-CH-EP $>$ N-CH $>$ M-CH. While N-H0-EP is appealing for its overall solution quality, from Figure 5.17, we see that it suffers from the worst computation times. From Table 5.3, we see that N-HO-EP takes on average 2.12 hours to compute a single instance of a problem for 100 iterations. The difference is larger when the problems are larger, with N-HO-EP taking considerably longer than the other methods. The NSGA-II framework, in general, takes longer to solve than the MOEA/D framework. The usage of the hybrid optimal approach also hampers the computation times when compared to using the construction approach. Across the tests conducted, using the HO method took 1.5 times longer than the CH approach, with all other factors the same. The main factor that increased the computation time was the usage of the MILP to solve the lateness objective. Even though we had a time limit set of 10s, the other objectives could be solved in a fraction of a second, whereas many instances with the lateness objective took the full 10s without solving to optimality, and this increased the computation time. Improving the MILP on any objective (especially lateness) helps considerably improve the solution time when the HO method is used for any meta-heuristic framework. The MOEA/D framework is significantly faster than the NSGA-II framework. This is because, during every iteration, NSGA-II finds all the non-dominated solutions. Then among the rest, it again finds the non-dominated solutions and so on. However, in the MOEA/D framework, every generated offspring is compared to the EP and added or removed from consideration. Also, for each Pareto front, time is spent in the NSGA-II framework to find the crowding distance for each solution. This difference in the evaluation of solution quality affects the computation times. However, since NSGA-II also finds all non-dominated solutions and employs an elitist approach, it usually finds better solutions. The crowding distance operator helps obtain a better spread of solutions.

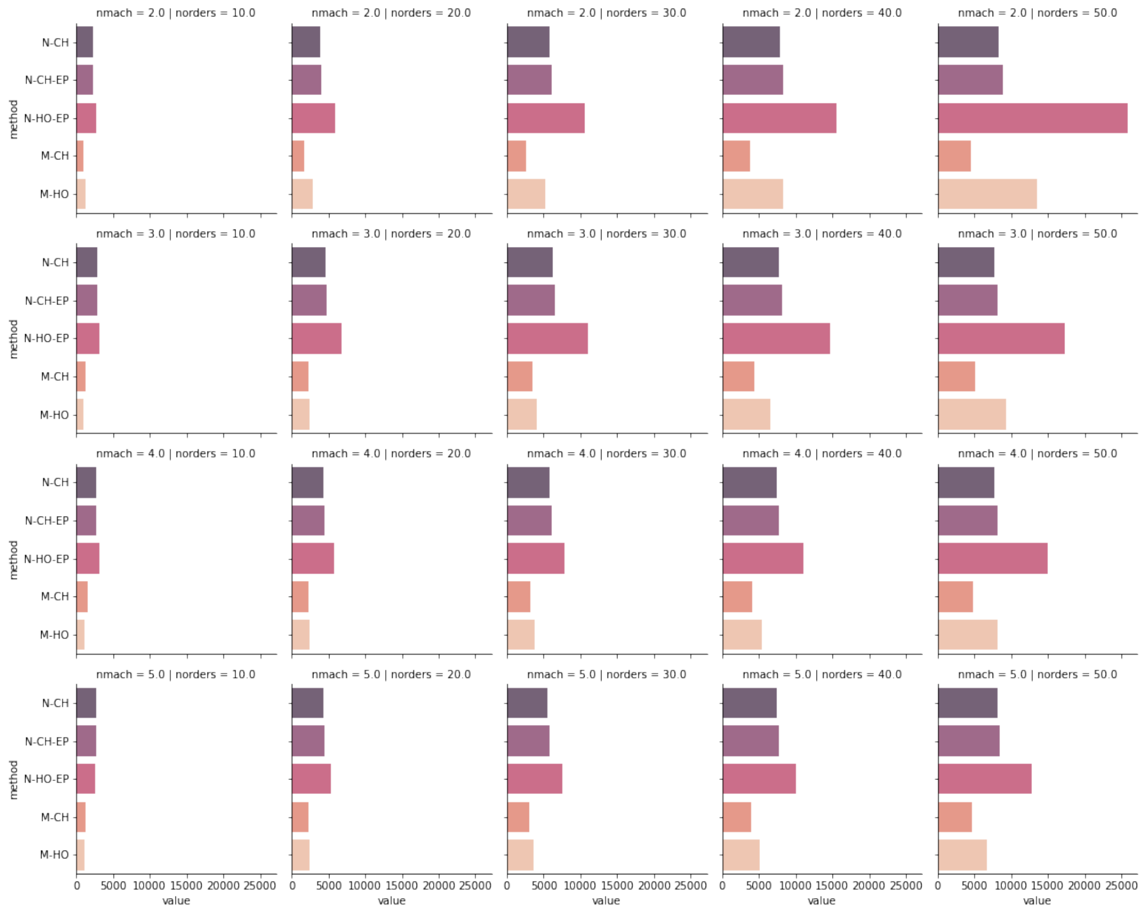

Figure 5.17: Computation time - results by number of machines and jobs

Chapter 6

Conclusion and Future Work

In this dissertation, we have we have obtained a multi-objective scheduling problem from an industrial sponsor. This problem consists of parallel machines, machine eligibility, sequence-dependent setup times and sequence-dependent constraints. The objectives important to this problem are to reduce the makespan, lateness and setup times (specifically). Also, to improve the quality of production in a fabric dying industry, we have introduced two new objectives: color preference to provide smooth transitions between color changes, and shade consistency to ensure smooth shade transitions and similar shades of colors running on the machines.

First, we present an MILP formulation to solve the problem. We thoroughly tested it for different sizes of problems, for all the objectives separately. We evaluated the performance of the base formulation on multiple metrics such as the optimality gap, percentage of instances solved with small gaps, number of nodes explored, time taken etc. Then we observed the root relaxation solutions, and the relaxation solutions at some of the nodes in the branch and bound tree, and found some violated valid inequalities. Then we added these inequalities in phases. At each phase, we evaluated the performance of the formulation on all the metrics. Then, we developed separation techniques to identify the a few more violated inequalities at each node, and added them as cuts using callback functions. These callback functions, we implemented on all the versions (phases) of the formulation.

Overall, we did not find the addition of the sequence dependent inequalities to drastically change the performance of any of the formulations for any of the objectives. Next, we found that the base formulation was inadequate in solving the makespan objective. Its performance on the makespan objective was significantly worse than its performance on the other objectives. For the makespan objectives, we found the usage of the w variables on top of the base variables very useful in providing additional valid inequalities. The gap went down from 74% to 2%, and most instances were solved to optimality or with small gaps of less than 10%. This version was also useful in cutting down the solution time. For the lateness objective, even though we found that the solution quality was good (about 12%), the formulation was unable to verify its quality by providing better lower bounds and closing down the gaps. Interestingly, the inequalities we added in v1m (chapter 3 helped provide those better bounds. However, this version was unable to find feasible solutions which beat the solutions provided by the initial formulation. For the setup objective, even though the base formulation was able to solve the instances to optimality, we found a better formulation which solved the same instances faster and explored fewer nodes in the tree. The best version for this objective as well as the color preference objective was v3 from chapter 3. For the shade consistency objective, the initial formulation was the best performer in terms of all the metrics.

So, in conclusion, we were able to greatly improve the performance of the exact methods on the makespan objective. We were able to solve the setup time and color preference objectives faster. We were able to provide better bounds for the lateness objective.

The improved methods presented are not only more useful to solve real-world problems, but also provides better ways to evaluate the performance of any heuristic or meta-heuristic methods. Also, using these methods, it is possible to develop better performing MILP-based heuristics and meta-heuristics.

With the improved exact methods, we were able to solve medium sized problems but for large sized problems, the curse of dimensionality still persisted. So, we adapted/developed multiple construction heuristics to generate schedules for large sets of data quickly. We adapted simple dispatching rules like SPT, EDD etc that are ubiquitous in literature, and developed slightly complicated insertion based (H2, H3) and bin-packing based (H4) heuristics. These heuristics gave acceptable results for our industrial sponsor on all the objectives but there was still scope for improvement. SO, as a research exercise, in order to further improve the quality of solutions, we have developed an MILP-based heuristic (PSR) which first breaks down a large problem into smaller easily solvable problems and solves each broken-down problem using the MILP in a reasonable amount of time. The method which breaks down the larger problem aims to minimize the information loss while splitting the problem. This splitting method is also a graph partitioning MILP which is also an NP-Hard problem. But, for this problem, breaking down the larger problems does not take a significant amount of time. Of course, it cannot solve all sizes of problems. We have shown that for problems with greater than 400 nodes, the splitting method does suffer from the curse of dimensionality.

The methods were then subject to rigorous testing to compare them against each other as well as to compare them to optimal results. The testing was split into two stages: first, the methods were compared against optimal results for small-sized problems. Then, for larger problems, the methods were compared among themselves.

For the makespan objective, PSR (chapter 4)was competitive but certainly not the best. The best methods for the makespan objective were H2, H3 (insertion based methods), LPT and LFJ. PSR struggled at this objective due to some cases where the splitting did not provide an even balance of machines and jobs. H4 was the worst on this objective.

For the lateness objective, EDD was the best but even that was significantly worse than optimal methods. PSR was second-best. The other methods were all slightly worse. However, EDD was much worse than the other methods on the other objectives.

On the other three objectives, the usage of index based insertion methods and bin-packing based methods truly paid off. PSR was also a big winner on these objectives, especially for larger problems. PSR, H2, H3 and H4 outperformed the rest on these objectives. On each of these three objectives, PSR was the best followed by H4. H4 was closely followed by H3 and then H2.

So, if a recommendation has to be made with respect an objective, we would choose H3 if makespan is important, considering the benefits it provides on other objectives. If lateness is the primary objective, EDD is definitely the way to go. However, the losses on other objectives has to be kept in mind. When it comes to the other three objectives, H4 is the best method. However, if MILP-based methods are an option and given a little more time, PSR performs consistently well and is the overall best method. It is competitive on the makespan-objective for most cases. It is only second-best on the lateness objective and significantly better than the rest of the methods on the other three objectives.

While we have used index based methods to capture the multi-objective nature of the problem, we have not tackled this problem in a truly multi-objective sense. Also, with five objectives, it is difficult to obtain the pareto-optimal solutions using weighted-sum or epsilon constraint methods in MILPs. So, to tackle this problem in a multi-objective sense, we have adapted and developed improved multi-objective meta-heuristic methods to obtain solutions that perform well on all objectives at the same time, if not optimal for any single objective.

In this dissertation, we have used two multi-objective evolutionary algorithmic (MOEA) frameworks to solve our problem. We have proposed two approaches, construction heuristic and hybrid-optimal heuristic, to create the initial population and also to generate feasible offsprings. A special chromosome design is presented for each approach and the heuristics are adapted within the NSGA-II and MOEA/D frameworks. We have proposed an MILP to solve each single machine sequencing subproblem within the HO method (5). An advantage of this method is that the objective used to evaluate each chromosome is encoded in a gene. HO approach decodes this gene and solves the MILP with the corresponding objective. This procedure helps the MOEAs to explore a better set of representative Pareto solutions.

In our problem, there are five objectives. The scale of the objective values for each objective varies. This scale difference creates an unequal environment for the objectives.

The NSGA-II framework does not require the scales of the objectives to be similar, but for MOEA/D, it is important to address this issue. So, we have applied normalization using the Tchebycheff approach in the MOEA/D framework to evaluate the objectives equally. While evaluating the different methods from a multi-objective perspective, we present normalized performance metrics that do not suffer from existence of different objective value scales.

We have used two MOEA frameworks, NSGA-II and MOEAD, and two feasible schedule generation methods, CH and HO. This resulted in four combinations; N-CH-EP, N-HO-EP, M-CH and M-HO. Additionally, we have added NSGA-II without external population (N-CH) to the comparison in order to show the positive impact of keeping all of the non-dominated solutions in the external population during the algorithm. The comparison has been made among five combinations as well as the ideal points of each objective provided by MILP and the best solution found by the heuristics given in chapter 4. The numerical results show that NSGA-II framework outperforms the MOEA/D framework at the cost of more computational time. Also, both MOEA frameworks provide higher quality solutions when the HO method is integrated but again at the cost of more computational time. In conclusion, when time is a major factor, it is better to use the MOEA/D framework. Even using the HO approach with the MOEA/D framework does not affect the computation time greatly, but it helps in providing significantly better solutions on all objectives. However, if time is available, using the NSGA-II framework, with the external population stored, and using the HO approach provides high quality solutions. While it does not consistently provide optimal solutions for the lateness objective, it is better than all the other methods on lateness too.

As part of future work, we think that there is scope for improvement on the lateness objective, in both being able to find better solutions as well as to find better lower bounds, especially when the number of jobs per machine is large. An improvement on the lateness objective not only makes the MILP more usable, but also for larger problems, it makes the best meta-heuristics and MILP-based heuristics perform better and faster. Also, the construction heuristics employed within the MOEA frameworks can be tailored for each objective so that the overall performance improves, without the need to solve MILPs. Finally, a major difficulty in evaluating and improving the performance of the multi-objective methods was that we had five objectives. With that many objectives, it is hard for the methods to improve all of them at the same time. As part of future work, it might be a worthwhile to explore ways to reduce objectives, while not hampering the performance on any single objective.

Bibliography

[1] Hamid Abedinnia, Christoph H Glock, Eric H Grosse, and Michael Schneider. Machine scheduling problems in production: A tertiary study. *Computers & Industrial Engineering*, 111:403–416, 2017.

[2] L. R. Abreu and B. A. Prata. A hybrid genetic algorithm for solving the unrelated parallel machine scheduling problem with sequence dependent setup times. *IEEE Latin America Transactions*, 16(6):1715–1722, June 2018.

[3] Mojtaba Afzalirad and Javad Rezaeian. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering*, 98:40–52, 2016.

[4] Derya Eren Akyol and G Mirac Bayhan. Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach. *The International Journal of Advanced Manufacturing Technology*, 37(5-6):576–588, 2008.

[5] Ali Ala, Fawaz E Alsaadi, Mohsen Ahmadi, and Seyedali Mirjalili. Optimization of an appointment scheduling problem for healthcare systems based on the quality of fairness service using whale optimization algorithm and nsga-ii. *Scientific Reports*, 11(1):1–19, 2021.

[6] Mohammad Alaghebandha, Vahid Hajipour, and Mojtaba Hemmati. Optimizing multi-objective sequencing problem in mixed-model assembly line on just-in-time: particle swarm optimization algorithm. *International Journal of Management Science and Engineering Management*, 12(4):288–298, 2017.

[7] Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, 2015.

[8] Bradley E Anderson, James D Blocher, Kurt M Bretthauer, and Munirpallam A Venkataramanan. An efficient network-based formulation for sequence dependent setup scheduling on parallel identical machines. *Mathematical and Computer Modelling*, 57(3-4):483–493, 2013.

[9] Jaime A Arango, Jaime A Giraldo, and Omar D Castrillon. Scheduling of non-related parallel machines with sequence dependent setup times and dynamic entry using genetic algorithms. *Información Tecnológica*, 24:73 – 84, 2013.

[10] Jean-Paul Arnaout. A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Annals of Operations Research*, 285(1-2):273–293, 2020.

[11] Jean-Paul Arnaout, Rami Musa, and Ghaith Rabadi. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines—part ii: enhancements and experimentations. *Journal of Intelligent Manufacturing*, 25(1):43–53, 2014.

[12] Oliver Avalos-Rosales, Francisco Angel-Bello, and Ada Alvarez. Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76(9-12):1705–1718, 2015.

[13] M Aramon Bajestani and Reza Tavakkoli-Moghaddam. A new branch-and-bound algorithm for the unrelated parallel machine scheduling problem with sequence-dependent setup times. *IFAC Proceedings Volumes*, 42(4):792–797, 2009.

[14] Kenneth R Baker and JWM Bertrand. A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management*, 3(1):37–42, 1982.

[15] Natalia P Basán, Mariana E Cóccola, Alejandro García del Valle, and Carlos A Méndez. Scheduling of flexible manufacturing plants with redesign options: A milp-based decomposition algorithm and case studies. *Computers & Chemical Engineering*, 136:106777, 2020.

[16] J Behnamian, M Zandieh, and SMT Fatemi Ghomi. Due window scheduling with sequence-dependent setup on parallel machines using three hybrid metaheuristic algorithms. *The International Journal of Advanced Manufacturing Technology*, 44(7-8):795–808, 2009.

[17] J Behnamian, M Zandieh, and SMT Fatemi Ghomi. Due windows group scheduling using an effective hybrid optimization approach. *The International Journal of Advanced Manufacturing Technology*, 46(5-8):721–735, 2010.

[18] Javad Behnamian, Mostafa Zandieh, and SMT Fatemi Ghomi. Parallel-machine scheduling problems with sequence-dependent setup times using an aco, sa and vns hybrid algorithm. *Expert Systems with Applications*, 36(6):9637–9644, 2009.

[19] Robert L Burdett, Paul Corry, Colin Eustace, and Simon Smith. Scheduling preemptible tasks with flexible resourcing options and auxiliary resource requirements. *Computers & Industrial Engineering*, 151:106939, 2021.

[20] Robert L Burdett, Paul Corry, Prasad KDV Yarlagadda, Colin Eustace, and Simon Smith. A flexible job shop scheduling approach with operators for coal export terminals. *Computers & Operations Research*, 104:15–36, 2019.

[21] Robert L Burdett and Erhan Kozan. An integrated approach for scheduling health care activities in a hospital. *European Journal of Operational Research*, 264(2):756–773, 2018.

[22] Ann Melissa Campbell and Martin Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation science*, 38(3):369–378, 2004.

[23] Erdal Caniyilmaz, Betül Benli, and Mehmet S Ilkay. An artificial bee colony algorithm approach for unrelated parallel machine scheduling with processing set restrictions, job sequence-dependent setup times, and due date. *The International Journal of Advanced Manufacturing Technology*, 77(9-12):2105–2115, 2015.

[24] Donald C Carroll. *Heuristic sequencing of single and multiple component jobs.* PhD thesis, Massachusetts Institute of Technology, 1965.

[25] Pei-Chann Chang and Shih-Hsin Chen. Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft Computing*, 11(1):1263–1274, 2011.

[26] Chun-Lung Chen and Chuen-Lung Chen. Hybrid metaheuristics for unrelated parallel machine scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 43(1-2):161, 2009.

[27] Jeng-Fung Chen. Scheduling on unrelated parallel machines with sequence-and machine-dependent setup times and due-date constraints. *The International Journal of Advanced Manufacturing Technology*, 44(11-12):1204–1212, 2009.

[28] Jeng-Fung Chen. Scheduling on unrelated parallel machines with sequence-and machine-dependent setup times and due-date constraints. *The International Journal of Advanced Manufacturing Technology*, 44(11-12):1204–1212, 2009.

[29] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[30] Edward G Coffman, Jr, Michael R Garey, and David S Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[31] Mohammad I Daoud and Nawwaf Kharma. Efficient compile-time task scheduling for heterogeneous distributed computing systems. In *12th International Conference on Parallel and Distributed Systems-(ICPADS'06)*, volume 1, pages 9–pp. IEEE, 2006.

[32] Indraneel Das and John E Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM journal on optimization*, 8(3):631–657, 1998.

[33] Mateus Rocha de Paula, Geraldo Robson Mateus, and Martín Gómez Ravetti. A non-delayed relax-and-cut algorithm for scheduling problems with parallel machines, due dates and sequence-dependent setup times. *Computers & Operations Research*, 37(5):938–949, 2010.

[34] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[35] Thanh-Cong Dinh and Hyerim Bae. Parallel servers scheduling with dynamic sequence-dependent setup time. In *Intelligent Decision Technologies*, pages 79–87. Springer, 2012.

[36] A. E. Ezugwu and F. Akutsah. An improved firefly algorithm for the unrelated parallel machines scheduling problem with sequence-dependent setup times. *IEEE Access*, 6:54459–54478, 2018.

[37] Absalom E Ezugwu, Olawale J Adeleke, and Serestina Viriri. Symbiotic organisms search algorithm for the unrelated parallel machines scheduling with sequence-dependent setup times. *PloS One*, 13(7), 2018.

[38] Baoqiang Fan and Guochun Tang. A column generation for a parallel machine scheduling with sequence-dependent setup times. *Tongji Daxue Xuebao/ Journal of Tongji University(Natural Science)*, 34(5):680–683, 2006.

[39] Yaping Fu, Min Huang, Hongfeng Wang, and Guanjie Jiang. An improved nsga-ii to solve multi-objective optimization problem. In *The 26th Chinese Control and Decision Conference (2014 CCDC)*, pages 1037–1040. IEEE, 2014.

[40] Bernat Gacias, Christian Artigues, and Pierre Lopez. Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, 37(12):2141–2151, 2010.

[41] Sara Gestrelius, Martin Aronsson, and Anders Peterson. A milp-based heuristic for a commercial train timetabling problem. *Transportation Research Procedia*, 27:569–576, 2017.

[42] AH Gharehgozli, R Tavakkoli-Moghaddam, and N Zaerpour. A fuzzy-mixed-integer goal programming model for a parallel-machine scheduling problem with sequence-dependent setup times and release dates. *Robotics and Computer-Integrated Manufacturing*, 25(4-5):853–859, 2009.

[43] Ravindra Gokhale and M Mathirajan. Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *The International Journal of Advanced Manufacturing Technology*, 60(9-12):1099–1110, 2012.

[44] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[45] Alper Hamzadayi and Gokalp Yildiz. Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. *Computers & Industrial Engineering*, 106:287–298, 2017.

[46] Magdy Helal, Ghaith Rabadi, and Ameer Al-Salem. A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3(3):182–192, 2006.

[47] Zheng-liang Hou and Xiu-ping Guo. Parallel machine scheduling with resources constraint and sequence dependent setup times. In *Proceedings of 2012 3rd International Asia Conference on Industrial Engineering and Management Innovation (IEMI2012)*, pages 801–811. Springer, 2013.

[48] Dayong Hu and Zhenqiang Yao. Genetic algorithms for parallel machine scheduling with setup times. In *The 2nd International Conference on Information Science and Engineering*, pages 1233–1236. IEEE, 2010.

[49] Dayong Hu and Zhenqiang Yao. Identical parallel machines scheduling with sequence-dependent setup times. *Jixie Gongcheng Xuebao(Chinese Journal of Mechanical Engineering)*, 47(16):160–165, 2011.

[50] Dayong Hu and Zhenqiang Yao. Parallel machines scheduling with sequence-dependent setup times constraints. *Advanced Science Letters*, 4(6-7):2528–2531, 2011.

[51] Chul Ju Hyun, Yeongho Kim, and Yeo Keun Kim. A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers & Operations Research*, 25(7-8):675–690, 1998.

[52] Xiaoyun Ji. *Graph partition problems with minimum size constraints*. Rensselaer Polytechnic Institute, 2004.

[53] Cheol Min Joo and Byung Soo Kim. Parallel machine scheduling problem with ready times, due times and sequence-dependent setup times using meta-heuristic algorithms. *Engineering Optimization*, 44(9):1021–1034, 2012.

[54] Edmar Hell Kampke, José Elias Claudio Arroyo, and André Gustavo Santos. Iterated local search with path relinking for solving parallel machines scheduling problem with resource-assignable sequence dependent setup times. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 107–118. Springer, 2010.

[55] Yong-Ha Kang, Sung-Shick Kim, and Hyun Joon Shin. A scheduling algorithm for the reentrant shop: an application in semiconductor manufacture. *The International Journal of Advanced Manufacturing Technology*, 35(5-6):566–574, 2007.

[56] Yong Ha Kang and Hyun Joon Shin. An adaptive scheduling algorithm for a parallel machine problem with rework processes. *International Journal of Production Research*, 48(1):95–115, 2010.

[57] Liangjun Ke, Qingfu Zhang, and Roberto Battiti. Moea/d-aco: A multiobjective evolutionary algorithm using decomposition and antcolony. *IEEE transactions on cybernetics*, 43(6):1845–1859, 2013.

[58] Jae-Gon Kim, Seokwoo Song, and BongJoo Jeong. Minimising total tardiness for the identical parallel machine scheduling problem with splitting jobs and sequence-dependent setup times. *International Journal of Production Research*, 0(0):1–16, 2019.

[59] Ji-Su Kim, Jung-Hyeon Park, and Dong-Ho Lee. Iterated greedy algorithms to minimize the total family flow time for job-shop scheduling with job families and sequence-dependent set-ups. *Engineering Optimization*, 49(10):1719–1732, 2017.

[60] Shiegheun Koh and Karunia A Mahardini. Heuristics for non-identical parallel machine scheduling with sequence dependent setup times. *Journal of Korean Institute of Industrial Engineers*, 40(3):305–312, 2014.

[61] ME Kurz and RG Askin. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International journal of production research*, 39(16):3747–3769, 2001.

[62] Young Hoon Lee, Kumar Bhaskaran, and Michael Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE transactions*, 29(1):45–52, 1997.

[63] Young Hoon Lee and Michael Pinedo. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474, 1997.

[64] Zne-Jung Lee, Shih-Wei Lin, and Kuo-Ching Ying. Scheduling jobs on dynamic parallel machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 47(5-8):773–781, 2010.

[65] Man-Fai Leung and Sin-Chun Ng. A hybrid algorithm based on moea/d and local search for multiobjective optimization. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[66] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE transactions on evolutionary computation*, 13(2):284–302, 2008.

[67] Xiaohui Li, Farouk Yalaoui, Lionel Amodeo, and Hicham Chehade. Metaheuristics and exact methods to solve a multiobjective parallel machines scheduling problem. *Journal of Intelligent Manufacturing*, 23(4):1179–1194, 2012.

[68] Ching-Jong Liao, Cheng-Hsiung Lee, and Hsing-Tzu Tsai. Scheduling with multi-attribute set-up times on unrelated parallel machines. *International Journal of Production Research*, 54(16):4839–4853, 2016.

[69] Ching-Jong Liao, Yu-Lun Tsai, and Chien-Wen Chao. An ant colony optimization algorithm for setup coordination in a two-stage production system. *Applied Soft Computing*, 11(8):4521–4529, 2011.

[70] Shih-Wei Lin, Chung-Cheng Lu, and Kuo-Ching Ying. Minimization of total tardiness on unrelated parallel machines with sequence-and machine-dependent setup times under due date constraints. *The International Journal of Advanced Manufacturing Technology*, 53(1-4):353–361, 2011.

[71] Shih-Wei Lin, Chung-Cheng Lu, and Kuo-Ching Ying. Minimization of total tardiness on unrelated parallel machines with sequence-and machine-dependent setup times under due date constraints. *The International Journal of Advanced Manufacturing Technology*, 53(1-4):353–361, 2011.

[72] Shih-Wei Lin and Kuo-Ching Ying. Abc-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times. *Computers & Operations Research*, 51:172–181, 2014.

[73] Yang-Kuei Lin and Feng-Yu Hsieh. Unrelated parallel machine scheduling with setup times and ready times. *International Journal of Production Research*, 52(4):1200–1214, 2014.

[74] Dong Liu, Qiang Huang, Yuanyuan Yang, Dengfeng Liu, and Xiaoting Wei. Bi-objective algorithm based on nsga-ii framework to optimize reservoirs operation. *Journal of Hydrology*, 585:124830, 2020.

[75] Rasaratnam Logendran, Brent McDonell, and Byran Smucker. Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34(11):3420–3438, 2007.

[76] Quan Lu and Maged M Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, 2006.

[77] Krzysztof Michalak. Improving the nsga-ii performance with an external population. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 273–280. Springer, 2015.

[78] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

[79] Mahdi Naderi-Beni, Ehsan Ghobadian, Sadoullah Ebrahimnejad, and Reza Tavakkoli-Moghaddam. Fuzzy bi-objective formulation for a parallel machine scheduling problem with machine eligibility restrictions and sequence-dependent setup times. *International Journal of Production Research*, 52(19):5799–5822, 2014.

[80] Manfred W Padberg and Laurence A Wolsey. Trees and cuts. In *North-Holland Mathematics Studies*, volume 75, pages 511–517. Elsevier, 1983.

[81] Michael Pinedo. *Scheduling*, volume 5. Springer, 2012.

[82] Masoud Rabbani, Razieh Heidari, and Reza Yazdanparast. A stochastic multi-period industrial hazardous waste location-routing problem: Integrating nsga-ii and monte carlo simulation. *European Journal of Operational Research*, 272(3):945–961, 2019.

[83] Mahdi Rambod and Javad Rezaeian. Robust meta-heuristics implementation for unrelated parallel machines scheduling problem with rework processes and machine eligibility restrictions. *Computers & Industrial Engineering*, 77:15–28, 2014.

[84] Pedro Leite Rocha, Martín Gómez Ravetti, Geraldo Robson Mateus, and Panos M Pardalos. Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 35(4):1250–1264, 2008.

[85] Rubén Ruiz and Carlos Andrés-Romano. Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 57(5-8):777–794, 2011.

[86] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[87] Marcos Wagner Jesus Servare Junior, Helder Roberto de Oliveira Rocha, José Leandro Félix Salles, and Sylvain Perron. A linear relaxation-based heuristic for iron ore stockyard energy planning. *Energies*, 13(19):5232, 2020.

[88] Pankaj Sharma and Ajai Jain. A review on job shop scheduling with setup times. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 230(3):517–533, 2016.

[89] A Sathya Sofia and P GaneshKumar. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using nsga-ii. *Journal of Network and Systems Management*, 26(2):463–485, 2018.

[90] Nitin Srinath, I Ozan Yilmazlar, Mary E Kurz, and Kevin Taaffe. Introducing preferences in scheduling applications. *Computers & Industrial Engineering*, page 107831, 2021.

[91] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[92] Gaurav Srivastava, Alok Singh, and Rammohan Mallipeddi. Nsga-ii with objective-specific variation operators for multiobjective vehicle routing problem with time windows. *Expert Systems with Applications*, 176:114779, 2021.

[93] Weihua Tan, Xiaofang Yuan, Jinlei Wang, and Xizheng Zhang. A fatigue-conscious dual resource constrained flexible job shop scheduling problem by enhanced nsga-ii: An application from casting workshop. *Computers & Industrial Engineering*, 160:107557, 2021.

[94] Reza Tavakkoli-Moghaddam, Farid Taheri, Mohammad Bazzazi, M Izadi, and Farrokh Sassani. Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Computers & Operations Research*, 36(12):3224–3230, 2009.

[95] Reza Tavakkoli-Moghaddam, Farid Taheri, Mohammad Bazzazi, M Izadi, and Farrokh Sassani. Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Computers & Operations Research*, 36(12):3224–3230, 2009.

[96] M Duran Toksarı, Daniel Oron, and Ertan Güner. Some scheduling problems with past sequence dependent setup times under the effects of nonlinear deterioration and time-dependent learning. *RAIRO-Operations Research*, 44(2):107–118, 2010.

[97] S Ali Torabi, Navid Sahebjamnia, S Afshin Mansouri, and M Aramon Bajestani. A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. *Applied Soft Computing*, 13(12):4750–4762, 2013.

[98] C Tsai and C Tseng. Unrelated parallel-machines scheduling with constrained resources and sequence-dependent setup time. In *Proceedings of the 37th International Conference on Computers and Industrial Engineering, Alexandria, Egypt*, pages 20–23, 2007.

[99] Shunji Umetani, Yuta Fukushima, and Hiroshi Morita. A linear programming based heuristic algorithm for charge and discharge scheduling of electric vehicles in a building energy management system. *Omega*, 67:115–122, 2017.

[100] Eva Vallada and Rubén Ruiz. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622, 2011.

[101] Ari PJ Vepsalainen and Thomas E Morton. Priority rules for job shops with weighted tardiness costs. *Management science*, 33(8):1035–1047, 1987.

[102] I-Lin Wang, Yi-Chi Wang, and Chih-Wei Chen. Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics. *Flexible Services and Manufacturing Journal*, 25(3):343–366, 2013.

[103] Penghong Wang, Jianrou Huang, Zhihua Cui, Liping Xie, and Jinjun Chen. A gaussian error correction multi-objective positioning model with nsga-ii. *Concurrency and Computation: Practice and Experience*, 32(5):e5464, 2020.

[104] Yue Xi and Jaejin Jang. Scheduling jobs on identical parallel machines with unequal future ready time and sequence dependent setup: An experimental study. *International Journal of Production Economics*, 137(1):1–10, 2012.

[105] Chuanbo Xu, Yiming Ke, Yanbin Li, Han Chu, and Yunna Wu. Data-driven configuration optimization of an off-grid wind/pv/hydrogen system based on modified nsga-ii and critic-topsis. *Energy Conversion and Management*, 215:112892, 2020.

[106] Kuo-Ching Ying and Hui-Miao Cheng. Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 37(4):2848–2852, 2010.

[107] Kuo-Ching Ying, Zne-Jung Lee, and Shih-Wei Lin. Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 23(5):1795–1803, 2012.

[108] YiZeng Zeng, Ada Che, and Xueqi Wu. Bi-objective scheduling on uniform parallel machines considering electricity cost. *Engineering Optimization*, 50(1):19–36, 2018.

[109] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[110] Xiaoyan Zhu and Wilbert E Wilhelm. Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(11):987–1007, 2006.

[111] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[112] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.