Smart Manufacturing Scheduling:

Blending AI, Optimization, and Stochastic Modeling

John Woodward

Data Science, University of Wisconsin

DS 785: Capstone

Dr. Tracy Bibelnieks

December 8th, 2024

Author Note

"AI" has become a catch-all phrase, encompassing fields from supervised and unsupervised machine learning to generative pre-trained transformers (e.g., ChatGPT). This paper used "AI" synonymously with supervised and unsupervised machine learning (ML) for accessibility. While "ML" is the more precise term, the choice of "AI" aimed to engage nontechnical readers while the project addressed the technical details of the algorithms. This approach ensured that the paper resonated with a diverse audience.

Abstract

At industrial bottlenecks, scheduling decisions often rely on guesswork or simplistic algorithms—overlooking resources and profits. This business case study blended AI, optimization, and uncertainty (i.e., stochastic modeling) to address two parallel-processing machines sharing a single queue, incorporating real-world factors such as sequence-dependent setups (i.e., asymmetry), machine-specific incapabilities, and unpredictable upstream queuing. Traditional dispatching scheduling treats each machine as independent, ignoring the advantages of cooperation. In contrast, this project's cooperative approach optimized maximum flow time and average machine work time, thoroughly evaluated with synthetic yet lifelike problem sets. By integrating discrete event simulation, machine learning (ML), scalable optimization, and evaluating scheduling decisions iteratively (i.e., sequential policy), the approach achieved 20% improvements—reaching up to 60% in some scenarios—over traditional methods (and realized 10% gains even at larger 75-job queues), all within a real-time 10-second solve using a personal desktop. These results demonstrate to companies that meeting complex problems with decision science delivers significant practical improvements, showcasing its transformative power to monetize data and drive long-term competitive advantages.

Keywords: Smart manufacturing, operations scheduling, optimization, real-time, artificial intelligence (AI), machine learning (ML), stochastic modeling, sequential policy, robust optimization, discrete event simulation, prescriptive analytics, decision science, operations research, high-performance computing

Table of Contents

Abstract 2

Table of Contents 3

List of Figures 7

List of Tables 8

Chapter I: Introduction 9

Manufacturing and Scheduling Opportunity 9

Manufacturing Scenarios 10

The Manual Schedulers 10

The Inadequate Schedulers 10

Overall Project Goals Statement 10

Study Purpose 11

Project Objectives 12

Study Significance 13

Limitations 14

Computational Limitations 14

Generalization Limitations 15

Conclusion 15

Chapter II: Literature Review 16

Operations Scheduling Tradition 16

Queueing Theory 19

Optimization 20

High-Performance Computing With Optimization 21

Machine Learning With Optimization 22

Performant AI vs Explainable AI 23

Parallel Machine Scheduling Classification 24

Stochastic Optimization 25

Robust Optimization and Chance Constraints 26

Stochastic Programming 26

Sequential Optimization 27

Conclusion 28

Chapter III: Methodology 29

Methodology Outline 29

Data Construction and Preparation 29

Realistic and Understood Data 30

Problem Set Generation 37

Model Development 40

Discrete Event Simulation (DES) Model, With Traditional and Unsupervised Clustering Rules 40

MILP Optimization Models 48

Model Evaluation and Performance Metrics 60

Evaluation Design 60

Performance Evaluation Metrics 61
Conclusion 61
Chapter IV: Results 63
Hyperparameter Tuning With AI 64
Deterministic Optimization Solver Pre-training 66
Stochastic Optimization Solver Pre-training 71
Holdout Problem Evaluations With Traditional SPT Baseline 77
AI LPT Evaluation 77
Deterministic Optimization Evaluation 78
Stochastic Optimization Evaluation 79
Comparing and Contrasting All Algorithms 80
Conclusion 81
Chapter V: Discussion 83
Summary of Findings 83
Summary of Results by Project Objectives 83
Discussion of Implications for Business 84
Limitations of Results and Future Research Suggestions 86
DES and Clustering Limits and Next Steps 86
Hyperparameter Tuning Limits and Next Steps 86
MILP Model Limits and Next Steps 87

Lookahead Model Limits and Next Steps 90

Miscellaneous Suggestions for Future Experimentation 92

Conclusion 92

References 94

Appendix A: Consideration of Optimal Decision Trees 101

Appendix B: Considerations of Rolling Horizon Policy Methodology 102

Appendix C: Code URL 104

List of Figures

Figure 1 *Distribution of Articles on AI in Manufacturing by Algorithm, Supervision Type, and Function* ... 22

Figure 2 *Machine Learning Classification Types* ... 25

Figure 3 *Generating 20 Recipe Temperatures (Four Modes*) ... 32

Figure 4 *Generating 20 Recipe Expirations (Three Modes)* ... 33

Figure 5 *Heatmap of Machine Setup Minutes (Recipe-to-Recipe Clusters)* ... 36

Figure 6 *Example of Generating Two Jobs' Remaining Minutes (One Job per Machine)* ... 37

Figure 7 *Example of Generating N-2 Jobs' Arrival Minutes (N=Queue Size=3)* ... 38

Figure 8 *Flow Chart of Discrete Event Simulation Algorithm* ... 42

Figure 9 *25-Job Sequence Gantt Chart ("Traditional_LPT" Algorithm)* ... 43

Figure 10 *Unique Scheduling Sequences for Recipe Clusters* ... 45

Figure 11 *Performance of Machine Sequencing Pairs on Full Training Set* ... 47

Figure 12 *Illustration of Optimal VRP Solution (Four Vehicles on 16 Nodes)* ... 48

Figure 13 *Deterministic Optimization: Bayesian Optimization (Optuna) for 25 and 75-Queue (N=2458)* . 67

Figure 14 *Deterministic Optimization: 'Delta_Start' Values by Queue Size and Outcome (N=2458)* ... 68

Figure 15 *Deterministic Optimization: Hyperparameter Importance of 25 Versus 75-Queue* ... 69

Figure 16 *Stochastic Optimization: Bayesian Optimization (Optuna) for 25 and 75-Queue (N=552)* ... 73

Figure 17 *Stochastic Optimization: 'Delta_Start' Values by Queue Size and Outcome (N=552)* ... 74

Figure 18 *Stochastic Optimization: Hyperparameter Importance of 25 Versus 75-Queue* ... 75

Figure 19 *AI LPT Algorithm Performance (N=400)* ... 78

Figure 20 *Deterministic Optimization Algorithm Performance (N=400)* ... 79

Figure 21 *Stochastic Optimization Algorithm Performance (N=400)* ... 80

List of Tables

Table 1 *Manufacturing Job Sequencing Example: One-Machine Job Queue* 17

Table 2 *Static Factory Settings for Synthetic Problem Generation (20 Recipes)* 34

Table 3 *Training Problem Set of Queue Size 10, Problem Number 2 (2/750 Problem Set)* 39

Table 4 *Methodology of Evaluating Traditional Versus Integrated Models* 60

Table 5 *Deterministic Optimization: Mixed Effects of Best-Three Models (Relative Objective Improvement %)* 71

Table 6 *Stochastic Optimization: Mixed Effects of Best-Three Models (Relative Objective Improvement %)* 76

Table 7 *Linear Mixed Effects Model of All Models (Relative Objective Improvement %)* 81

Chapter I: Introduction

Manufacturing and Scheduling Opportunity

In the United States, traditional manufacturing schedulers at *bottlenecks*[1] tend to be managed by (1) human intuition or (2) basic scheduling rules. This complacency in tradition often results in inefficient resource allocation, lost profits, and employee burnout, especially during peak demand times.

Although manufacturing collects enormous amounts of data from its Internet of Things (IoT) devices, data is typically only used reactively for historical, descriptive analytics rather than proactively for real-time decision-making aids or analytics that prescribe solutions (i.e., prescriptive analytics). *Smart manufacturing* or using prescriptive analytics (i.e., optimization) with predictive analytics (i.e., AI) could help manufacturers profit in highly competitive and uncertain times, such as supply shocks like COVID-19 or recessions.

Scheduling queues of jobs with optimization has the potential to advise decisions that maximize profitability. Likewise, manufacturers have long needed real-time scheduling tools to solve demanding real-world problems like scheduling jobs. However, attempts to schedule purely with optimization have struggled to solve real-world-scaled problems.

The following section presents two hypothetical scenarios to illustrate the challenges faced in traditional manufacturing scheduling.

Manufacturing Scenarios

The Manual Schedulers

An expert operator, Lisa, struggles to manage a two-machine bottleneck[1] that oversees millions of units daily. She is overwhelmed with the 50-100 batch *jobs*[2] and high-pressure decisions on costly setups made by intuition. Her supervisor scolded her more than once for impossible schedules, such as blundering in assigning a job to a machine that could not process its recipe. As personal responsibilities clash with high-pressure work demands—like Lisa's child being sick at school and needing to be picked up and her supervisor requesting a broader schedule than usual—she faces rising anxiety, teetering on the edge of panic due to the unmanageable burden.

The Inadequate Schedulers

A skilled operator, David, faces constant costly setups. At the same time, the factory's mega press two-machine bottleneck has stacks and stacks of jobs waiting in the queue from inefficient setups back-to-back. Despite being told to "follow the schedule" and "it is a smart system," David thinks otherwise, with the excessive setups wasting valuable materials, time, and productivity, and even high-value products expiring in the queue. Still, if he deviates significantly from the schedule again, management will give him another warning. Sighing, he debates whether to send this feedback.

Overall Project Goals Statement

The author's previous work experience and optimization efforts laid a firm foundation for this case study and its deeper dive. This project builds on the author's prior groundwork in prescriptive analytics, where a flawed scheduling implementation struggled to scale to 25 jobs in the queue. The

[1] A *bottleneck* is in all flowing systems; in manufacturing, it is one machine group slowing the entire factory's pace.
[2] A *job* in manufacturing is a batch of product units flowing through a series of machine steps outlined in a *routing*, where each step—like cutting or shaping—the job queues, processes, and moves to the next until completion.

author has since learned creative improvements in data science (e.g., multithreading, warm starts, AI) to address scaling issues well.

Specifically, a sturdy foundation—a fully functional optimization model—was completed before the case study. Thanks to this, rather than building a deterministic optimization model from scratch, the author had the chance to focus on enhancing this model with data science methods learned during their master's program.

The author chose this project because it commits a step toward their dream: a career that blends machine learning, decision optimization, and planning under uncertainty (i.e., stochastic modeling). Therefore, this project's theme was integrating AI, optimization, and stochastic modeling. With a decade of industry experience, the author envisions deploying such integrated applications to benefit companies, their customers, and many communities.

Study Purpose

Even assuming manufacturers have all the data configured (e.g., data warehouses, ETL), they frequently use basic scheduling rules to manage a bottleneck—such as sorting jobs using their *recipe*[3] and Shortest Processing Time (SPT) or deciding with guesswork—squandering time and money.

Instead, what if a data science application optimized key business metrics at a bottleneck better than the current methods? In that case, it may shape manufacturers' profit margins and competitive advantages, particularly in increased uncertainty or complex markets.

Two key manufacturing bottleneck metrics drive profitability:

[3] In manufacturing, a *recipe* defines a machine's precise settings for a job's *routing step*; much like a cookbook page that instructs oven settings for preparing a meal. For example, Product A's routing on step 29 may instruct to on Machine One use "Recipe 10" (e.g., 100 units per minute, 500°C).

1. *Max flow time*: The slowest machine's complete work sequence time gages the group's throughput.
2. *Average Machine Work Time*: The average of all machines' work time. Reducing this time saves operating expenses and increases profit margins.

These two metrics complement each other and reflect the profit realities of production. So, this project optimized using a time *biobjective* among multiple machines, *minimizing* (1) the *max makespan and* (2) *the average machine work time*. It also aimed to improve its reliability with AI, realism, and flexibility with stochastic modeling.

Project Objectives

This project aimed to achieve the following milestones to optimize scheduling:

1. *Deterministic Optimization*: Deliver 20%+ improvement through deterministic optimization under certainty.
2. *AI-Boosted Scheduling*: Achieve 10%+ improvement using AI compared to control, optimizing throughput and total work time (max makespan and total makespan).
3. *Stochastic Optimization*: Attain 30%+ improvement using stochastic optimization under uncertainty.
4. *Speed and Scalability*: Optimize schedules within five minutes and a 5% *optimality gap*[4] for 99.9% of 25-job problems, 99% of 50-job problems, and 95% of 75-job problems.

The first three objectives measured scheduling effectiveness in series, while the fourth assessed performance quality.

[4] An *optimality gap* is the relative difference between a solution and the best possible—for example, if a schedule takes 110 minutes but could take 100 minutes, there would be a 10% optimality gap.

Study Significance

The case study benchmarked traditional scheduling and compared it to an integrated approach with optimization, AI, and stochastic modeling.

Though the study generated these problem sets synthetically, the data mirrored real-world manufacturing complexities like the following:

- blended random probability distributions, such as poisson, exponential, uniform, and triangular
- job unit quantity, arrival times, and expiration times
- recipe details, like temperature (50°C–350°C) and processing speeds
- processing incapability of specific recipes on certain machines[5]
- thermodynamics, differences between heating and cooling times (linear heating versus nonlinear cooling)

The case study provided precise and quantifiable improvements for one pseudo factory environment and its two-machine bottleneck, such as a 20% increase in throughput at bottlenecks and similar reductions in average machine work time. The project demonstrated that an integrated approach worked well with scheduling, performing well on larger volumes while maintaining real-time performance.

These better-quality schedules were also solved using a personal computer configuration, showing that increased profit margins may be possible if a manufacturer's Decision Science Department replicated implementing these algorithms, modeling and training them in their factory environment.

[5] A job assigned to a machine must have a compatible recipe, or it will not run. It is like trying to cook a large turkey (job) in a toaster (machine)—the equipment is unsuitable.

Similarly, those hypothetical scenarios of the inefficient and manual schedulers have good news when followed up after deploying the algorithms in production:

- *Lisa's automated scheduler*: After implementation, she breathed many sighs of relief because it automatically generated the proper schedule according to complex constraints like jobs expiring and arriving from elsewhere. She has a precious life-work balance again and feels less burnt out. Satisfied, Lisa imagined working there longer with fewer headaches.
- *David's more efficient scheduler*: IT and data scientists trained the integrated application on David's factory data and calibrated the smart manufacturing algorithm for implementation. David verified and crosschecked that the recipe-to-recipe setup times were valid, further improving it. He felt his ongoing contributions improved the bottleneck flow iteratively, training it, so he saw the value in his continuous collaborations. Fulfilled, David trusted that the system handled data better than he could.

Lisa and David's hypothetical scenarios benefited their mental health and well-being, as well as the manufacturer's operations, with fewer scheduling blunders and increased job satisfaction and productivity, making operations smoother.

The study aimed to inspire the manufacturing industry to adopt "decision science"—integrating AI, optimization, and stochastic modeling—into their scheduling systems, helping them modernize their operations and profit in real-time, uncertain environments.

Limitations

The following were the limitations encountered during the capstone project.

Computational Limitations

The computational setup was a high-performance desktop with an NVIDIA GeForce GTX 3080 GPU, 23 processor threads (with a 24th thread as a buffer), and a Windows Subsystem for Linux (WSL2).

Likewise, it used an academic license on WSL2 for Gurobi, a commercial optimization solver; the project selected this Gurobi solver as it is known to handle optimization faster (e.g., including multithreading capability) than most open-source solvers, such as “CBC” (COIN-OR, n.d.). Likewise, the Linux operating subsystem offered better speed and scalability than Windows.

However, limitations, such as problem complexity and non-distributed computing, limited the queue size that the algorithm could handle. Thus, the project reduced the queue size to 75 to ensure success. In hindsight, the project did not evaluate the maximum scalable queue size (e.g., 100, 250, 1000) for gauging production readiness to meet larger-scale manufacturers. However, the proof-of-concept results were promising even with the limitations imposed. For more discussion, see Chapter V.

Generalization Limitations

Generalizing the project’s code has limitations, as its algorithms were pre-trained using random distributions for a synthetic factory and two machines. For example, buying a third machine changes things, as does new or updated recipes. Likewise, although manufacturers may feasibly adopt the general code and principles, applicability may vary, such as by their niche or data readiness.

Conclusion

Chapter I introduced the project's aim: improving manufacturing scheduling through blending optimization, AI, and stochastic modeling, along with relevant contexts. Chapter II reviewed the relevant literature. Chapter III described the methods for simulating the problem sets, a conventional scheduler algorithm, and their integrated algorithms approach. Chapter IV presented the results of applying these methods, demonstrating scheduling and real-time performance improvements. Finally, Chapter V summarized the results, discussing how replicating these decision science advancements could benefit manufacturers through increased profitability.

Chapter II: Literature Review

The study reviewed the literature on improving scheduling in smart manufacturing, focusing on blending AI, optimization techniques, and stochastic modeling. This chapter reviewed operations scheduling traditions, queuing theory, optimization methods like the Traveling Salesperson Problem (TSP), machine learning in optimization, high-performance computing, and stochastic optimization, providing insights to enhance scheduling beyond current industry standards—specifically for this case study, to achieve profitability improvements and real-time scalability.

Operations Scheduling Tradition

The literature review was conducted by first reviewing a standard industrial engineering undergraduate textbook by Nahmias (2009), specifically the chapter on 'Operations Scheduling,' to outline traditional scheduling practices. *Operations Scheduling* means sequencing the precise schedule for a job shop problem. A *job shop* is a "set of machines and workers who use [a] machine" where "jobs may arrive all at once or randomly throughout the day".

Specifically, the business case study focused on a *parallel-processing job shop*—where one machine group with identical machines cooperates on a shared job queue. Specifically, the project selected the factory's bottleneck (i.e., the machine group limiting the total flow). In the seminal book *The Goal*, Goldratt and Cox (2014) stress the importance of addressing bottlenecks to maximize profits.

Conventional rules, such as first-in-first-out (FIFO), shortest processing time (SPT), earliest due date (EDD), and others[6,7] have been adopted for decades as the tried-and-true algorithms for operations scheduling, according to Nahmias (2009). These traditional algorithms dictate how to manage a shared

[6] *Longest processing time (LPT)* is another common sequencing rule. Sort the list of jobs by processing time in descending order.

[7] *Critical ratio* (CR) is another common sequencing rule. To calculate: (due date – current time)/processing time at each selection and prioritize the smallest CR value job.

queue of jobs vying for processing completion. They sort specific data columns, like processing time (e.g., SPT and ascending), due date (e.g., EDD and ascending), or timestamp (e.g., FIFO and ascending).

Table 1

Manufacturing Job Sequencing Example: One-Machine Job Queue

Job	Processing Time	Due Date
1	11	61
2	29	45
3	31	31
4	1	33
5	2	32

Note. From *Production and Operations Analysis*, chapter "Operations Scheduling" (6th ed., p. 425), by S. Nahmias, 2009, McGraw-Hill/Irwin; reproduced from Example 8.1.

Table 1's case offers a problem to benchmark scheduling algorithms using (1) *average flow time* or the meantime a machine completes its queue; (2) *tardy* jobs, or not finishing jobs before due dates:

- FIFO sorted the queue by *ascending timestamps* (i.e., job), resulting in the sequence 1 – 2 – 3 – 4 – 5, which produced an average flow time of 54 minutes and three tardy jobs.
- SPT ordered the queue by *ascending processing time*, making the sequence 4 – 5 – 1 – 2 – 3, with an average flow time of 27 minutes and one tardy job.
- EDD ranked the queue by *ascending due date*, producing the sequence 3 – 5 – 4 – 2 – 1, leading to an average flow time of 47 minutes and, notably, four tardy jobs.

Therefore, each algorithm can strongly affect the results; moreover, each has pros and cons, and manufacturers have differing, conflicting objectives, such as minimizing average flow time, reducing setup times, and focusing on priority jobs, according to Nahmias (2009). Thus, manufacturers mix and

match rules across a factory of machine groups[8], such as one mixing SPT, EDD, and FIFO, according to a study by Ren et al. (2022).

Note that Table 1's illustration only shows one machine, not multiple, foreshadowing the sequencing challenges when adding realism. Similarly, Table 1 does not have a column for setup times—the time it takes to sequence from job to job—which creates a tricky combinatorics problem.

Furthermore, with parallel processing job shops, the text by Nahmias (2009, p. 446) supports using the Longest Processing Time (LPT) even over SPT, unless it is for just one machine, in which case SPT is mathematically superior, corroborated by a study by Gozali et al. (2019). This distinction, anchored in typical queueing behavior (e.g., exponential distribution), hints at the counterintuitive nature of selecting sequencing rules, emphasizing the importance of scheduling algorithm choices.

A meta-analysis of over 90 academic papers from Lei (2009) demonstrated that conventional techniques (e.g., sequencing sorting rules) were inadequate when better approaches were available:

> [The Job Shop Scheduling Problem] is *seldom solved* [emphasis added] using the conventional techniques [because of] the high complexity of the problem and the limited optimization ability of the conventional methods. Meta-heuristics such as [genetic algorithm] and [evolutionary algorithm] have become the main approaches. (p. 932)

This quote from Lei (2009) shows how basic scheduling methods fail to tackle complex scheduling problems successfully, stressing the need for sophisticated ones like those from this project.

[8] In previous work experience as industrial engineer, a blend of FIFO, critical ratio, and job priority was used.

Queueing Theory

From Solberg's (2009, p. 179) textbook for random processes, a *queue* occurs whenever entities (e.g., cars, people, and products) join a waiting line to be processed; notable cases are heavy car traffic and long, anxious waits at airports.

For most manufacturers, a product order can repeatedly visit a machine group's queue as separate jobs. For example, an active "Product A" order's routing might revisit the same machine group on Steps 10, 24, and 58, where each step has unique differences. However, because of real-life queueing (e.g., processing, transporting, and setups), many schedule plans are naïvely impossible, even with deterministic optimization and AI—only stochastic plans can genuinely address randomness.

According to Solberg (2009, pp. 198–200), randomness with queues is not intuitive, and many managers and laborers misjudge queues. Queueing behavior often confuses supervisors and workers, as its randomness can create idle periods and sudden surges. Notably, t*raffic intensity*, defined as the ratio of arrival to service rates, predicts *steady-state*, long-term queue behavior[9].

Because network queueing (e.g., factory) is random, Solberg (2009, p. 221) assures that it is impossible to predict schedules with certainty; advanced queueing models can only predict trends and *discrete event simulations* forecast timings. Simulating queueing with discrete event simulations (DES), like with SimPy (n.d.), is effective for imitating reality by modeling complex queueing; thus, DES is a valuable scheduling tool, said Hillier & Lieberman (2015, pp. 939– 940) and Nahmias (2009, p. 462).

Thus, the case study reviewed discrete event simulation's (DES) success in modeling complex queueing systems, such as time-tracking using queue and resource constraints in manufacturing.

[9] A traffic intensity near 100% results in queues approaching infinity (Solberg, 2009).

Notably, it is required for translating high-level scheduling rules into detailed, problem-specific schedules, crucial for the project's traditional and stochastic scheduling algorithms.

Optimization

Optimization is central to manufacturing and the heart of data science; at its core, optimization is about finding the best solution from all options, according to Hillier and Lieberman's text (2015, p. 16). Samuel Eilon, a star in operations research, said, 'Optimizing is the science of the ultimate; [satisfying] is the art of the feasible'. Thus, optimization strives for optimality within balancing real-world realities.

The case study addressed optimization's thorny problem, the Traveling Salesperson Problem (TSP), which is finding the shortest route through multiple points. A 49-US-state TSP was first solved in the 1950s by Dantzig et al. (1954) with linear programming; deceptively straightforward, the problem had 10^{62} possible answers, more than any computer today could enumerate. This seminal study laid the foundation for optimization in the United States and the world, from operations scheduling thousands of airplanes daily to planning same-day deliveries. TSP has spawned many formulation upgrades, such as time windows within visits by Desrochers and Laporte (1991), multi-objectives by Azadeh et al. (2016), and more efficient MILP TSP formulations by Pferschy and Staněk (2017). Likewise, with extensions into the multi-traveling salesperson problem (mTSP) by Breaban (n.d.) and vehicle routing problem (VRP) by Brandinu and Trautmann (2014) within the tourism bus schedule, the TSP is pervasive cross-industry.

While exact solutions for large-scale TSP are computationally challenging, hybrid approaches holistically combine exact methods (e.g., MILP), metaheuristics (e.g., genetic algorithm), heuristics (e.g., AI), and high-performance computing measures commonly help solve large-scale problems. For instance, Shahbazian et al. (2024, p. 93089) examined cutting-edge machine learning uses in VRP (like Q-learning and other Reinforcement Learning). Another example was the innovative use of polar

coordinate reference frames in mTSP by Wang et al. (2021), along with a genetic algorithm. Lastly, Evreka (2024) used clustering k-means with MILP for large-scale VRP solving.

High-Performance Computing With Optimization

High-performance computing techniques have been reviewed to achieve this project's objective of scalability and helping near-real-time decision-making. Progress in computational improvements and optimization solvers make it possible to find exact solutions for medium-scale problems efficiently, even in spreadsheets on local computers, like with Frontline Systems (2021) in Excel or the general spreadsheet implementations from Rasmussen (2011). Furthermore, MILP optimization solvers come in open-source (COIN-OR, n.d.; HiGHS, n.d.) and commercial (Gurobi Optimization, 2024; Hexaly, 2024; IBM, n.d.), the most supporting partial multithreading and parallel processing to rush solution times.

Multithreading and parallel processing instances, such as with Gurobi (2024) and Pyomo (n.d.), can hasten to solve optimization problems. By algorithmically dividing and conquering: taking a large problem and splitting it into smaller subproblems, large-scale instances were solved precisely, such as Othman et al. (2019) using k-means clustering and parallelizing a 2D mTSP problem into distinct TSPs. Likewise, using GPU CUDA, Qiao and Créput (2020) locally searched feasible but suboptimal solutions in parallel via k-opt swaps (e.g., 2-opt, 3-opt… k-opt). Most promising, recent cutting-edge advancements from Lu and Applegate (2024) pivoted from decades of purely CPU- to GPU-based multiprocessing using a “primal-dual hybrid gradient”; this method has proven significant linear algebra advances in parallelizing solving (matrix multiplication instead of matrix factorization), making it possible to solve large-scale problems distributively.

Hyperparameter tuning, which involves adjusting an algorithm’s parameters to maximize its performance, has been shown in the literature to enhance solver speeds significantly. Studies from Hosny and Reda (2024) and Ishihara and Limmer (n.d.) have demonstrated that methods like Bayesian

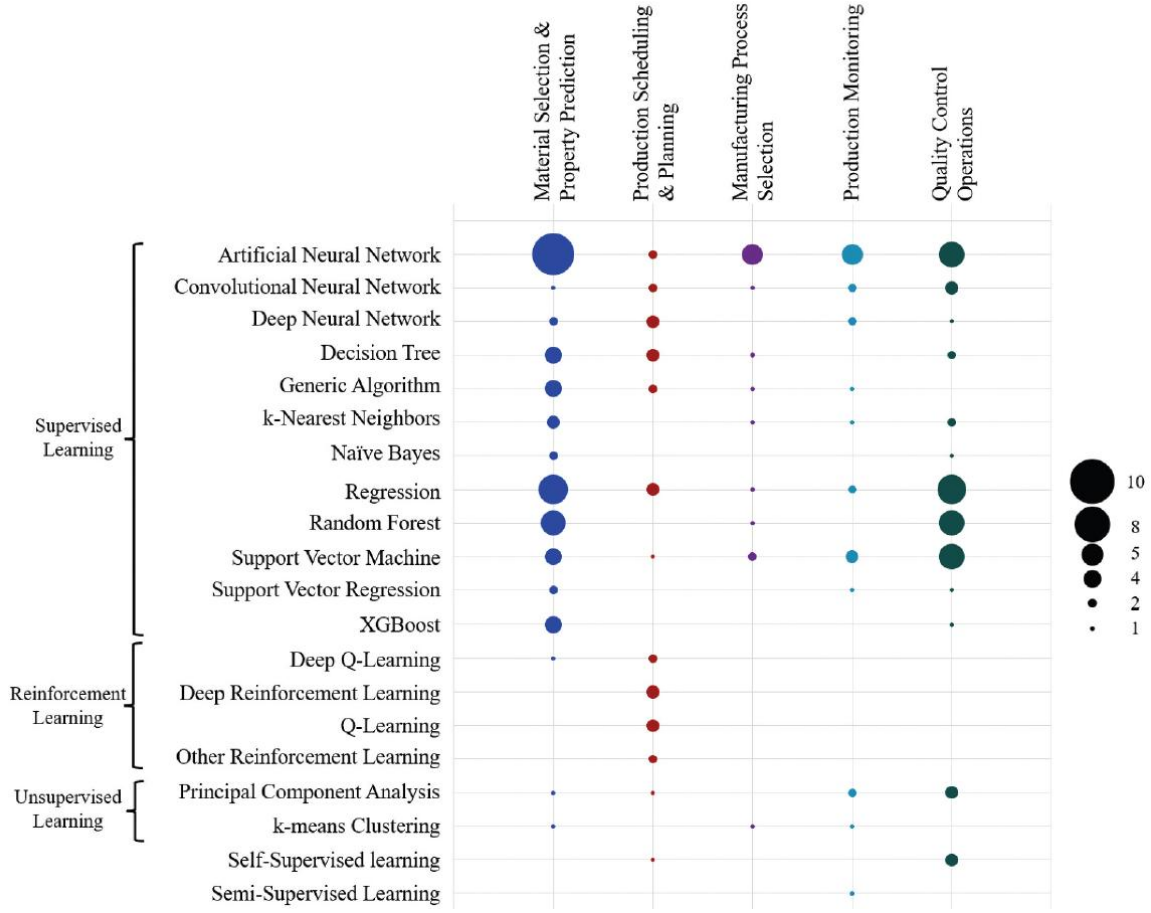
optimization are effective for hyperparameter tuning in optimization solvers, leading to substantial improvements in solve times, often by 40 – 60%. These findings stress the value of blending optimization with high-performance computing for scalability and practicality, especially in this project.

Machine Learning With Optimization

Blending in machine learning was critical to achieving the study's overall project goals, and it was also helpful in manufacturing meta-analysis studies like Ördek et al. (2024), as seen in Figure 1.

Figure 1

Distribution of Articles on AI in Manufacturing by Algorithm, Supervision Type, and Function



Note. From "Machine learning-supported manufacturing: A review and directions for future research" by B. Ördek, Y. Borgianni, & E. Coatanea, 2024, *Production & Manufacturing Research*, *12*(1). (https://doi.org/10.1080/21693277.2024.2326526); reproduced from Figure 5.

This meta-analysis by Ördek et al. (2024) of AI use in manufacturing (Figure 1) showed that while machine learning techniques are widely applied to tasks like "Material Selection & Property Prediction" and "Quality Control Operations," however, their use in "Production Scheduling & Planning" was still evolving and less mature; “Production Scheduling & Planning”, “Manufacturing Process Selection”, and “Production Monitoring” have fewer academic articles exploring their uses. Different machine learning models, including Neural Networks, Decision Trees, Genetic Algorithms, Reinforcement Learning, and Graph Neural Networks (GNNs), were applied in manufacturing, each with pros and cons.

Performant AI vs Explainable AI

The literature review covered a range of AI models applied in large-scale VRPs, probing the trade-offs between performance and interpretability. On the performance side, reinforcement learning techniques such as Q-learning, sequence-to-sequence models, attention mechanisms, and Graph Neural Networks (GNNs) are potent in predicting schedules, as detailed by Shahbazian et al. (2024). These models, often called 'big model' methods, are highly effective but have significant computational demands due to their dependence on big data. Likewise, while GNNs tempted exceptional promise for capturing intricate patterns, their training requirements, lack of explainability, and the need for substantial computational resources made them less suitable for the project despite their success.

On the other hand, manufacturers regularly prioritize interpretability to ensure maintainability, allowing operators to understand and trust algorithmic decisions. This preference for interpretability persists even when black-box models promise substantial performance gains, underscoring the critical role of transparency in AI adoption—particularly when explaining or troubleshooting scheduling decisions. In the Ördek et al. meta-analysis (2024), the following are the prevailing interpretable models:

- *Supervised*: Decision Trees, K-Nearest Neighbors (kNN), Regression, and Support Vector Machines (SVM) provided transparent decision-making processes. Decision Trees were especially noted for their direct interpretation, making them well-suited for manufacturing.
- *Unsupervised*: Techniques like Principal Component Analysis (PCA) and k-means clustering proved effective in uncovering clear data patterns and understanding complex datasets.
- *Metaheuristics*: Genetic Algorithm, often improving scheduling solutions through heuristics; likewise, Qiao & Créput (2020) explored k-opt swaps on TSP problems.

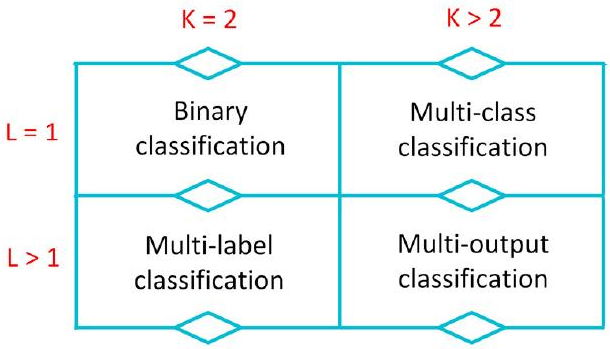
Parallel Machine Scheduling Classification

Still, even with supervised ML, a pressing issue remained: pinning scheduling rules for parallel machines with a shared queue—discussed in an industrial engineering textbook by Nahmias (2009, pp. 446–447). Namely, how does one sequence different recipes for scheduling purposes to encourage parallel machines to cooperate? Scheduling decisions for one machine influence the others, making traditional scheduling approaches that treat each machine separately just local optimization.

Instead of treating machines independently for a two-machine schedule prediction and with multiple possible sequence rule classifications, a *multi-output classification* seemed to capture these interdependencies as implemented in different industries by Ratnam (2022) and Yıldız et al. (2024). Figure 2 presents the types of classification models to clarify the dimensionality of multi-output models using a 2x2 matrix to illustrate different approaches, ranging from more straightforward single-label classifications (L=1, K=2) to more unfamiliar multi-label, multi-class models (L>2, K>2).

Figure 2

Machine Learning Classification Types



Note. L is the number of labels, and K is the number of possible values assigned to each label. Reproduced from "Improved chain-based multi-output classification for packaging planning" by S. N. Yıldız, F. Y. Okay, A. Islamov, & S. Özdemir, 202, *Procedia Computer Science*, *231*, 697–702. (https://doi.org/10.1016/j.procs.2023.12.159).

Figure 2 sets the stage for how multi-output models may capture interdependencies critical to scheduling parallel machines. One effective technique identified in the literature uses *chained*[10] classifiers, where each output prediction informs successive ones. In a distribution-center study by Yıldız et al. (2024), a chained Decision Tree classifier demonstrated superior performance over conventional KNN and Random Forest models in capturing interdependencies between classifications. This sequential chaining of ML models showed promise in improving machine-to-machine scheduling with a shared queue. Multi-output classification and chaining methods are available within Python libraries like scikit-learn (n.d.) and XGBoost (n.d.); however, they are in "experimental" modes as niche methods.

Stochastic Optimization

In deterministic optimization, all variables are assumed to be known with certainty, allowing perfect decisions. However, uncertainty is the only certainty in real-world manufacturing, leading to outcomes ranging from suboptimal to infeasible solutions—discussed in Hillier and Lieberman's operations research textbook (2015, pp. 264–267) and a stochastic textbook from Infanger (1994, p. xi).

[10] *Chaining* in multi-output models means predicting outputs in a sequence, where each prediction informs the next. For example, should Machine One's output be predicted first so that Machine Two can use it as a feature? Or vice versa? A goal is determining which output sequence delivers the best predictive results.

Stochastic optimization manages uncertainty and provides a more holistic approach to erratic variables. For this capstone project, integrating stochastic elements related to the unpredictable job arrival times, thus, promised potential for ensuring a more adaptive and robust schedule.

Robust Optimization and Chance Constraints

Robust optimization handles uncertainty by conservatively tuning constraints, ensuring feasibility under worst-case conditions, says Hillier and Lieberman (2015, pp. 264–267). In manufacturing, unpredictable job arrival times often lead to infeasible schedules if not appropriately anticipated. Robust optimization can mitigate this using the most conservative estimates, ensuring the schedule remains feasible even when jobs are delayed. For this project, robust optimization was practical; job arrival times strongly dictated which jobs could be scheduled, especially if those jobs were not in the current queue. If jobs arrived later than planned, the entire sequence risked becoming infeasible. By applying robust optimization to these critical constraints, the project ensured that such situations did not derail the schedule, maintaining feasibility even under worst-case conditions.

Chance constraints offer a flexible alternative to the strictly conservative nature of robust optimization, according to Hillier and Lieberman (2015, pp. 268–271). Instead of planning for the absolute worst case, chance constraints allow deadlines or conditions to hold probabilistically—such as ensuring that job deadlines were met 95% of the time rather than requiring 100%. This approach is helpful in scenarios where missing a deadline is undesirable but not catastrophic. In the context of this project, chance constraints were especially relevant to the right tail end of the time windows, where a job might slightly exceed the planned schedule without causing significant issues.

Stochastic Programming

While robust optimization handles constraints well, Hillier and Lieberman (2015, pp. 271–276) discuss that stochastic programming extends optimization by “looking ahead” at multiple possible

scenarios and is a powerful technique for scheduling when futures have probabilities. For this project, stochastic programming provided a way to consider future job arrival scenarios under uncertainty.

Ideal stochastic modeling approaches pioneered by Infanger (1994, pp. xi–xiii; 2011, p. viii) involved sophisticated scenario generation through *importance sampling*, a clever method to reduce sampling and the number of scenarios, and *sample average approximation* (SAA), a theorem approximating the objective model by blending the expected value of all scenarios. The project reviewed the literature on stochastic implementations such as (a) Gurobi's (2019) use in a simple stochastic SAA newsvendor problem, (b) importance sampling Python implementations by Sefidian (2022), and (c) a stochastic programming implementation by Upadhya (2022). Likewise, the literature noted stochastic solvers such as (a) add-ins in Excel by Frontline Systems (2021) and (b) powerful simulation-based optimizing by InsideOpt (n.d.). These works infer that for the case study, balancing complexity and practicality is crucial in modeling randomness.

Sequential Optimization

Sequential optimization, pioneered by Ghadimi and Powell (2022), addresses optimization by breaking complex problems into manageable steps; it allows each part to be addressed in sequence while updating information as it becomes available. This method is particularly beneficial for dynamically managing uncertainty. Similarly, the literature revealed that a technique used by Bischi et al. (2019), called *rolling horizon policy*, can succeed in large-scale planning, such as queue networks, by clipping the total planned horizons and iteratively solving a big problem as a series of smaller sequential ones.

In the context of this project, the sequential optimization and rolling horizon approaches were particularly relevant due to the unpredictable nature of job arrivals caused by queueing's effects. By focusing on immediate and near-future decisions rather than attempting to predict and solve all future requirements, each approach allowed for iterative, adaptive scheduling, aligning well with the project's objectives and mergeable with lookahead functionality, like in stochastic optimization.

Conclusion

The literature review emphasized the challenges and evolving approaches in smart manufacturing scheduling. Traditional methods, such as operations scheduling and queueing theory, provided solid foundational knowledge, while optimization techniques like the Traveling Salesperson Problem (TSP) aimed to improve them further. However, these traditional methods, including academia's TSP models, were still insufficient to tackle large-scale, real-world manufacturing problems. Moreover, the literature showed that machine learning and stochastic modeling advancements have confirmed significant potential for enhancing scheduling effectiveness. As a result, integrating AI, optimization, and stochastic modeling showed promise, justifying a comprehensive data science approach to overcoming the challenges addressed by this business case study.

Chapter III: Methodology

Scheduling operations in a parallel-processing job shop is overwhelmingly complex (e.g., recall the David and Lisa scenarios); thus, solving it requires a sophisticated methodology. This project tackled schedule solving by blending AI, optimization, and stochastic modeling with an appropriate and thoughtful application of data science techniques. The project objectives focused on evaluating these advanced methods versus traditional scheduling by seeking 10, 20, and 30% improvements, respectively, and in a scalable, real-time manner (i.e., 75-job queue within five-minute solves). This chapter chronologically defines the project's plan.

Methodology Outline

To achieve this, the methodology included at a high level:

1. Prepping the synthetically generated data.
2. Implementing a discrete event simulation (DES) to render high-level rules into schedules.
3. Porting prior work into a Python-based mixed integer linear programming (MILP) model.
4. Generalizing the implementation with a sequential policy that simulates the MILP over time.
5. Building on this sequential policy with a stochastic lookahead model that considers simple scenarios in the queue over time.
6. Cross-evaluating these models fairly and objectively.

The next sections of this chapter provide the rationale and rigor for this outlined methodology.

Data Construction and Preparation

This project relied on synthetic data from the start, as the original manufacturing information was confidential, limited in scope, and a trivial snapshot. However, this prior arrangement inspired the structure of the problems to meet the project's objectives. Moreover, with the technical rigor required

for solving the scheduling algorithms of the business case study, synthetic data was needed for training, validating, and testing.

Therefore, the project generated synthetic data: hundreds of problems for realistic simulations of factory conditions, hyperparameter tuning, and meticulous evaluation. This synthetic data captured the practical complexity needed for meaningful algorithmic tests by mimicking real-world factory settings. Designing the data required upfront planning for this case study, and integrating manufacturing industry experience offered several benefits:

1. *Realism with control*: The simulated data mirrored real-life manufacturing problems and allowed complete data comprehension since the underlying distributions were explicitly known.
2. *Simplified analysis and transformations*: Synthetic data simplified typically time-consuming tasks, such as extraction, transformation, and loading (ETL) and exploratory data analysis (EDA). This efficiency aided a better literature review, methodology, and evaluation focus.
3. *Balanced training and evaluation sets*: The synthetic data provided a fair and balanced training and evaluation structure with precise experimental controls.

Realistic and Understood Data

A bottleneck is the slowest step in a manufacturing system (Goldratt & Cox, 2014); while it can generally shift after increasing capacity (e.g., buying more machines), it is ultimately unavoidable. Each time a manufacturer upgrades a bottleneck machine group, it shifts to another one, but the cost of upgrading also escalates, often reaching tens or hundreds of millions of dollars. With each new bottleneck becoming more expensive to eliminate, manufacturers always reach an asymptomatic point where further expansions are no longer financially feasible. This forces operations to optimize within existing constraints. As a result, a bottleneck machine group typically correlates to only a few machines.

Thus, for this project, $k = 2$ machines were assumed. Keeping the parallel machine count low makes the model more manageable for this case study and still realistically represents real-world manufacturing, where cost constraints keep bottlenecks small.

Second, for this operations scheduling project, there needed to be a way for the jobs to be defined with processing and setup times. While it would have been simpler to generate times randomly per job and move on, manufacturers tend to use something else. They use what some call *recipes*, others call "work instructions" or "templates". These are the complete machine settings that vary product-to-product and step-to-step in routes and dictate how a machine processes a product. Fundamentally, before a machine can process a job, the machine needs precise, detailed instructions like speed, temperature, die type, etcetera; otherwise, nothing can be processed—recipes are crucial.

The synthetic dataset construction assumed fixed machine processing abilities for each recipe "factory-wide" (static for the rest of the project). A triangular distribution was used to represent the processing speed of all recipes by machine to make the data. Explicitly, for each recipe i and machine j, compatibility X_ij was generated as $X_{ij} \sim \text{Triangular}(0, 0.001, 10)$ units processed per minute, where 0 indicates incompatibility, 0.001 is the mode, and 10 represents the upper bound speed for possible units per minute. By setting a mode close to zero, this distribution biased all the problem data toward tricky cases where a given machine cannot process specific recipes. This aligns with real-world conditions where bottleneck machines are often selective in parallel capability.

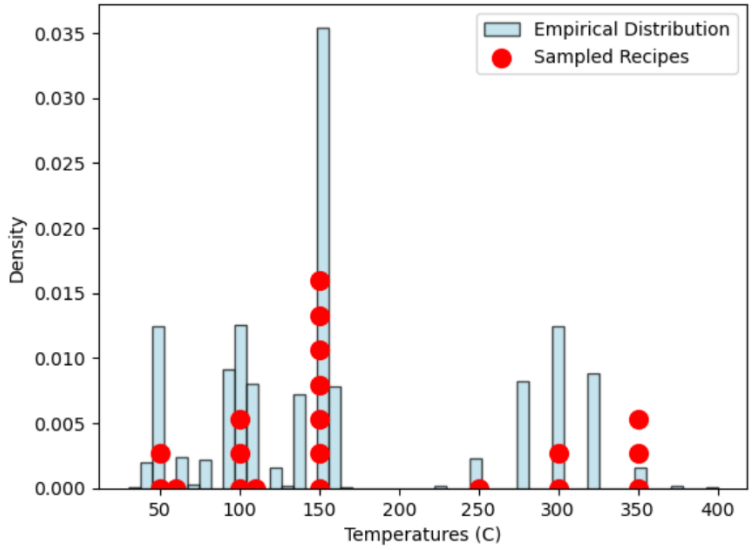
A post-hoc adjustment was needed to prevent recipes from having no capable machines. Specifically, for any recipe i where $X_{i1} = 0 \text{ and } X_{i2} = 0$ (indicating both machines could not process it), a random selection was made to assign a capability and processing speed to one machine. This involved generating a random value $r \sim \text{Uniform}(0,1)$; if $r > 0.5,\ X_{i1}$ was set to 1 unit per minute; otherwise, X_{i2} was set to 1 unit per minute. This ensured that each recipe could be processed by at least one

machine, preventing deadlocks in scheduling due to impossible schedules. See Table 2's "Units/Min Speed" column indicating incapability for the unit processing speeds.

A recipe's "Temperature" was how to estimate changeover times. As a result, much thought went into realistically segmenting them. Likewise, the project's initial AI algorithm needed to have them explicitly clustered; thus, an assumption was to randomly sample from a 4-modal distribution. In Figure 3, after four modes of normal distribution were made into an empirical distribution, the recipe temperatures were sampled 20 times for Table 2 to select the recipe temperatures randomly:

- *Mode 1*: $N(50\ °C, 5\ °C)$, 500 samples, rounded to 10 °C (the lowest cluster).
- *Mode 2*: $N(100\ °C, 10\ °C)$, 1000 samples, rounded to 10 °C.
- *Mode 3*: $N(150\ °C, 5\ °C)$, 1500 samples, rounded to 10 °C.
- *Mode 4*: $N(300\ °C, 25\ °C)$, 1000 samples, rounded to 25 °C (the highest cluster).

Figure 3
Generating 20 Recipe Temperatures (Four Modes)



Next, Figure 4 shows how 20 recipe expirations for Table 2 were chosen (similar to Figure 3):

- *Mode 1*: $\text{Poisson}(500)$, 100 samples, rounded to 10 minutes (the most restrictive mode).
- *Mode 2*: $N(1500, 50)$, 1000 samples, rounded to 25 minutes.
- *Mode 3*: $N(3500, 0)$, 2000 samples, rounded to 25 minutes (the least restrictive mode).

Figure 4

Generating 20 Recipe Expirations (Three Modes)

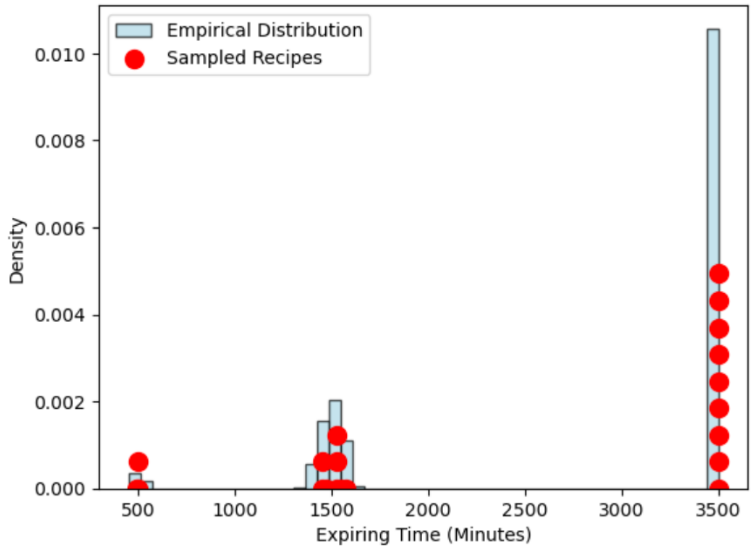


Table 2 joined all critical features into a single, static DataFrame, forming the backbone for every spawned problem. This table was central to the project, supporting each problem set and problem and deserving close attention for its influences on the problem space.

Table 2

Static Factory Settings for Synthetic Problem Generation (20 Recipes)

Cluster	Recipe	Units/Min Speed		Temperature	Probability	Expiring
		Machine1	Machine2	(°C)	of Visit (%)	Minutes
A	6	2	5	50	6.5	1450
A	13	0*	4	60	6.5	1575
A	18	7	2	50	6.5	3500
B	1	0*	1	100	6.5	500*
B	9	1	4	110	3.2	3500
B	19	2	5	100	3.2	3500
B	20	3	6	100	3.2	3500
C	2	1	1	150	3.2	3500
C	3	1	0*	150	6.5	1525
C	5	5	5	150	3.2	1550
C	7	3	7	150	3.2	1525
C	8	4	4	150	3.2	1475
C	11	7	3	150	3.2	490*
C	14	6	5	150	3.2	3500
D	4	0*	3	300	6.5	3500
D	10	8	4	350	6.5	3500
D	12	2	2	350	6.5	500*

Note. A job randomly selects a recipe or a table row when created.

[a] "Probability of Visit (%)" biases jobs to draw from difficult recipes more often, such as incapable (i.e., 0) or low or high temperatures (i.e., below 60 or above 300 °C), to boost difficulty.

[b] "Expiring Minutes" is the due date of when a recipe should start, with asterisks showing the short-to-expire ones.

"Temperature" correlates directly to setup costs in this project's bottleneck, so the greater the difference between recipes, the more setup costs. Thus, temperature determined the recipe clustering method. Given Table 2's recipe configurations and temperatures (in °C), the crucial setup costs—time required for machines to change between recipe temperatures—were estimated using thermodynamics, specifically Newton's Law of Cooling for cooling and a linear rate for heating. This created asymmetric setup times based on the order of temperature change:

1. For cooling transitions, the project applied Newton's Law of Cooling, $\frac{dT}{dt} = -k(T - T_{\text{ambient}})$ per hour, where T is the temperature, $T_{\text{ambient}} = 25$ °C, and $k = 0.79$ (calculated cooling rate[11]). By numerically solving this differential equation[12], cooling times to reach a target temperature were calculated within ±0.1 °C tolerance, applying realistic thermodynamics for cooling times.
2. Heating transitions, by contrast, were simplified with a constant linear rate of $\frac{dT}{dt} = 200$ °C per hour, assuming constant heating power. For a temperature increase from T_{initial} to T_{final}, heating time was computed as $t = \frac{T_{\text{final}} - T_{\text{initial}}}{200}$, reflecting steady heating commonly found in manufacturing.

Figure 5 visualizes these asymmetric setup times in a dual heatmap, with each cell showing the minutes required for each machine to transition from one recipe temperature to another. Machine 2 operates five minutes slower than Machine 1 for transitions between differing temperatures (e.g., 50 °C to 60 °C), introducing a slight but realistic asymmetry.

[11] Newton's law of cooling calculator (Calculator Academy, n.d.) was used to calibrate the cooling rate constant k so that cooling from 300 °C to 50 °C would take precisely 3 hours.

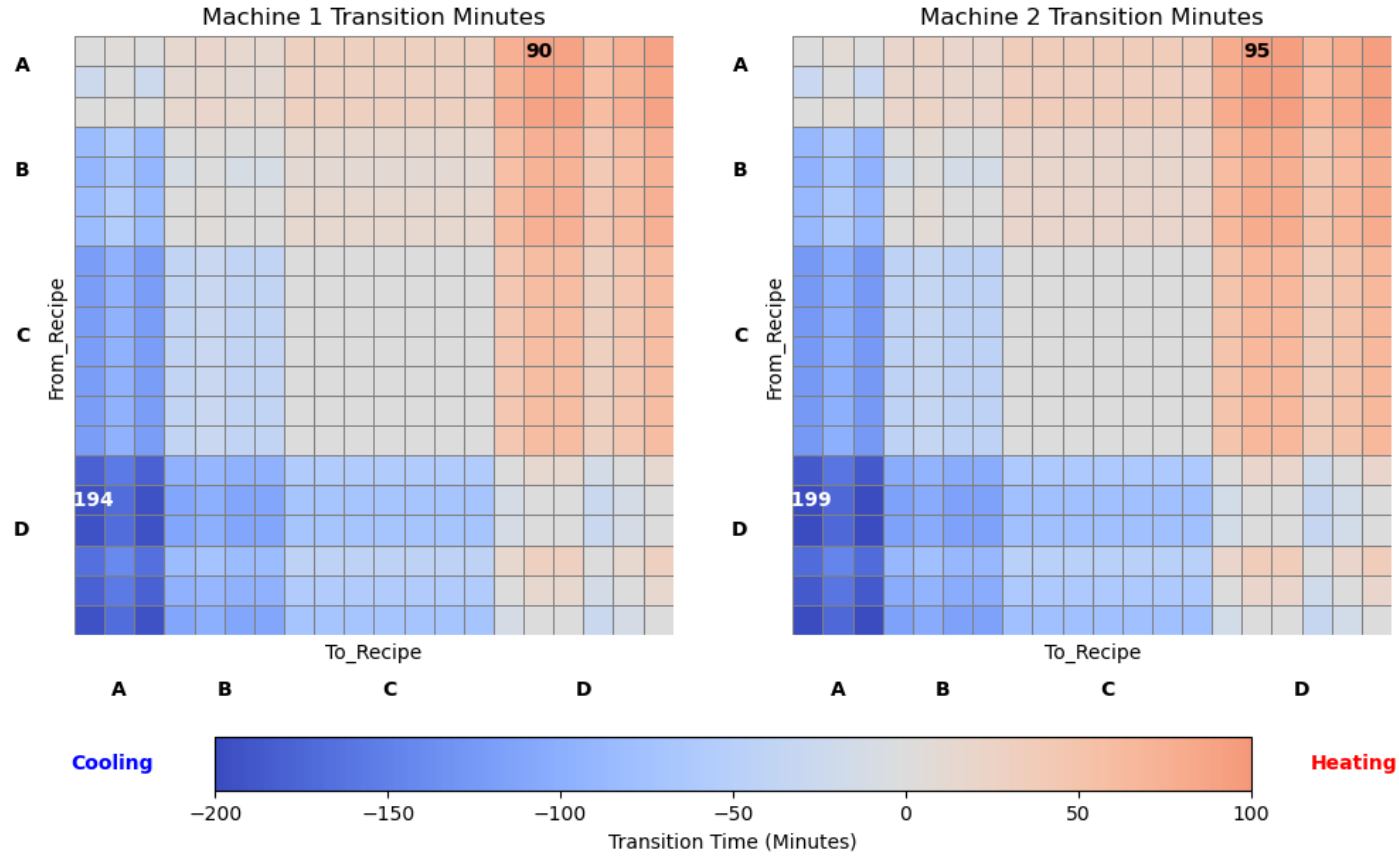
[12] Due to tight time constraints, the project used Python's *scipy.integrate* library with the *odeint* function and ChatGPT (OpenAI, 2024) to solve Newton's Law of Cooling numerically; investing extra time to locate an explicit solution was zero-sum. An explicit formula was later identified (What is Newton's Law of Cooling?, n.d.), consistent with prior coursework on differential equations.

This setup matrix (Figure 5) combines exponential cooling and linear heating models to realistically capture machine setup times between recipes, improving the project's realism to model job scheduling under diverse and complex setup conditions.

As a result, the static tables of 20 recipes (Table 2) and their corresponding recipe-to-recipe setup matrices per machine (Figure 5) were used to generate each problem consistently.

Figure 5

Heatmap of Machine Setup Minutes (Recipe-to-Recipe Clusters)



Note. This heatmap shows the cost of every recipe-to-recipe temperature change, with 20 by 20 recipes per machine or 800 cells. Notably, cooling times have exponentially greater costs than linear heating.

[a] Machine 1 and 2 are nearly identical, but Machine 2 was assumed to be 5 minutes slower than Machine 1 on setups with different temperatures (i.e., 50°C to 50°C on either machine is 0 minutes).

[b] While "negative minutes" were used to visualize cold transitions, time is always positive.

Problem Set Generation

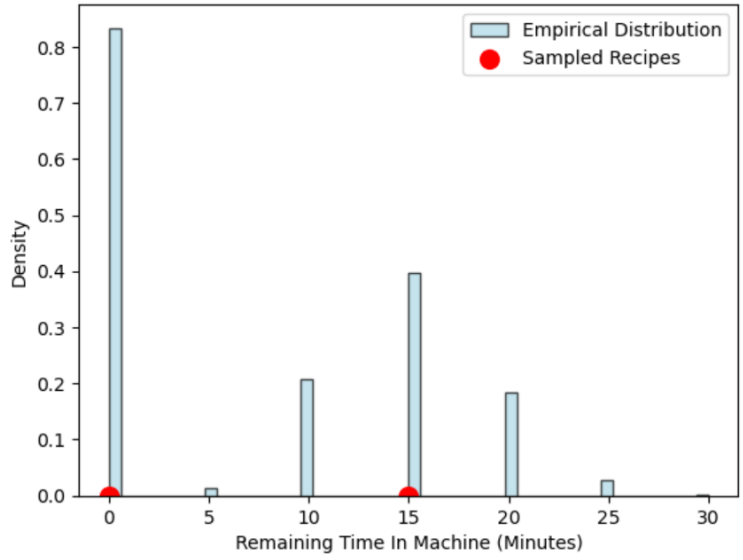
With critical pieces in place, only a few details lingered for creating distinct scheduling problems:

- *Remaining minutes*: The time left for each machine's processing (in-process vs. idle),
- *Arrival minutes*: The time until a job is available for scheduling,
- *Units in the job*: The workload required per machine according to each recipe.

The "remaining minutes" value begins scheduling problem generation, accounting for decision cases where a machine may still be in setup and not 100% finished processing its job. Furthermore, to establish the current recipe, the model starts with every machine having a job "loaded", either in-processing or post-processing. If a job is still in process, the machine remains occupied until it becomes available. Figure 6 illustrates one example of job assignments across two machines using random distributions: (a) *Mode 1*: $\mathrm{Uniform}(0, 0)$, 10000 samples (Idle machines, 50% chance assumed); (b) *Mode 2*: $\mathrm{Poisson}(15)$, 10000 samples, rounded to 5 minutes (In-process machines).

Figure 6

Example of Generating Two Jobs' Remaining Minutes (One Job per Machine)

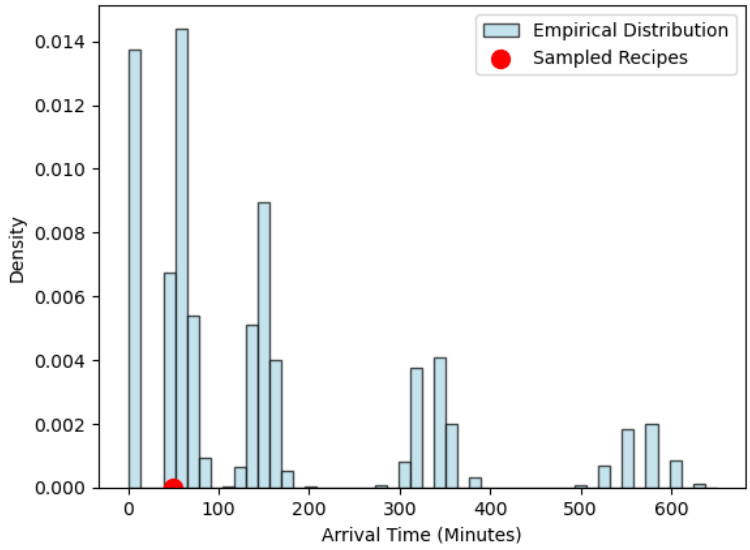


An important variable is “arrival minutes” when a job enters the machine group’s shared queue. This variable, the most stochastic element of the problem set, introduces hard constraints because early or late arrivals disrupt scheduling plans, making it one of the most influential random variables. Figure 7 shows arrival times sampled from a 5-modal empirical distribution to approximate "true" arrival minutes—a variable deliberately hidden from scheduling decisions:

- *Mode 1*: $\mathrm{Uniform}(0, 0)$, 2500 samples (current queue).
- *Mode 2*: $\mathrm{Poisson}(60)$, 5000 samples, rounded to 10 minutes (one step upstream).
- *Mode 3*: $\mathrm{Poisson}(149)$, 3500 samples, rounded to 15 minutes (two steps upstream).
- *Mode 4*: $\mathrm{Poisson}(335)$, 2000 samples, rounded to 25 minutes (three steps upstream).
- *Mode 5*: $\mathrm{Poisson}(565)$, 1000 samples, rounded to 30 minutes (four steps upstream).

Figure 7

Example of Generating $N - 2$ Jobs’ Arrival Minutes ($N = \text{Queue Size} = 3$)



Note. For the study, the queue size included the machine’s last processed or currently in process for consistency and convenience (i.e., two machines subtract the queue size).

Last is the number of units of a job; a simple random variable determined them: $\mathrm{Uniform}(10, 200)$ units. For this project, every job used this method for their workload amount.

After blending everything, including the factory settings and Figure 5 setup matrices, a problem's simulated random variables to a queue size (e.g., 10, 25), and merging DataFrames, a problem is produced. For visualization, see Table 3 for a problem made: fully ready to be solved.

Table 3

Training Problem Set of Queue Size 10, Problem Number 2 (2/750 Problem Set)

Job	Cluster	Recipe	Processing Minutes		Remaining	Queue	a	b	a_true	b_true	Units
			Machine1	Machine2	Minutes	Step					
1	C	2	242	242	25	0	0	3500	0	3500	242
2	A	13	10000*	52	30	0	0	1575	0	1575	207
3	C	14	12	14		0	0	3500	0	3500	71
4	D	16	71	10000*		1	60	3560	50*	3550	214
5	D	4	10000*	47		1	60	3560	50*	3550	141
6	B	1	10000*	144		1	60	560	50*	550	144
7	A	18	6	21		1	60	3560	60	3560	42
8	D	15	16	10000*		1	60	1585	70*	1595	33
9	B	1	10000*	34		2	149	649	150	650	34
10	C	8	33	33		2	149	1624	150	1625	133

[a] "Processing Minutes" is determined by dividing the units of each job by the processing speed in the recipe. For instance, Job 1 (Recipe 2, 1 unit/min) has 242 units, resulting in 242 minutes on either machine. Jobs with a 10,000-minute value marked indicate machine incapability for that job. Note that the first machine assignments are by job numbers, such as Job 1 for Machine 1 and Job 2 for Machine 2.

[b] Variables a and a_{true} and b and b_{true} signify time window constraints "Arrival Minutes" and "Expiring Minutes" in shorthand. $[a, b]$ and $[a_{\mathrm{true}}, b_{\mathrm{true}}]$ state expected or actual (true) time windows for each recipe. Asterisks on a_{true} values contrast random arrival timing with a's fixed times.

Model Development

The model development involved several coordinating methods to improve the optimization scheduling algorithms, each complementing the next. This project blended discrete event simulation (DES) and mixed-integer linear programming (MILP) optimization, extending with a sequential policy and a lookahead (stochastic). Each methodology added value, supporting the next.

1. DES provides queue-based scheduling for high-level scheduling rules for traditional and clustering methods to generate feasible solutions consistently—(a) the traditional solutions feed into the evaluation for every algorithm's baseline, and (b) the clustering solutions warm start the MILP-based sequential policy and lookahead models.
2. The MILP model constructs a biobjective and constraints, which are solved in a sequential and parametrized policy, later tuned with hyperparameter optimization.
3. Finally, the lookahead model (stochastic) extends these constraints, mixing in parallel worst-case scenarios of next-job arrivals to account for uncertain arrival times.

This framework was expected to outperform the traditional method, blending each method's strengths to tackle the operations scheduling problem.

Discrete Event Simulation (DES) Model, With Traditional and Unsupervised Clustering Rules

Why consider adding an intricate discrete event simulation (DES) for this project? Ironically, DES is the simplest means of translating high-level scheduling rules—traditional or clustering-based—into tangible schedule outputs. Drawing from a foundational career in queueing networks, DES was a natural fit for scheduling scenarios that rely on managing job sequences and their queueing for parallel processing machines. As Solberg's textbook on random processes highlights (Solberg, 2009, p. 221), simulating queueing networks allows for intuitive and wholly customizable modeling of job flows and

resource constraints. DES is often the only way to get detailed results, such as a direct schedule[13], and was long-established in textbooks like operations research (Hillier & Lieberman, 2015, pp. 939–940) and production planning (Nahmias, 2009, p. 462).

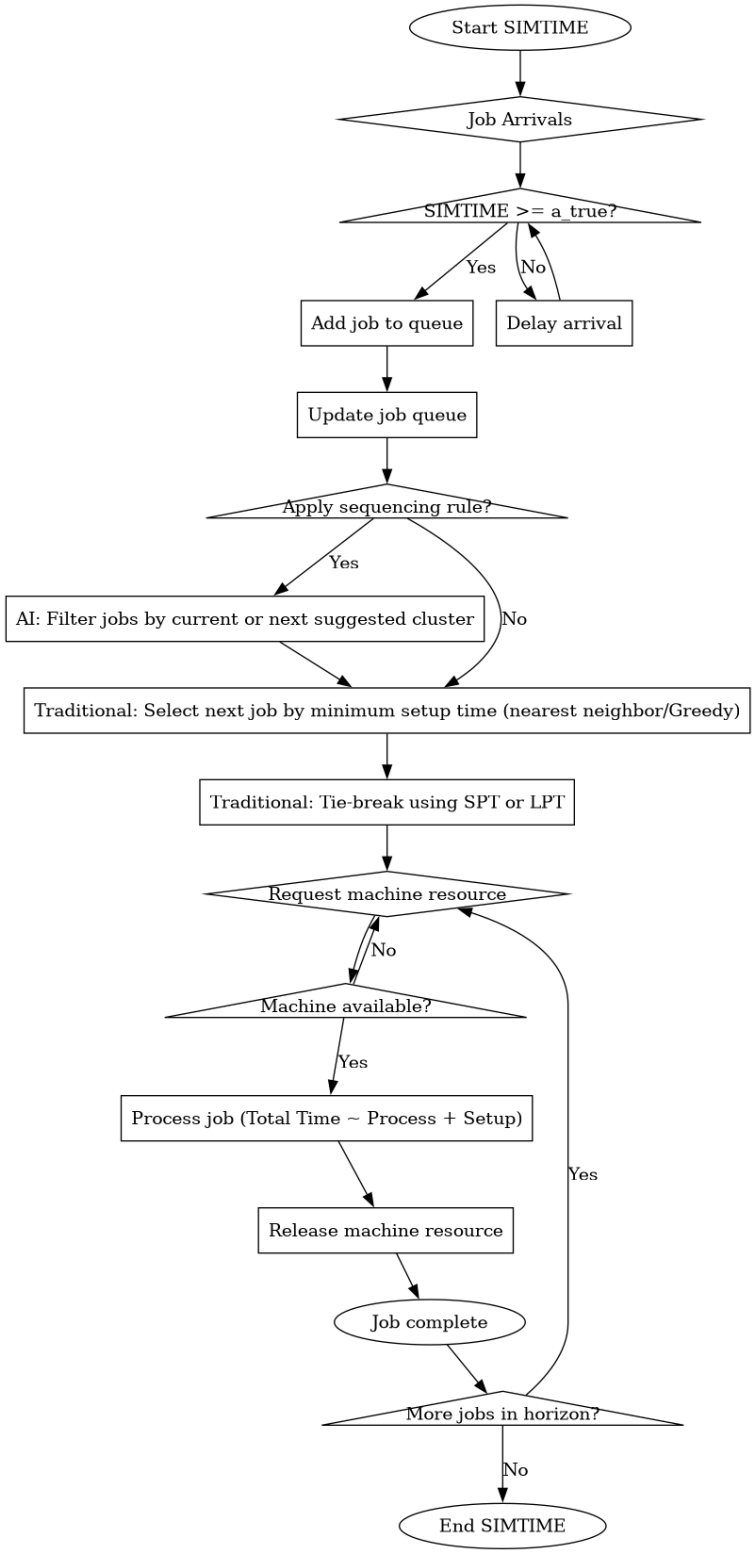
Discrete Event Simulation (DES) Model. In practice, the DES used here aligns with the simulations from DS 710 yet extends simulations with an added complexity: discrete events, queuing theory, and agent-entity-based modeling. However, ChatGPT was required for DES coding and debugging (OpenAI, 2024), translating all high-level scheduling strategies—from traditional rules or AI clustering—into explicit schedules. This method allowed this project to succeed because DES is outside the program's scope, and the library SimPy and its syntax were unacquainted (SimPy Developers, n.d.). A working DES ensured that the schedules generated could effectively warm up the start of later optimization phases, which was fundamental for improved results.

Figure 8 flow chart details the precise, high-level algorithm settled upon for the code of “DiscreteEventSimulations.py”: a script used to run all of the Traditional and AI algorithms (AI is used loosely here due to unsupervised learning of k-means clustering and a pivot discussed soon). Notably, the DES allowed the in-parallel, multiprocessing of separate sequencing rules (or none), saving additional time as the DES took in arguments from an external Jupyter Lab notebook, translating high-level rules into detailed and comparable scheduling outputs.

[13] Not even differential equations or closed-form "Jacksonian" network models were feasible alternatives (Solberg, 2009, p. 221). While they provide an aggregate understanding of random variables such as idle time proportions, average queue length, or average waiting times, they cannot produce the precise schedules required for warm starting downstream optimization solvers.

Figure 8

Flow Chart of Discrete Event Simulation Algorithm



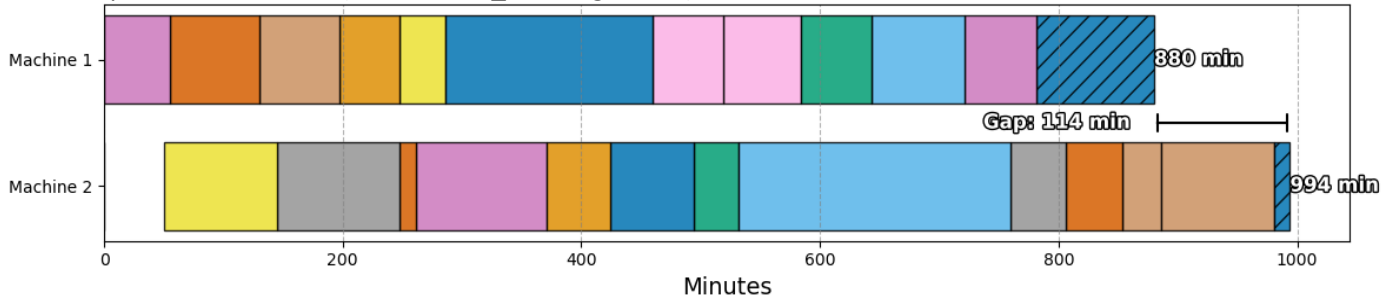
The highlights of this algorithm in Figure 8 are that each job is tracked in a shared queue, as well as the machine status (idle, processing) and the planned jobs that have yet to join the currently shared queue at the arrival minutes, a_{true}—while DES can simulate random distributions within, the project handled those externally; thus, DES ran a strictly deterministic unfolding of the jobs' true arrival times.

Traditional Rules. As discussed in Chapter I, traditional sequencing rules like Shortest Processing Time (SPT) and Longest Processing Time (LPT) play essential roles in scheduling. To implement these in the project's DES algorithm (Figure 8), they fit well after a nearest-neighbor selection method was used to decide the next job. When completing a sequence of identical recipes, a "greedy" approach was used in all DES runs to select the recipe with the minimum setup time. SPT or LPT was used for tie-breaking among jobs with the same recipe. In this project, the "Traditional_LPT" algorithm matched best, as it aligned with the same tie-breaking method used in later models (i.e., LPT); thus, complementing warm starts better for later models without violating constraints (Equation 19).

Before diving deeper into sequencing rules, Figure 9 illustrates a DES output using the traditional Longest Processing Time (LPT) rule. This figure shows DES's full schedule, including delays and parallel job processing with specific arrival times—a challenging feat in scheduling.

Figure 9

25-Job Sequence Gantt Chart ("Traditional_LPT" Algorithm)



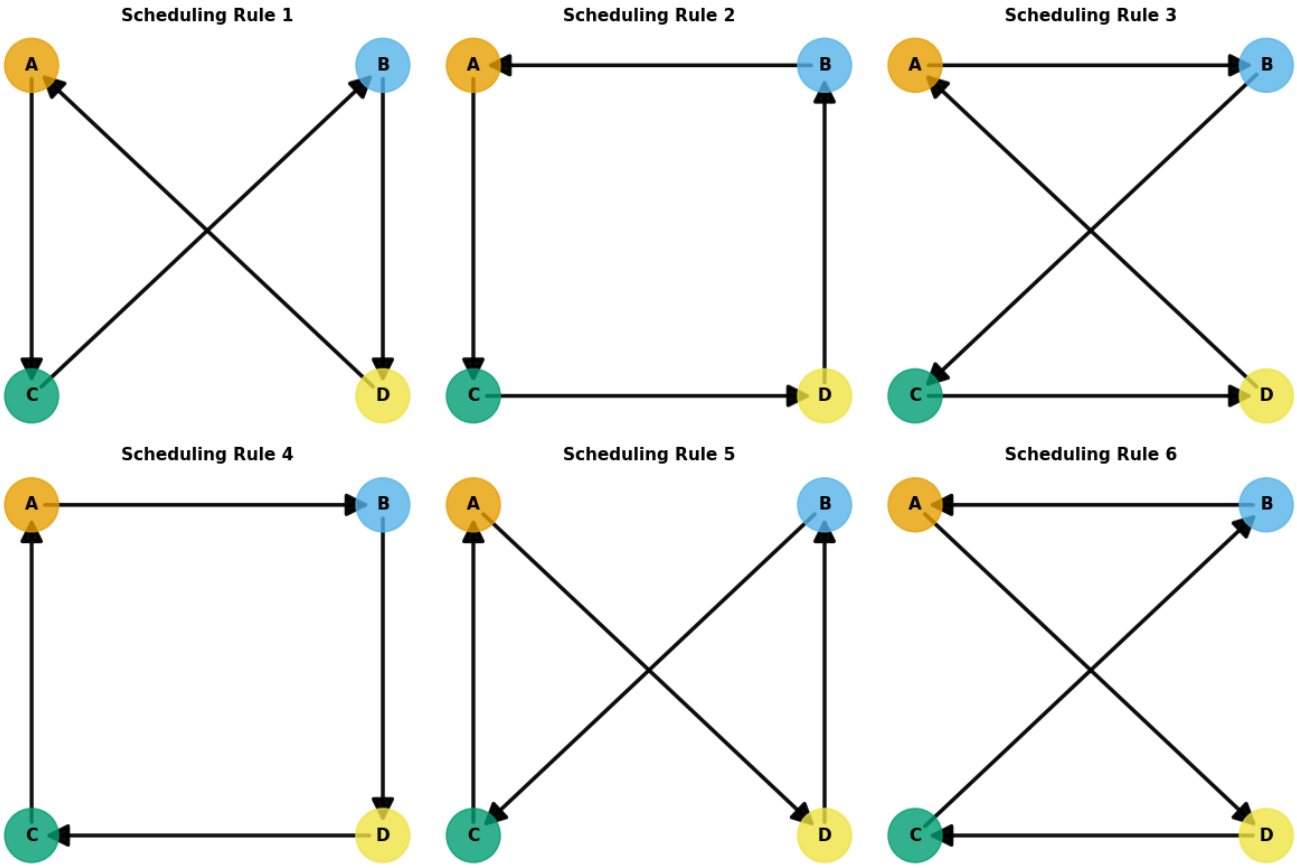
Note. The Gantt chart shows a maximum flow time of 994 minutes for Machine 2, including remaining delay minutes, with crosshatched sections marking each machine's route end. Although Machine 1's makespan is 880 minutes with no delays, delays and idle times—like those in Machine 2—can obscure the average makespan calculation (the other MILP objective), making it less evident in the Gantt chart alone.

Unsupervised Clustering Rule (AI). The project used unsupervised clustering, specifically k-means, to simplify the job sequencing problem by grouping jobs into clusters based on recipe temperature. This clustering reduced the problem's complexity and enabled computationally feasible brute force given the small k clusters of four. This method aligns well with the literature on mTSP and VRP problems, where clustering often aids in achieving optimal or near-optimal results faster, such as with Othman et al. and how they used clustering and parallelized mTSP into TSPs (2019), or Evreka company's use with clustering in capacitated VRP (2024).

As illustrated in Figure 10, the clustering of recipes—A, B, C, D—allowed for a maximum of $(N-1)! = 3! = 6$ unique sequences per machine, creating $6 \times 6 = 36$ possible scheduling rule combinations when applied across two machines. Thus, limiting the clusters to a tiny amount made brute-forcing the effectiveness of all scheduling rules feasible without the need for predictive AI models.

Figure 10

Unique Scheduling Sequences for Recipe Clusters



Initially, the project explored predictive AI methods, such as Decision Trees and other tree-based models, to predict effective sequence rules. However, brute-force enumeration ultimately proved to be a more practical and effective solution for scheduling, given that it had all the data. This led to a pivoted role for AI: instead of directly predicting schedules, the project used AI for foundational clustering and enhancing solver performance through Bayesian optimization. Studies by Hosny & Reda (2024) and Ishihara & Limmer (n.d.) confirmed that hyperparameter tuning in optimization solvers yields significant performance gains (40–60%), reinforcing AI's value in optimizing "black-box" solver configurations, such as the MILP and lookahead models.

Therefore, the project implemented a hybrid AI methodology. First, unsupervised clustering structured the decision space by pre-grouping similar jobs, enabling brute-force enumeration. Specifically, the project applied brute-force enumeration with the AI/clustering-informed scheduling algorithm, assigning each machine a schedule rule from a finite set, 1 through 6, as shown in Figure 10. This updated method[14] eased the efficient exploration of combinations without predictive AI, as the optimization process successfully managed sequencing complexities.

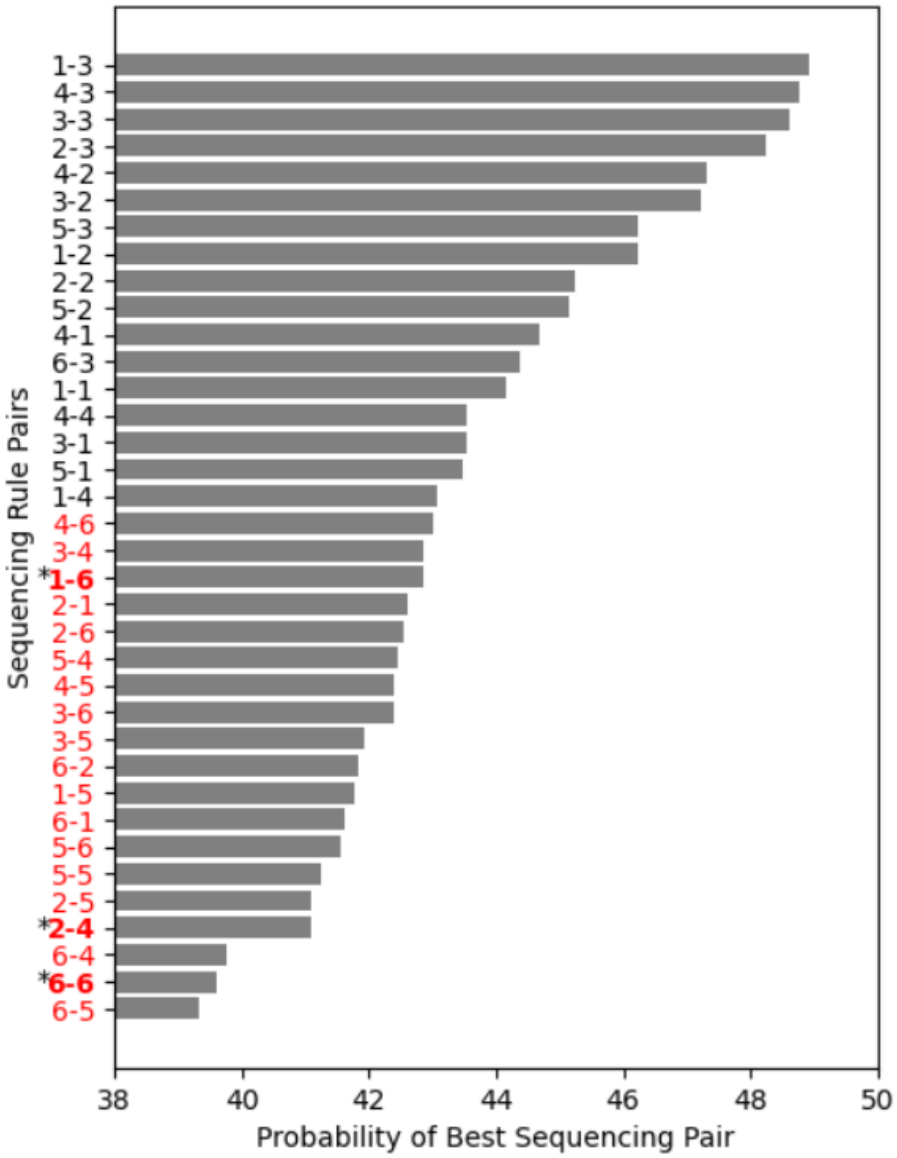
While this clustering informed brute-force scheduling, the project later used Bayesian optimization to fine-tune solver parameters, accelerating convergence—Chapter IV discussed the full details of the tuning process and results.

Figure 11 shows the scheduling performance impact of the sequencing rules from Figure 10, paired as $(\mathrm{Machine1} - \mathrm{Machine2})$. DES simulated each rule pair across a training problem set, and then the biobjective function (weighing average makespan and maximum flow time) was ranked. Pairs in red represent the bottom 50% in performance, which may have niche applications. However, asterisks mark "deadweight" rules that tied for best performance but never achieved it alone—these three pairs (1 – 6, 2 – 4, 6 – 6) were pruned from the DES brute-force pool, allowing computing to focus on the 33 beneficial rules.

[14] Although the project initially considered rule *preemptions*—interrupting the flow between recipe clusters—they added limited value, as the combined brute-force rules and solver optimization proved effective.

Figure 11

Performance of Machine Sequencing Pairs on Full Training Set

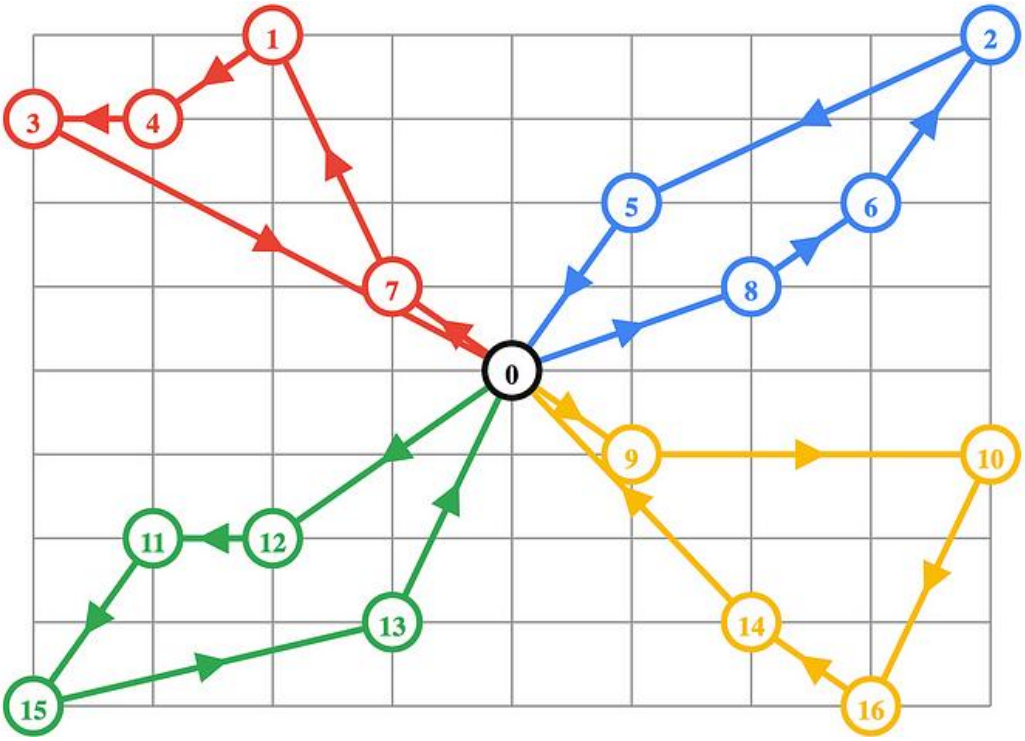


MILP Optimization Models

While the project's MILP models optimized objectives in a parallel processing job shop setting for manufacturers' benefit, Figure 12 illustrates a similar problem in 2D, assuming symmetric costs with four vehicles. It shows how these principles apply across industries—from Amazon deliveries to garbage collection. The goal of all these mTSP and VRP contexts is frequently to balance workloads and minimize costs, aligning with the project's focus on optimizing job sequencing across cooperative machines for better manufacturing profits.

Figure 12

Illustration of Optimal VRP Solution (Four Vehicles on 16 Nodes)



Note. Adapted example of an optimized VRP route of 16 size, where node "0" represents a single *depot* (or dummy origin node for this project); reproduced from R. G. Torres, S. Acharya, P. Goel, 2023, December 23, *Solving Capacitated Vehicle Rerouting Problem with GNNs*[15] (https://medium.com/gnns-for-cvrp/solving-capacitated-vehicle-rerouting-problem-with-gnns-88f1b90611ae).

[15] GNNs and RL AI, though well-suited to this VRP use case, were excluded from the project scope due to complexity and time constraints.

Similarly, this project adopted the mTSP/VRP methodology to tackle the two-machine scheduling problem, combining real-world constraints like recipe-specific processing times, setup durations, and arrival sequences.

The following sections drill down into the model's formulation, parameters, and constraints.

Ported MILP. The project's MILP model was built upon this established optimization method (VRP/mTSP) and customized for this specific operation scheduling problem. Fortunately, instead of developing the MILP model from scratch, the approach used the objective, constraints, and formulations from prior research in manufacturing operations scheduling.

As stated in Chapter II, key studies and foundational references provided the basis for structuring the MILP model; thus, the project included citations alongside the equations. While not all equations could be retraced to their source, the project spent considerable effort referencing related works that align with the model's structure.

The next sections detail the mathematical model's formulation, parameters, and constraints:

Sets:

- $\boldsymbol{J} = \{0, 1, 2, \dots, n\}$: be a set of jobs, where 0 represents a depot.
- $\boldsymbol{J_{1toN}} = \{1, 2, \dots, n\}$: be a set of jobs.
- $\boldsymbol{K} = \{1, 2, \dots, k\}$: be the set of machines.
- $i, j \in \boldsymbol{J}$**:** be default indices for jobs.
- $k \in \boldsymbol{K}$**:** be the default index for machines.
- $\boldsymbol{C}$: be the set of capable, runnable jobs.
- $\boldsymbol{S}$: be the set of pairs of jobs for enforcing Longest Processing Time tie-breaking.

Parameters:

- c_{ijk}: Cost (or time) from job i to job j on machine k.
- a_i: Earliest start time for job i (hard constraint).
- b_i: Latest finish time for job i (soft constraint).
- M: A large constant (Big M method).
- $w_1 \in [0,1]$: Weight for the first objective in the biobjective function.

Decision Variables:

- $x_{ijk} \in \{0,1\}$: Binary variable, 1 if machine k processes from job i to job j, 0 otherwise.
- $u_{ik} \in \{0,1\}$: Binary variable, 1 if job i assigns to machine k, 0 otherwise.
- $t_i \geq 0$: Real variable, start time of job i.
- $t_{ik} \geq 0$: Real variable, start time of job i on machine k (only relevant if $u_{ik} = 1$); extends beyond the last job to capture final flow time: $t_{ik} \in \{0, 1, 2, \dots, n+1\}$.

Objective:

- Minimize the weighted sum of the max flow time and average makespans (Azadeh et al., 2016):

$$\text{Minimize} \quad w_1 y_1 + (1 - w_1) y_2. \qquad \text{(Biobjective) (1)}$$

Where:

$$y_1 \geq t_{(n+1)k} \quad (\text{for } k), \qquad \text{(Max Flow Time) (2)}$$

$$y_2 = \frac{1}{k} \sum_{\substack{i,j,k \\ i \neq \text{j}}} c_{ijk} x_{ijk} \quad (for\ i, j, k \in \boldsymbol{C}),$$

$$\text{(Average Makespan) (3)}$$

In Equation 2, minimizing a max flow time had to be linearized (as max functions are nonlinear); as such, a real variable y_1 was added to track this max variable across k machines.

In Equation 3, minimizing a makespan is seen in nearly every TSP model (Dantzig et al., 1954; Desrochers & Laporte, 1991; Pferschy & Staněk, 2017).

Subject to:

$$t_i - c_{0ik}x_{0ik} \geq 0 \quad (\text{for } i \in \boldsymbol{J_{1toN}};\ k), \qquad \text{(Closed-Loop Start) (4)}$$

$$t_i + c_{i0k}x_{i0k} \leq t_{nk} \quad (\text{for } i \in \boldsymbol{J_{1toN}};\ k), \qquad \text{(Closed-Loop End) (5)}$$

Equations 4 and 5 close the loop of the starting and ending times possible for the routings.

$$t_{ik} - t_{jk} + \left(b_i - a_j + c_{ijk}\right)x_{ijk} \leq b_i - a_j \quad (\text{for } i \neq j;\ i,j \in \boldsymbol{J_{1toN}}; k), \text{ (Subtour Elimination) (6)}$$

Equation 6 is complex due to the unintuitive nature of subtours: disjointed tours (e.g., jumping from 1–2 to 3–4) create infeasible solutions without this constraint. The closest reference is Desrochers & Laporte (1991); though they used integer modeling, this project's modeling uses all binary variables.

$$x_{ijk} = 0 \quad \left(\text{for if } c_{ijk} = M; i; j; k\right), \qquad \text{(Incapable Arcs) (7)}$$

Equation 7 creates a method to speed up the solutions by automatically ruling out the impossible arcs (e.g., incapable recipes).

$$x_{0jk} = \begin{cases} 1, \text{if } j = k \\ 0, \text{if } j \neq k \end{cases} \quad (\text{for } j; k), \qquad \text{(First Job Assignment) (8)}$$

$$\sum_i x_{i0k} = 1 \quad (\text{for } k), \qquad \text{(Last Job Assignment) (9)}$$

Equation 8 is a simple way to ensure consistent starting recipe information by having each machine start on its job by the machine number.

Equation 9 was standard in most mTSP/VRP formulations (Othman et al., 2019) to have multiple salespersons or vehicles return home to the single depot, 0.

$$\sum_{i,k} x_{ijk} = 1 \quad (\text{for } j \neq i;\ j \in \boldsymbol{J_{1toN}}), \qquad \text{(Outgoing Assignments) (10)}$$

$$\sum_{i,k} x_{jik} = 1 \quad (\text{for } j \neq i; j \in \boldsymbol{J_{1toN}}), \qquad \text{(Incoming Assignments) (11)}$$

Similarly, Equations 10 and 11 ensure that each non-depot job is visited exactly once, with one outgoing and one incoming route, maintaining flow continuity across all rows (outgoing) and columns (incoming) (Othman et al., 2019).

$$x_{iik} = 0 \quad (\text{for } i \in \boldsymbol{J_{1toN}}; k), \qquad \text{(Self-Loop Elimination) (12)}$$

Equation 12 guarantees that a job cannot return to itself; that would be impossible.

$$a_i \leq t_i \leq b_i \quad (\text{for } i \in \boldsymbol{J_{1toN}}), \qquad \text{(Time Windows) (13)}$$

Equation 13 restrains the start of each job to within the time windows, or it is infeasible; the arrival times are *hard*, while the due times are *soft* constraints[16].

$$\sum_j x_{ijk} = u_{ik} \quad (\text{for } i \neq j;\ i, k),$$

(Linking Variables x and u) (14)

$$t_0 = 0, \qquad \text{(Initial Depot Time) (15)}$$

$$t_{ik} \leq t_i \quad (\text{for } i; k), \qquad \text{(Time Linking) (16)}$$

$$t_{ik} \geq t_i - M(1 - u_{ik}) \quad (\text{for } i; k), \text{ (Big M Time Linking) (17a)}$$

$$t_{ik} \leq M u_{ik} \quad (\text{for } i; k), \qquad \text{(Big M Time Linking) (17b)}$$

Equations 15, 16, 17a, and 17b establish time relationships using the Big M method to create either-or constraints that link job start times with machine start times (Hillier & Lieberman, 2015, pp. 483–484). Machines that do not initiate a job are assigned a start time of 0, while those that do allocate a nonzero start time.

$$t_{ik} \cdot \mathrm{u_{jk}} \leq t_{jk} \quad (\text{for } (i, j) \in \boldsymbol{S}; k), \text{ (Symmetry-Breaking) (18)}$$

[16] Hard constraints are non-negotiable and must always be satisfied. Soft constraints, however, allow flexibility and may be adjusted to support solvability, providing structure without rigid enforcement.

Finally, Equation 18 introduces a *quadratic constraint*[17] for pre-determined job pairs, enforcing tie-breaking by the Longest Processing Time (LPT) tradition. *Symmetry-breaking constraints*, commonly used in MILP to enhance solver efficiency (Brandinu & Trautmann, 2014), prune the solution space by eliminating redundant solutions in identical, equivalent feasible solutions. This quadratic constraint was set as a hyperparameter, allowing Bayesian optimization to activate it strategically.

While this MILP model is intricate, its design leverages only binary and real variables, excluding general integers (e.g., 2, 3, 4 …), significantly improving computational efficiency. This is impressive, as most MILP VRP or mTSP models in the literature rely on non-binary integer variables, which can substantially slow solving. Operations research literature underscores that binary integer programming is more desirable to solve than general integer programming (Hillier & Lieberman, 2015, pp. 497–501;531–532), with advanced methods and solvers available (Gurobi Optimization, 2024).

However, porting the model from CMPL to Python's Pyomo posed substantial challenges. The original Excel-based model was structured in 2D with explicit k machine slices and needed to be abstracted into a 3D dimension to meet project format needs. This conversion required careful reconstruction of constraints and variables and extensive unit testing to ensure accuracy. While ChatGPT provided support for overall code debugging (OpenAI, 2024), it was ineffective in ensuring accurate mathematical constraints, which demanded meticulous comparison and tweaks.

The final optimization code class, "VRPSolver", established a modular foundation, supporting further expansions with stochastic programming. This object-oriented approach minimized code redundancy and enhanced flexibility for sequential optimization.

[17] Although u is binary and t is real, the constraint is *quadratic* or nonlinear due to the product of two variables. Solvers like Gurobi can often linearize such terms automatically. While this adds complexity, it also offers potential benefits, so the constraint was kept as an adjustable hyperparameter.

Sequential Policy. Inheriting the ported MILP model, the next phase involved implementing s*equential policies* (Powell, n.d.)—a method that optimizes decision policy π iteratively over time, with each iteration n corresponding to a system state with a new job entering the current queue step. The project dynamically simulated the decision policy for each problem, where the next job's arrival triggered a new optimization run.

Furthermore, the methodology of simulating model performance across the job queueing evolution in time mirrored the discrete event simulation (DES) approach to accurately score how the model adapted to arrival times as they happened. This simulated scoring method allowed for an "apples-to-apples" comparison with the traditional and clustering models, providing a fair and consistent framework to test each model's effectiveness in an imperfect information queueing system.

The formulation follows below:

Parameters:

- a: Estimated arrival times used in the initial planning.
- a_{true}: Actual arrival times are revealed progressively, defining each $\text{SIMTIME}^{(n)}$.
- $\text{SIMTIME}^{(n)}$: Current simulation time at iteration n, set by the actual arrival time $a_{\text{true}}^{(n)}$. Each nonzero a_{true} value determines a new n iteration of a job arriving at queue step 0.
- $a^{(n)}$: Updated arrival times at iteration n, defined as: $a^{(n)} = \begin{cases} a_{\text{true}}, & \text{if SIMTIME}^{(n)} \geq a_{\text{true}} \\ a, & \text{otherwise.} \end{cases}$

Previous Solution Variables:

- t_{ik}^{prev}: Start time of job i on machine k from the previous solution.
- x_{ijk}^{prev}: Previous solution's value for x_{ijk}.

$$x_{ijk} = x_{ijk}^{\text{prev}} \quad \left(\text{for } (i,j,k) \text{ such that } t_{ik}^{\text{prev}} \leq \text{SIMTIME}^{(n)}\right)$$

(Sequential Policies Constraint) (19)

The sequential policies approach breaks down the main scheduling problem into snapshots of different realizations as the system queueing occurs. In Equation 19, each iteration moves forward with new constraints based on unique, nonzero arrival times $a_{\text{true}}^{(n)}$, with the total number of iterations n determined by how many unique arrival times there are. So, later iterations carry prior decision policies forward to a final decision policy, $\pi^{(N=\text{final})}$. This objective value can be compared fairly across algorithms. These final solutions are how the policy performed over time, iteratively:

- The iterative solving algorithm progressively updates the simulation time $\text{SIMTIME}^{(n)}$ and adjusts cumulative constraints $x_{ijk}^{\text{cumulative}}$. Each iteration refines the decision policy π based on the updated parameters and previous solutions: $\pi^{(n+1)} =$ $\text{Solve min Objective, subject to updated constraints and } x_{ijk}^{\text{cumulative}}$.
- An alternative form of Equation 19, the x_{ijk}^{N} fixed assignments at the N-th iteration cumulatively union the decisions made across all past iterations in time:

$$x_{ijk}^{\text{N}} = \bigcup_{n=1}^{N} x_{ijk}^{(n)} \quad \left(\text{for } (i,j,k) \text{ such that } t_{ik}^{\text{prev}} \leq \text{SIMTIME}^{(n)}\right)$$

(Sequential Solving) (19b)

This “live” or online approach (Equation 19b) for the MILP models generates not just one schedule but all step-by-step schedules, representing a policy adapting as information updates—namely, the true arrival times being learned. Therefore, the objective captures not just one optimization but the accumulated effect of sequential optimizations in a post-mortem evaluation, $\pi^{(N=\text{final})}$, providing a complete view of how the 10-second optimization policy did over time. This setup crucially enables fair, apples-to-apples comparisons with traditional or clustered methods, showing how each policy handled uncertain job arrivals truthfully. The results are thus more substantial, revealing each scheduling policy's performance and effectiveness in genuine comparisons.

Stochastic Optimization Model. The next logical step was incorporating stochastic programming, building on the existing sequential policy approach. Although the project reviewed advanced methods like sample average approximation, importance sampling, and large-scale stochastic programs, they exceeded the program's technical scope (Infanger, 1994, pp. xi–xiii; Infanger, 2011, p. viii). Instead, a more straightforward lookahead stochastic approach was adopted, drawing inspiration from parameterized sequential optimization techniques (Ghadimi & Powell, 2022; Powell, n.d.).

This project's sequential optimization method addressed uncertainty by concentrating on short-term predictions of the next job arrival in the queue. The model grouped high-level scenarios based on recipe clusters and worst-case arrival timings (i.e., using the longest processing time within a cluster). The initial scheduling decision matrix remained unchanged, keeping the model's complexity manageable and enabling adaptive scheduling decisions without excessive intricacy.

The following is the lookahead model methodology:

Sets:

- $\boldsymbol{W} = \{w_1, w_2 \dots w_m\}$: A generalized set of m scenarios, i.e., recipe clusters (A, B, C, D) present in the next queue step. This set allows for discontinuities and non-sequential elements.
- $m = \max(Unique\ Clusters\ in\ Next\ Queue\ Step, 1)$. The maximum number of unique clusters in the next queue step ensures that at least one scenario is always present. If only one scenario is present, a deterministic model is used instead.
- $w \in \boldsymbol{W}$: Index for scenarios.

Parameters:

- p_w: Probability of scenario w, calculated as $p_w = \frac{\mathrm{Count}_w}{\mathrm{TotalCount}_w}$, where Count_w is the number of jobs in recipe cluster w (e.g., A, B, C, D) from the incoming queue of $\mathrm{TotalCount}_w$ jobs.

- a_{iw}: Earliest start time for job i at scenario w (hard constraint), the adjustment accounts for a worst-case job's possible earliness (by 30 minutes), the current iteration's snapshot in time ($\mathrm{SIMTIME}^{(n)}$), and a minimum time difference of 5 minutes. Then, the scenarios are adjusted as follows: $a_{iw} = \begin{cases} a_i, & \text{if } m = 1, \\ \max(a_i - 30, \mathrm{SIMTIME}^{(n)} + 5), & \text{if } m > 1 \text{ (for } w). \end{cases}$
- Here:
    - For $m = 1$ (deterministic snapshots), there is no need to consider scenarios.
    - For $m > 1$ (stochastic snapshots), the arrival time a_i is altered but not too early or violating the minimum difference in time, and the job i with the longest processing time (LPT) within the cluster is adjusted based on average processing times (using mean or nanmean hyperparameter for bigM handling).
- b_{iw}: Latest finish time for job i at scenario w (soft constraint), calculated similarly to a_{iw} above.

Decision Variables:

- $x_{ijk} \in \{0,1\}$: Binary variable, 1 if machine k processes from job i to job j, 0 otherwise (unchanged).
- $u_{ik} \in \{0,1\}$: Binary variable, 1 if job i assigns to machine k, 0 otherwise (unchanged).
- $t_{iw} \geq 0$: Real variable, start time of job i in scenario w.
- $t_{ikw} \geq 0$: Real variable, start time of job i on machine k in scenario w; extends beyond the last job to capture final flow time: $t_{ikw} \in \{0, 1, 2, \dots, n+1\}$.
- $y_{1w} \geq 0$: Max flow time in scenario w.

Objective:

- Minimize the weighted sum of the max flow time and average makespans:

$$\text{Minimize} \quad w_1 \sum_w p_w y_{1w} + (1 - w_1) y_2$$

(Expected Biobjective) (1s)

Where:

$$y_{1w} \geq t_{(n+1)kw} \quad (\text{for } k; w), \qquad \text{(Max Flow Time) (2s)}$$

$$y_2 = \frac{1}{k} \sum_{\substack{i,j,k \\ i \neq \text{j}}} c_{ijk} x_{ijk} \quad (\text{for } i, j, k \in \mathbf{C}),$$

(Average Makespan) (3)

Subject to:

$$t_{iw} - c_{0ik} x_{0ik} \geq 0 \quad (\text{for } i \in \boldsymbol{J_{1toN}}; k; w), \qquad \text{(Closed-Loop Start) (4s)}$$

$$t_{iw} + c_{i0k} x_{i0k} \leq t_{nkw} \quad (\text{for } i \in \boldsymbol{J_{1toN}}; k; w), \qquad \text{(Closed-Loop End)(5s)}$$

$$t_{ikw} - t_{jkw} + \left(b_{iw} - a_{jw} + c_{ijk}\right) x_{ijk} \leq b_{iw} - a_{jw} \quad (\text{for } i \neq j;\ i, j \in \boldsymbol{J_{1toN}}; k; w),$$

(Subtour Elimination) (6s)

$$x_{ijk} = 0 \quad \left(\text{for if } c_{ijk} = M; i; j; k\right), \qquad \text{(Incapable Arcs) (7)}$$

$$x_{0jk} = \begin{cases} 1, \text{if } j = k \\ 0, \text{if } j \neq k \end{cases} \quad (\text{for } j; k), \qquad \text{(First Job Assignment) (8)}$$

$$\sum_i x_{i0k} = 1 \quad (\text{for } k),$$

(Last Job Assignment) (9)

$$\sum_{i,k} x_{ijk} = 1 \quad (\text{for } j \neq i; j \in \boldsymbol{J_{1toN}}),$$

(Outgoing Assignments) (10)

$$\sum_{i,k} x_{jik} = 1 \quad (\text{for } j \neq i; j \in \boldsymbol{J_{1toN}}),$$

(Incoming Assignments) (11)

$$x_{iik} = 0 \quad (\text{for } i \in \boldsymbol{J_{1toN}}, k), \qquad \text{(Self-Loop Elimination) (12)}$$

$$a_{iw} \leq t_{iw} \leq b_{iw} \quad (\text{for } i \in \boldsymbol{J_{1toN}}; w), \quad \text{(Time Windows) (13s)}$$

$$\sum_j x_{ijk} = u_{ik} \quad (\text{for } i \neq j; i;\ k),$$

(Linking Variables x and u) (14)

$$t_{0w} = 0 \quad (\text{for } w), \qquad \text{(Initial Depot Time) (15s)}$$

$$t_{ikw} \leq t_{iw} \quad (\text{for } i; k; w), \qquad \text{(Time Linking) (16s)}$$

$$t_{ikw} \geq t_{iw} - M(1 - u_{ik}) \quad (\text{for } i; k;\ w), \quad \text{(Big M Time Linking) (17as)}$$

$$t_{ikw} \leq M u_{ik} \quad (\text{for } i; k; w), \quad \text{(Big M Time Linking) (17bs)}$$

$$t_{ikw} \cdot \mathrm{u_{jk}} \leq t_{jkw} \quad (\text{for } (i,j) \in \mathbf{S}; k; w), \text{(Symmetry-Breaking) (18s)}$$

$$x_{ijk} = x_{ijk}^{\mathrm{prev}} \quad \left(\text{for } (i,j,k) \text{ such that } \max_{\mathrm{w} \in \mathbf{W}} \left(t_{ikw}^{\mathrm{prev}}\right) \leq \mathrm{SIMTIME}^{(n)}\right)$$

(Sequential Policies Constraint) (19s)

Although Equations 3, 7–12, and 14 remain unchanged from the original model, the remaining equations add an extra *w* scenario dimension to the time variables, with t_i and t_{ik} now expanded to t_{iw} and t_{ikw}, respectively (similarly done to the time windows). The objective equation, initially Equation 1, is now displayed as Equation 1s (i.e., s for stochastic), blending an expected value term based on the probability of each representative scenario. The average makespan does not include stochastic terms.

Equation 19s shows that the project took the maximum arrival times of all the scenarios to manage sequential policy assignments.

This lookahead extension captures robust optimization elements by evaluating “what-if” scenarios for recipe clusters. Each scenario independently represents the potential for a high-workload job within a cluster to arrive first, using LPT to symbolize worst-case arrival scenarios among clusters.

Model Evaluation and Performance Metrics

This section outlines the basis for evaluating the project's models. By focusing on a biobjective approach—minimizing average makespan and max flow time, the project measured model performance against baseline methods. Table 4 summarizes the evaluation design, objectives, and metrics.

Table 4

Methodology of Evaluating Traditional Versus Integrated Models

Evaluation design	Training dataset	Evaluation (holdout) dataset
Primary objective	Achieve biobjective improvement over baseline scheduling methods through optimized models	Same as the training set
Problem sets (job counts)	7,500 jobs (25-queue: 300, 75-queue: 100)	Similar but unique problems specific to holdout test
Models	Traditional_true_SPT (for baseline using true arrivals), AI_LPT (for warm starts), Sequential_Optuna, Lookahead_Optuna	Same as the training set, but also an AI_true_LPT (using true arrival times)
Hyperparameter tuning	Optuna hyperparameter optimization applied to tuning models	None; for model training, only
Metrics evaluated	Primary: Biobjective (Relative to Traditional_SPT), runtime (for tuning)	Primary: Biobjective improvement; Supplementary: Runtime
Performance metric	Percentage improvement in biobjective relative to Traditional_true_SPT	Same as the training set

Note. Optuna tuning focused on hyperparameters, such as solver-specific settings (e.g., MIPFocus as "1", "2", or "3"). In the training phase, these parameters were tuned in parallel within problem sets, with differing tuning results across problem sets.

Evaluation Design

As seen in Table 4, the project split the data into training and holdout datasets to carefully train and evaluate across different problem scales and models:

1. *Training dataset*: Used for tuning hyperparameters and optimizing model performance, this set included 7,500 job instances replicated at varying queue sizes (25 and 75 jobs). Optuna was applied to tune parameters for Sequential_Optuna and Lookahead_Optuna, focusing on

improving relative to Traditional_true_SPT (based on true arrivals). The training metrics included the biobjective value and runtime, guaranteeing that each pre-trained optimization model is solved in under five minutes.

2. *Holdout dataset*: The holdout set, reserved for validation, consisted of unique instances in queue sizes of 25 and 75 jobs, allowing an unbiased comparison of final model configurations. The biobjective performance remained the primary metric.

Performance Evaluation Metrics

The metric used in both training and holdout evaluations included (See Table 4):

- *Relative biobjective improvement*: The primary metric compares each model's combined performance in average makespan and max flow time against Traditional_true_SPT. This relative percentage-based link ensured a consistent standard across models and problem sizes and guaranteed that the project directly observed relative objective improvements.

Conclusion

Chapter III detailed the manner for developing sophisticated scheduling models within a synthetic manufacturing environment; each intended to achieve the project's objectives. The process began with generating synthetic data to create realistic problem sets, mirroring job shop conditions, recipe clustering, machine compatibility, and a firm foundation for fair and consistent model evaluation. With this data, a Discrete Event Simulation (DES) produced schedules using traditional sequencing rules (Traditional_true_SPT) for evaluation metrics and an AI clustering rule (AI_LPT model) for optimized warm starts.

The methodology advanced with a mixed integer linear programming (MILP) model ported from prior efforts to enhance scheduling efficiency. This MILP was extended into a sequential policy

framework, enabling fairness in comparing algorithm performances over time. Finally, a stochastic lookahead extension introduced scenario-based arrival times to account for scheduling uncertainty.

Each methodology step in this chapter supported the next, building a comprehensive evaluation basis that balanced optimization speed with quality. Sequential hyperparameter tuning with Optuna on training problem sets prepared the advanced models (Sequential_Optuna and Lookahead_Optuna) for holdout testing, simulating real-world applications in a “production” environment. Chapter IV presents the results, evaluating each model’s effectiveness in meeting the project’s goals to make a significantly better scheduler than the traditional algorithm.

Chapter IV: Results

The following chapter reports the results of blending AI with optimization and stochastic modeling. The project's original objectives provide context for this chapter:

- Compared to a traditional scheduling method (AI SPT), achieve a relative improvement of 10%, 20%, and 30% for AI, deterministic optimization, and stochastic optimization.
- Develop a scalable and real-time algorithm solving within five minutes and a 5% optimality gap for 99.9% of 25-job problems, 99% of 50-job problems, and 95% of 75-job problems.

Due to the obscured difficulty of objectively comparing optimizations by "realized" schedules over time—and the project's strict timeline—the initial objective of five-minute solves had to be even quicker, a mere 10 seconds. This is because unknown arrival times become known sequentially in reality; therefore, a single schedule optimization was incomplete. Instead, to evaluate an honest objective value, the optimization had to schedule not just the first state—the current snapshot (10 seconds)—but also notching forward with past decisions and scheduling sequentially: the next job's true arrival (20 seconds), the next job's true arrival (30 seconds), and so forth until it reached the end state with all arrival times known (e.g., 18 iterations was 180 seconds or 3 minutes). Thus, the revised time objective (10 seconds) became a fixed hyperparameter constant, representing a practical real-time policy. Consequentially, the 50-job problems were descoped in the training and evaluation to ensure project timeline achievability.

Similarly, the "5% optimality gaps" challenged achievability and whether it added value to this business case study. Although solving each problem to its optimal conclusion would have had some value, such as measuring the missed opportunity, optimally solving 400 unique problems (with 100 in the 75-job queue) would have been exceptionally computationally expensive. Optimal solves were observed in development in only the simplest cases (e.g., 10-job queues solved optimally within two

seconds). Moreover, while optimally solving problems could have still been forced, the business case study's other objectives concentrated on the relative improvements over a traditional baseline rather than a purported missed “best possible opportunity”. In other words, better feasible (i.e., primal) solutions mattered more than approaching the best possible theoretically (i.e., dual). Thus, the project’s scalability objective was revised into a real-time optimization that raced to schedule 25-job and peak demand 75-job problems successfully within 10 seconds, outperforming the traditional way.

The results evaluated how successfully the project had integrated AI and optimization and were presented in the following order:

1. *Optimization algorithms tuning*: Bayesian hyperparameter tuning, trailed by validation.
2. *All algorithms testing*: Comprehensive holdout testing.

Hyperparameter Tuning With AI

As discussed in Chapter III, achieving the objectives in a real-time, scalable way benefitted from careful hyperparameter tuning of the Pyomo optimization solver shell, Gurobi. In other words, the ideal solver configurations should be learned before evaluating optimization models on a holdout test (similar out-of-sample problems).

Rather than random search, grid search, and so on, Bayesian optimization was an obvious choice for hyperparameter tuning because of lengthy runtimes (i.e., 2 – 4 minutes apiece), a small search space of less than 20 parameters (i.e., 6 – 8), and an available methodology slot for AI (i.e., after the initial predictive AI classification exploration). Bayesian optimization models learn hyperparameter relationships for each trial and where to search next. Crucially, they can work well with “black-box” objective evaluation, making them a perfect fit for this project. Furthermore, after comparing open-source Python libraries for black-box Bayesian optimization, HyperOpt and Optuna appeared the most

frequently. The latter was selected for user-friendliness and its top-end features[18]. Some feature examples included how Optuna displayed visualizations and learned dependencies in hyperparameters (i.e., multivariate correlations or multicollinearity).

As a result, loading training problems only and using Optuna (Optuna, n.d.), the project conducted hyperparameter searches. The hyperparameter search used Optuna's *Tree-structured Parzen Estimator* (TPE), a tree-based Bayesian optimization method for sampling new parameters. Each two-trial cycle started with a single random search to provide exploration before switching to TPE for its ability to balance optimization exploration and exploitation. Key parameters tuned included:

- *Delta start*: padding the soft constraint of a time window to avoid infeasibility (i.e., 0 to 3000 minutes)
- *Heuristics*: settings to balance solution quality or speed (i.e., 0 to 100%)
- *Symmetry-breaking constraints*: user-created toggle to limit redundancy in optimization models with an LPT tie-breaking constraint (i.e., True, False)
- *MIP focus*: setting to fine-tune goals for optimization (i.e., 0=auto, 1=optimality, 2=feasibility, 3=balance)

After that, Optuna was trailed by a small validation among the three top parameters and optimization models on ten sampled training problems for each queue size (25 and 75). This validation step tested the top configurations on ten consistent training problem samples per queue size, verifying the chosen parameters' adaptability across diverse problems. The project ran the deterministic and stochastic solver tunings separately and explored their hyperparameter spaces in that order, which allowed learning things in a sequentially applied way.

[18] The following blog helped inform the project by fairly comparing HyperOpt and Optuna (Czakon, 2023).

Compared to the deterministic solver, the stochastic one included an added hyperparameter, influencing how new scenarios were created. These scenarios represented different random possibilities based on the potential next arriving job's recipe cluster. They used a maximum processing time to model the worst-case job to arrive next per cluster (in LPT fashion). Specifically, a `nanmean` boolean hyperparameter determined whether processing times of the incapable recipe machines, represented by "big M" values (e.g., 10,000), were treated as missing (NaN) or included as 10,000 in averaging. If set to True, "big M" values were ignored; otherwise, if set to False, the scenarios penalized incapable jobs. This choice impacted how the project made the possible scenarios, which, in turn, influenced the effectiveness of the stochastic solver in reaching a better objective value. Thus, the hyperparameter, `nanmean`, allowed the solver to explore two approaches for modeling uncertainty in scheduling.

Adding Bayesian optimization to the mix with the above methodology ensured that the optimization models had a better chance of achieving the project objectives for the business case study.

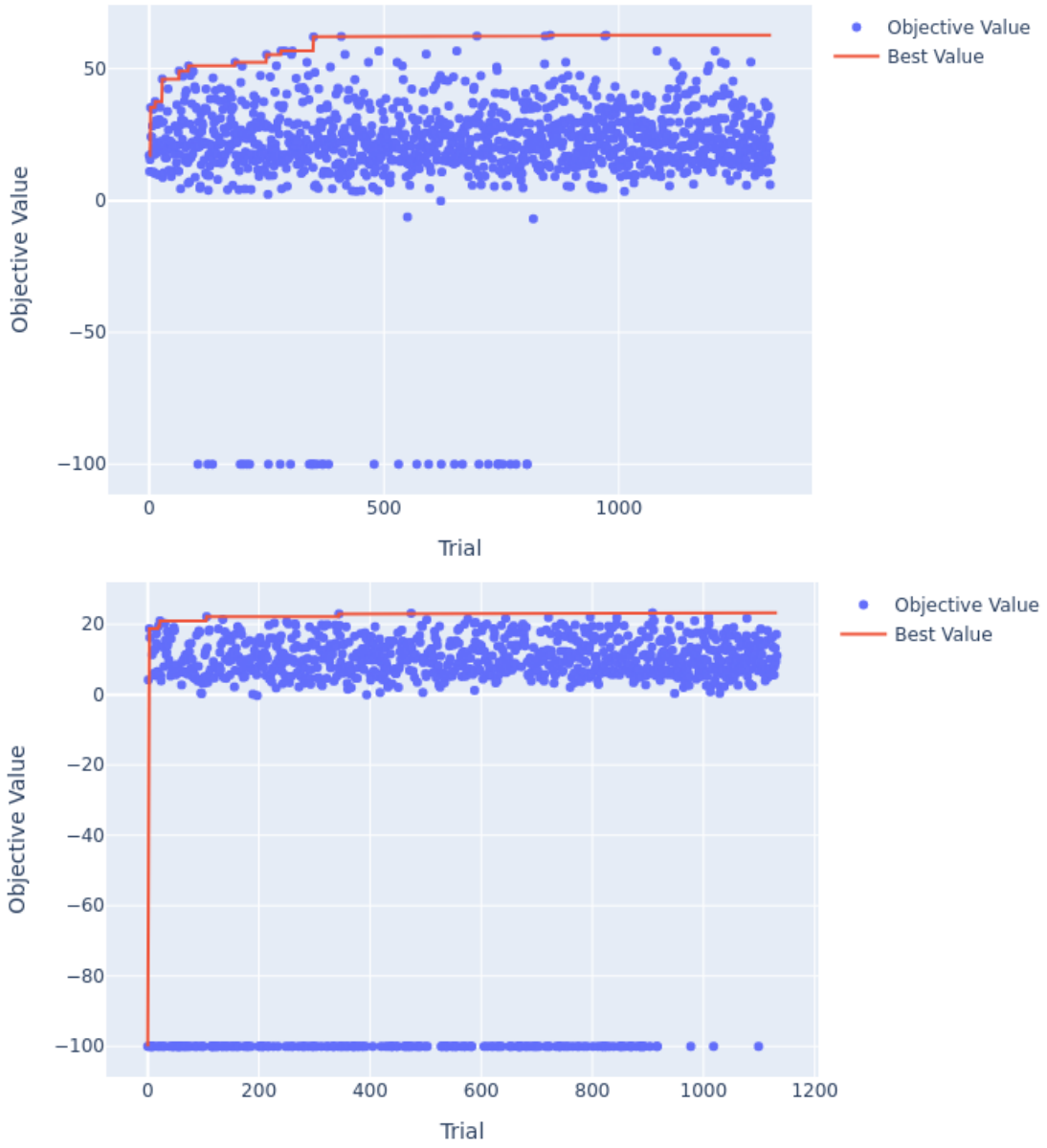
Deterministic Optimization Solver Pre-training

Optuna focuses on the hyperparameter interactions (rather than the objective function's inner workings), making it perfect for this project's hyperparameter tuning. After using Optuna, the project loaded each study's outputs and validated the best three parameters for evaluation.

Bayesian Optimization Tuning. Figure 13 shows the Optuna studies chronologically trialing a new set of parameters by sampling either random search or TPE using provided bounds.

Figure 13

Deterministic Optimization: Bayesian Optimization (Optuna) for 25 and 75-Queue (N=2458)



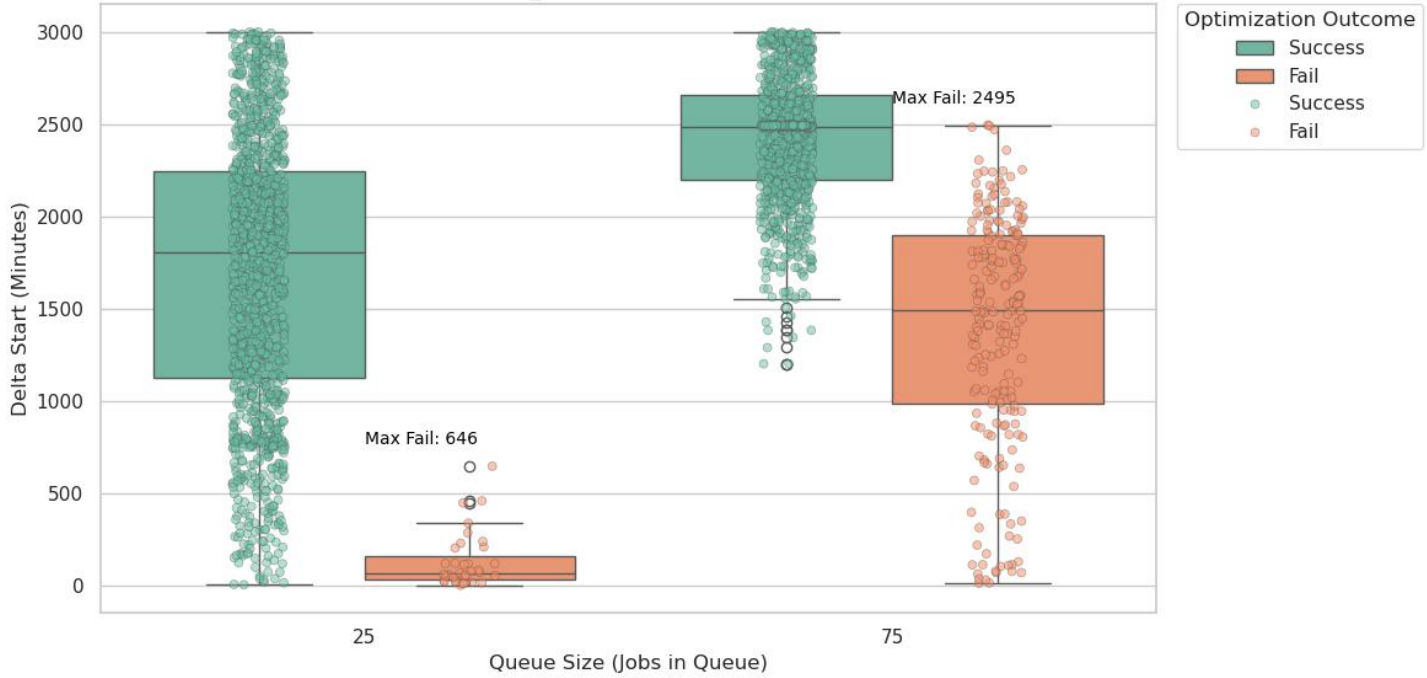
Note. "-100%" values show failed runs (infeasible) due to soft constraint violations on time windows (e.g., [a, b) with b slack), eased by padding the upper bound using a `delta_start` hyperparameter.

Figure 13 shows the "Best Value" progression for objective values ("relative objective improvement %" compared to baseline traditional SPT), moving up over time with learning. While sampling random search explored more space and stippled the chart everywhere, TPE maximized the

noisy objective—noisy because of how unique each problem characteristic adjusted the optimization space. The noise problem was also challenging when tuning an especially tricky parameter called `delta_start`, which controlled an upper bound slack to the time windows constraint. Figure 14 illustrates how slippery tuning this parameter was for 2,458 trials, often with some problems being OK while others the very same value being impossible to schedule.

Figure 14

Deterministic Optimization: 'Delta_Start' Values by Queue Size and Outcome (N=2458)



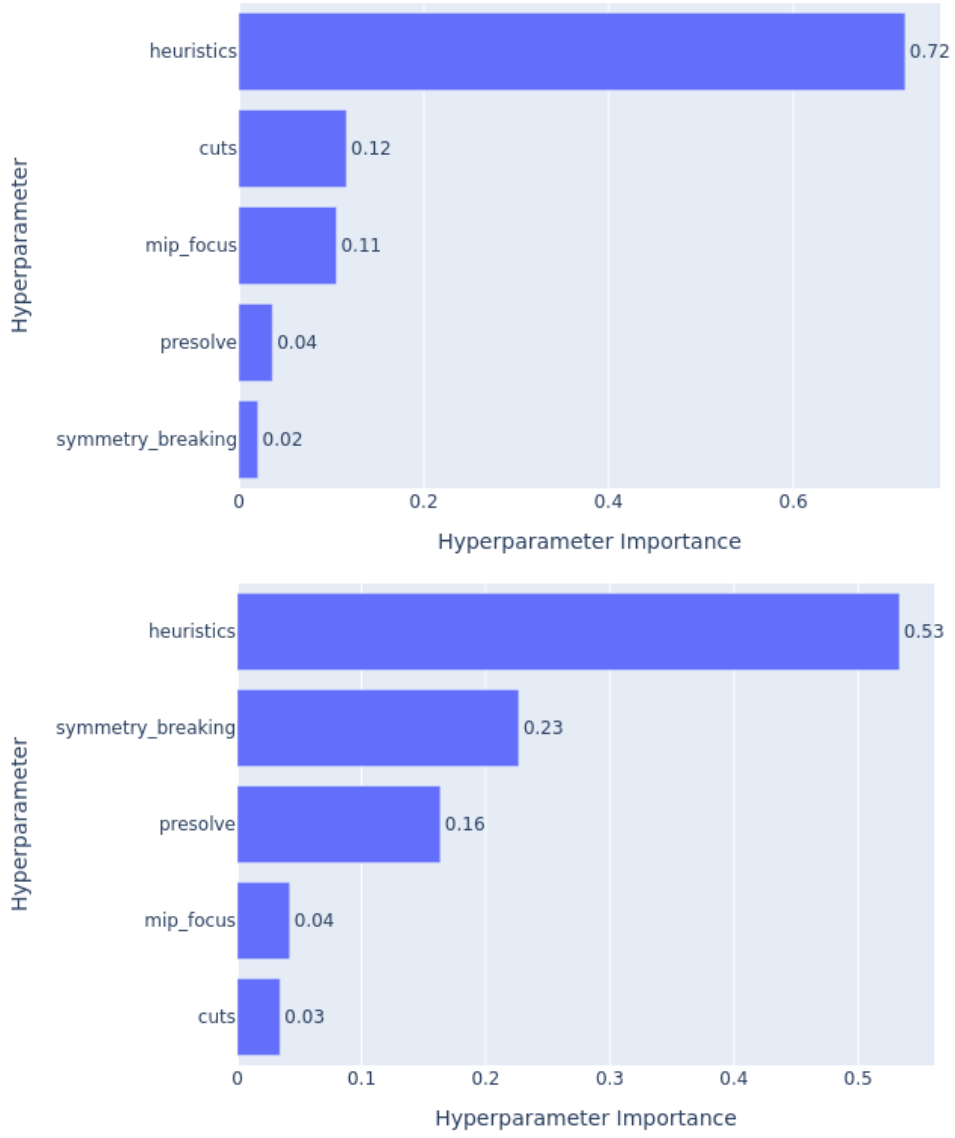
Note. Optimization outcomes could regularly fail immediately due to the constraints being impossible to satisfy; after meticulous debugging, this infeasibility meant the cause was that no schedule could fit within the time window limitations. Notably, the 75-job queue suffered more from this issue than the 25-job queue.

The project urgently needed to address the high failure rates because of the novelty of the hyperparameter tuning results shown in Figure 14. Specifically, "Max Fail" informed lower `delta_start` bounds mid-study for the 25- and 75-job queues. Thus, tuning the number of minutes to pad the time windows (delta start) was required, as seen in Figure 13, where the trials' "-100" values ceased.

Because of the mid-study change, Optuna did not show the delta start in Figure 15 or the hyperparameter importance plot; regardless, it still indicated important relationships between the remaining hyperparameters by quantifying with a percentage.

Figure 15

Deterministic Optimization: Hyperparameter Importance of 25 Versus 75-Queue



For example, after excluding `delta start`, the `heuristics`, by and large, was the most important parameter for this deterministic optimization model, and it was not even close. This parameter determined how much available time was balanced on proving optimality or finding solutions.

Interestingly, the `cuts` parameter (branching and bounding) was important for the 25-job queue but not for the 75 one. Similarly, `mip_focus` (the goal for the MILP optimization) mattered for the former but not the latter. However, vice versa, `presolve` (preprocessing applied to the problem) mattered for the latter but did not for the former. Last but not least was the `symmetry_breaking` constraints parameter; this one did not affect the usual 25-queue problems, but for the 75, it had much greater importance when Optuna sampled new trials.

As a result, the top three parameters per queue size were aggregated and moved to a validation stage where the project would choose the best model for evaluating a holdout problem set:

- *Model 25A*: {'symmetry_breaking': False, 'mip_focus': 1, 'cuts': 3, 'heuristics': 0.8706962488000624, 'presolve': 0, 'delta_start': 1291}
- *Model 25B*: {'symmetry_breaking': False, 'mip_focus': 2, 'cuts': 2, 'heuristics': 0.38255324651081446, 'presolve': 2, 'delta_start': 1291}
- *Model 25C*: {'symmetry_breaking': True, 'mip_focus': 2, 'cuts': 3, 'heuristics': 0.80407985237477, 'presolve': 0, 'delta_start': 1291}
- *Model 75A*: {'symmetry_breaking': False, 'mip_focus': 1, 'cuts': 0, 'heuristics': 0.9544772714508394, 'presolve': 1, 'delta_start': 2509}
- *Model 75B*: {'symmetry_breaking': False, 'mip_focus': 1, 'cuts': 2, 'heuristics': 0.7991870295315516, 'presolve': 1, 'delta_start': 2509}
- *Model 75C*: {'symmetry_breaking': True, 'mip_focus': 0, 'cuts': 1, 'heuristics': 0.6720072296006921, 'presolve': 2, 'delta_start': 2509}

Validation of Best Three Parameters. Now, the project pivoted to validating: after sampling ten problems without replacement from the training problem set for validation, Table 5 illustrates the results of linear mixed-effect models per queue size.

Table 5

Deterministic Optimization: Mixed Effects of Best-Three Models (Relative Objective Improvement %)

Effect	Coefficient	*SE*	95% CI		*p*
			LL	*UL*	
25-Queue					
Fixed Effects					
Model 25A	23.551	2.154	19.330	27.772	<.001
Model 25B	23.421	2.154	19.199	27.642	<.001
Model 25C	23.531	2.154	19.309	27.752	<.001
Random Effects					
Problem Number	46.256	73.681			
75-Queue					
Fixed Effects					
Model 75A	10.719	1.428	7.921	13.518	<.001
Model 75B	11.281	1.428	8.483	14.079	<.001
Model 75C	10.612	1.428	7.814	13.410	<.001
Random Effects					
Problem Number	19.941	17.427			

Note. Validated using only "training" problem sets (not holdout problems). Precisely, 25-queue's ten problems sample: 204, 267, 153, 10, 234, 227, 197, 110, 6, and 176; 75-queue's ten problems sample: 3, 34, 68, 97, 51, 84, 89, 77, 66, and 41. Number of problems for 25-Queue = 300, number of problems for 75-Queue = 100. CI = confidence interval; *LL* = lower limit; *UL* = upper limit. Intercepts were removed to aid contrasts.

Table 5 proved statistically with firm confidence (nearly 100%) that the study can reject the null hypothesis—that any tested deterministic optimization models were the same as the traditional approach—all p-values were near 0, and all 95% confidence intervals far above a 0% relative improvement percent. Likewise, accounting for "Problem Number" when modeling the group random effects was successful; both had high coefficients, overshadowing the fixed effects from each model's evaluation. Thus, the mixed effects model signaled how the validated models performed without all the confounding noise from unique problem instances. For model selection, Models 25A and 75B were reasonable models for evaluation.

Stochastic Optimization Solver Pre-training

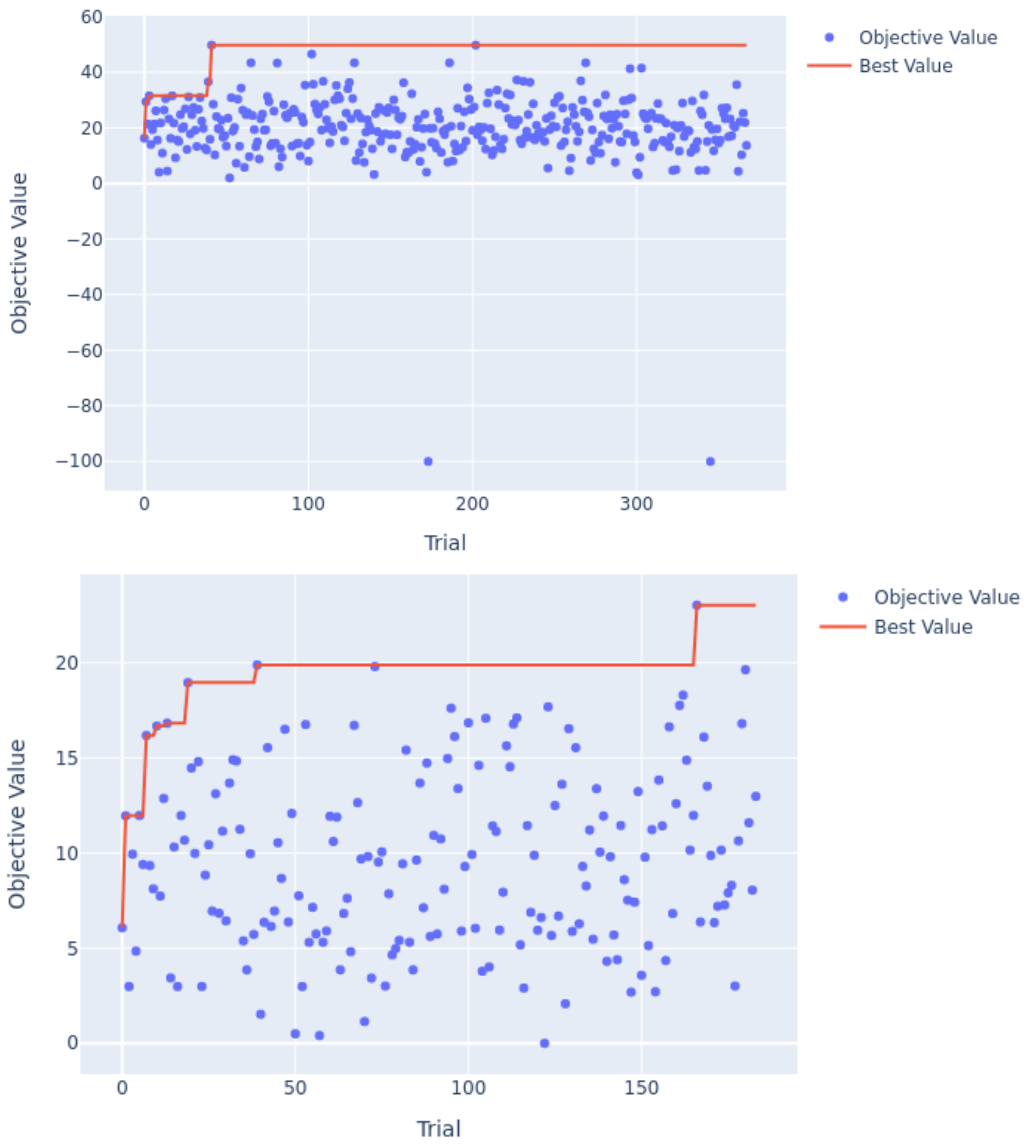
Similarly, stochastic optimization studies used the same setup. The only difference was that the `delta_start` bounds were set and unchanged—thanks to the learning from the prior deterministic optimization studies—allowing the pivotal soft-constraint-based variable to be included in

hyperparameter importance charting. Another difference was that the project conducted fewer trials due to limited time; however, given what was learned from the last studies, that was counterbalanced. Lastly, a new parameter was added to the mix, `nanmean`, or how scenarios averaged machine processing minutes into one array (NaN or 10000 for “bigM”), affecting the model plan.

Bayesian Optimization Tuning. Figure 16 shows the evolution of the Optuna’s hyperparameter tuning of the stochastic solver. Choosing more informed lower bounds for `delta_start` variables resulted in fewer failures or infeasible solutions (compared with Figures 13 and 14).

Figure 16

Stochastic Optimization: Bayesian Optimization (Optuna) for 25 and 75-Queue (N=552)



Note. "-100%" values show failed runs (infeasible) due to soft constraint violations on time windows (e.g., [a, b) with b slack), eased by padding the upper bound using a `delta_start` hyperparameter.

Figure 17 corroborates Figure 16, showing much fewer failures than Figure 14 had for the deterministic optimization. This confirms that the choice of `delta_start` was beneficial. Likewise, Figure 17's boxplots indicate distribution spreads within the success and fail solves. In particular, queue size 25

showed clustering in successful lower values despite the proximity to more chances of failure—a balance between tighter time windows and better-solving objectives versus an unsolvable problem.

Figure 17

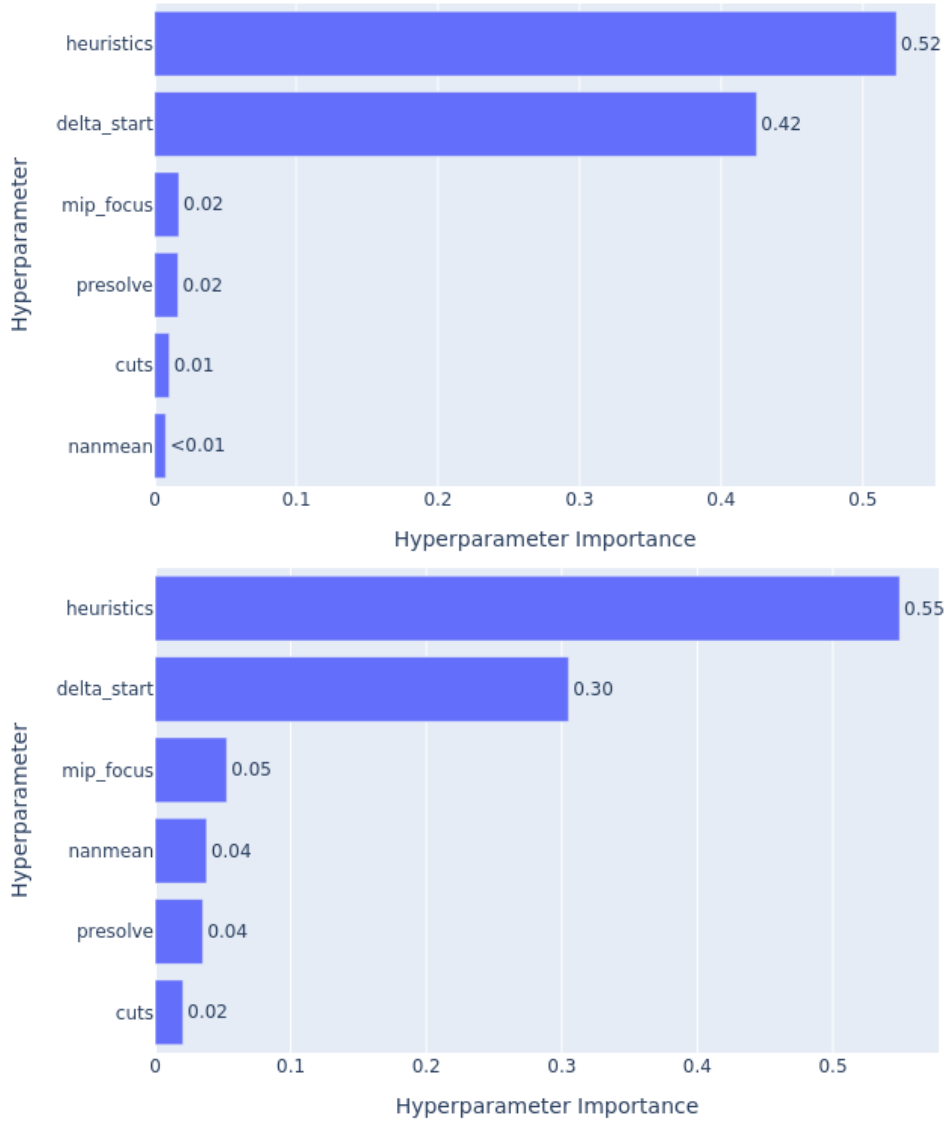
Stochastic Optimization: 'Delta_Start' Values by Queue Size and Outcome (N=552)

Note. After learning what `delta_start` minutes caused failures from Optuna's deterministic optimization search, the stochastic optimization studies ran with better lower bounds for `delta_start`; thus, they were less likely to fail.

Similarly, Figure 18 highlights the hyperparameters' importance, with `delta_start` being extremely important—second in importance for both subplots of 25- and 75-job queue solves—for assisting with objective value maximization, albeit lower than heuristics. Interestingly, `nanmean` did not have much importance for determining the objective value for the 25-job queue but had an order of magnitude more importance (more than 10 times) with 75-job queues.

Figure 18

Stochastic Optimization: Hyperparameter Importance of 25 Versus 75-Queue



Optuna's studies resulted in tuning the stochastic models' hyperparameters, finding that the following parameters performed best within the limited search trials:

- *Model 25A*: {'mip_focus': 0, 'cuts': 3, 'heuristics': 0.7563919824960467, 'presolve': 1, 'nanmean': False, 'delta_start': 818}
- *Model 25B*: {'mip_focus': 0, 'cuts': 3, 'heuristics': 0.7684551530323259, 'presolve': 1, 'nanmean': False, 'delta_start': 818}

- *Model 25C*: {'mip_focus': 3, 'cuts': 3, 'heuristics': 0.8678944201458589, 'presolve': 1, 'nanmean': False, 'delta_start': 818}
- *Model 75A*: {'mip_focus': 1, 'cuts': 2, 'heuristics': 0.5851062188632922, 'presolve': 1, 'nanmean': False, 'delta_start': 2619}
- *Model 75B*: {'mip_focus': 1, 'cuts': 2, 'heuristics': 0.7242407324722422, 'presolve': 0, 'nanmean': False, 'delta_start': 2619}
- *Model 75C*: {'mip_focus': 1, 'cuts': 3, 'heuristics': 0.9677167160301956, 'presolve': 0, 'nanmean': False, 'delta_start': 2619}

Validation of Best Three Parameters. Similar to deterministic optimization, the varying top model parameters for the best performing were validated using the same sample of ten jobs to select the top model per queue size. Table 6 shows the details of the results.

Table 6

Stochastic Optimization: Mixed Effects of Best-Three Models (Relative Objective Improvement %)

Effect	Coefficient	*SE*	95% CI		*p*
			LL	*UL*	
25-Queue					
Fixed Effects					
Model 25A	23.873	1.666	20.607	27.139	<.001
Model 25B	23.900	1.666	20.634	27.165	<.001
Model 25C	23.218	1.666	19.952	26.484	<.001
Random Effects					
Problem Number	27.254	22.126			
75-Queue					
Fixed Effects					
Model 75A	9.331	1.886	5.635	13.026	<.001
Model 75B	8.402	1.886	4.706	12.097	<.001
Model 75C	7.490	1.886	3.794	11.185	<.001
Random Effects					
Problem Number	34.322	18.061			

Note. Validated using only "training" problem sets (not holdout problems). Precisely, 25-queue's ten problems sample: 204, 267, 153, 10, 234, 227, 197, 110, 6, and 176; 75-queue's ten problems sample: 3, 34, 68, 97, 51, 84, 89, 77, 66, and 41. Number of problems for 25-Queue = 300, number of problems for 75-Queue = 100. CI = confidence interval; *LL* = lower limit; *UL* = upper limit. Intercepts were removed to aid contrasts.

Table 6 proves that stochastic optimization models were significantly better than the traditional method yet performed similarly to deterministic optimization. They did not achieve their project objective of 10% gains compared to the deterministic version (i.e., 30% compared to traditional).

Holdout Problem Evaluations With Traditional SPT Baseline

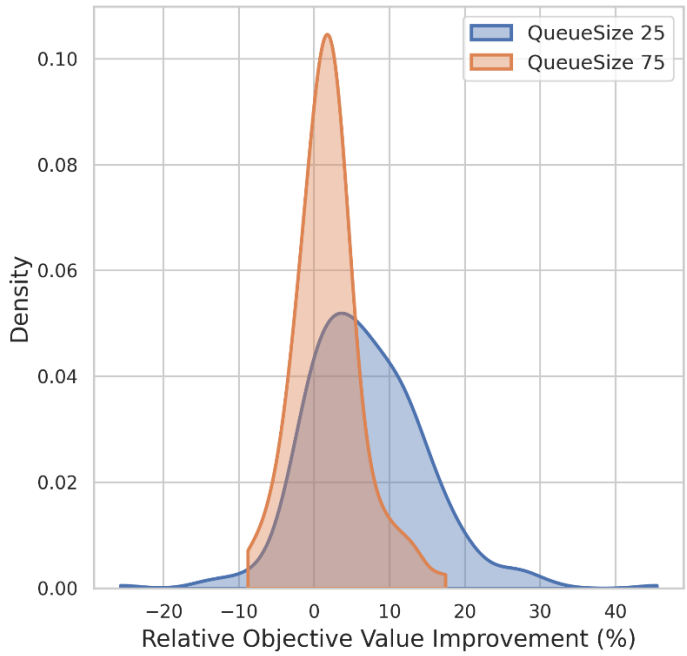
The project's culmination used the best deterministic and stochastic models from validation and computed results over thirty-six-hour stretches for every holdout problem in a total approach (N=all). This provided conclusive results (for all but the ”AI LPT” algorithm), informing manufacturers on a strategy for sophisticated scheduling algorithms that affect their bottom line: profits.

AI LPT Evaluation

For optimization, the discrete event simulation (DES) simulated an AI LPT approach (using four k-means recipe clusters) to provide good feasible warm starts based on naiive, expected arrival times; however, to fairly evaluate with traditional SPT using the holdout problems and true arrival times, for fairness, AI LPT also had to use true arrival times to parallel. In that context, and given that the AI-relevant project objective was a 10% relative improvement gain, Figure 19 shows an AI LPT holdout test.

Figure 19 includes a 0% improvement and negative, too, so there were instances where simply using the Traditional SPT approach outperformed the AI LPT method. Notably, the queue size of 75 clusters was much closer to the traditional approach than the 25-job queue version.

Figure 19
AI LPT Algorithm Performance (N=400)



A limitation of the results for AI LPT was that because the problem was sequentially scheduled, fixing a single sequencing rule constant across the entire horizon was unrealistic rather than having each iteration state fluidly allow the best rule. However, the project never imagined the need for this.

In this context, the provided results were inconclusive. Specifically, with a static decision policy (current DES AI LPT code) versus a more dynamic decision policy (e.g., “Use rule 1, then 3, then 1, then 20, then 4, …” versus “use rule 3”), there are always lower improvements. Thus, the project cannot prove that the AI LPT method showed a 10% improvement over the traditional SPT method.

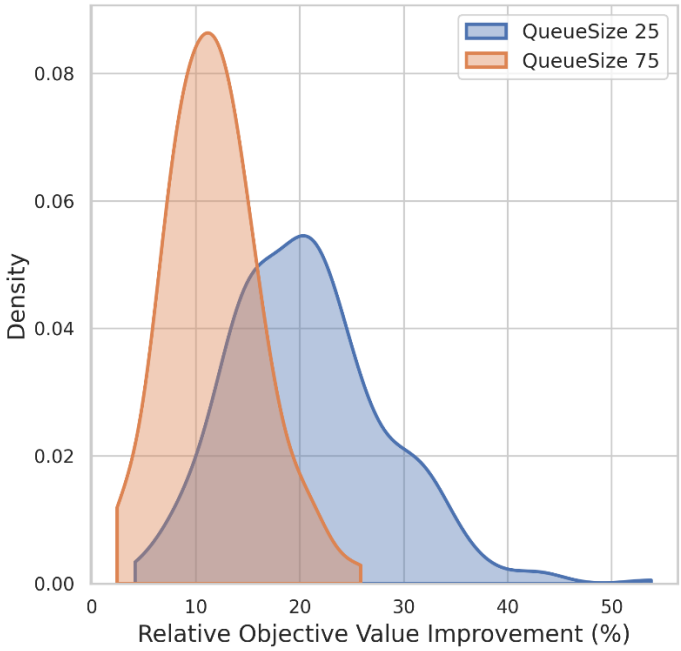
Deterministic Optimization Evaluation

Figure 20 shows the holdout results for the deterministic optimization using the best parameters selected from Optuna’s Bayesian search. As Figure 20 indicates, the entire domain of both the queue sizes of 25 and 75 showed above 0% improvements and the modes of around 20 and 10%

improvement, respectively. Interestingly, the 25-job queues had much greater opportunities within the 10-second solves than the 75-job queue solves. This will be discussed further in Chapter V.

Figure 20

Deterministic Optimization Algorithm Performance (N=400)



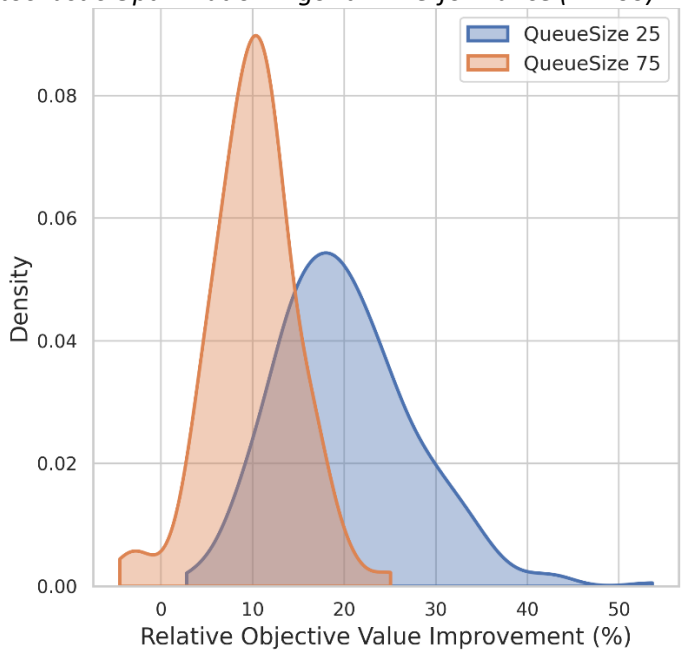
As a result, the deterministic optimization achieved its primary objective of improving at 20% or greater (25-job queue) and scaling well to the largest problem size (75-job queue) by never having a 0% improvement to the traditional algorithm.

Stochastic Optimization Evaluation

In contrast, Figure 21 illustrates the stochastic optimization model (i.e., lookahead), with parameters tuned similarly to the deterministic optimization.

Figure 21

Stochastic Optimization Algorithm Performance (N=400)



While Figure 21 indicates similar results for the 25-job queue compared to the deterministic optimization, the 75-job queue deviates markedly. With the larger problem sizes, there were cases where the optimization could not improve on the warm-start, thus even having a lower improvement than the traditional approach.

Comparing and Contrasting All Algorithms

Table 7 used a linear mixed effect model of all models to analyze and statistically compare the three algorithms by relative objective improvement percentages. This also allowed for grouping and controlling for the random effects of the problem number; this is evidenced by the fact that the SE of the Problem Number random effects for both queue sizes was 0.665 and 1.280, both relatively precise numbers when compared to the coefficients of 31.362 and 15.351, respectively.

Table 7

Linear Mixed Effects Model of All Models (Relative Objective Improvement %)

Effect	Coefficient	*SE*	95% CI		*p*
			LL	*UL*	
25-Queue					
Fixed Effects					
AI_LPT	6.838	0.549	5.762	7.913	<.001
DeterministicOpt	20.680	0.549	19.604	21.756	<.001
StochasticOpt	19.345	0.549	18.269	20.421	<.001
Random Effects					
Problem Number	31.362	0.665			
75-Queue					
Fixed Effects					
AI_LPT	1.798	0.456	0.904	2.692	<.001
DeterministicOpt	11.708	0.456	10.814	12.601	<.001
StochasticOpt	10.088	0.456	9.195	10.982	<.001
Random Effects					
Problem Number	15.351	1.280			

Note. Tested using all holdout problems. Number of problems for 25-Queue = 300, number of problems for 75-Queue = 100. CI = confidence interval; *LL* = lower limit; *UL* = upper limit. Intercepts were removed to aid contrasts.

Table 7 shows that all algorithms outclassed the traditional SPT algorithm baseline ($p < .001$) across the 25- and 75-job queues. Likewise, comparing the 95% CIs of the "AI_LPT" models with the "DeterministicOpt" and "StochasticOpt", the former significantly underperformed—[5.762, 7.913] not contained within [19.604, 21.756] and [18.269, 20.421], respectively (similarly with queue size 75). Moreover, Table 7 confirms that the stochastic optimization model contained 20% relative improvements, not achieving the main project objective of 30% for the business case study. Furthermore, as the optimization models' CI overlapped, the stochastic model failed to differentiate itself from its deterministic one.

Conclusion

Chapter IV evaluated the performance of blended AI and optimization models implemented in this project, addressing the objectives of improving manufacturing scheduling. Regarding the real-time objective of solving within 5 minutes, every algorithm was finished in under 10 seconds; thus, that

objective was achieved. However, optimizations were not within 5% optimality, as that was intractable for the problem space (stochastic optimization in particular) and project timeline; therefore, it was a mixed achievement according to the initial objective.

The AI Longest Processing Time (LPT) model produced inconclusive results below the 10% improvement target, with occasions where it underperformed compared to the traditional SPT method, particularly for larger queues. Its evaluation was limited by a fixed sequence rule, which constrained its dynamic ability to adapt to changing conditions.

In comparison, the deterministic optimization model reliably accomplished a relative improvement beyond 20% in the biobjective metrics (max flow time and average makespan) compared to the traditional Shortest Processing Time (SPT) approach, successfully meeting its scalability and improvement objectives.

Lastly, the stochastic optimization model showed improvements comparable to deterministic optimization for smaller queue sizes but did not outperform deterministic optimization for larger queues. It failed to achieve the 30% improvement objective and sometimes struggled compared to traditional methods.

These results partially met the project objectives and exposed opportunities for further enhancement. Chapter V addresses these limitations and recommends future research and implementation strategies.

Chapter V: Discussion

This chapter summarizes the findings in plain language regarding how well the project's blending of AI, optimization, and stochastic modeling performed with the parallel-processing job shop problem. It includes line items by project objectives and interpretation of the implications for a business implementing these algorithms. In more technical depth, the chapter explores the limitations of this project, as well as future research. Lastly, the chapter ends with a comprehensive project conclusion.

Summary of Findings

From Table 7, the most significant finding was that every algorithm produced in this project outperformed the traditional method. This means that even the "simplest" policy used—clustering and brute force discrete event simulations in parallel—outperformed the traditional approach. While the result was not satisfyingly wide, as the 25- and 75-job queues for the clustering method were at least 5 – 7% and 1 – 3% better, respectively, it is still safe to say that the findings support that all of the algorithms tended to perform significantly better.

In comparison, in Table 7, the MILP scheduling (with and without certainty) far outperformed the conventional way: 19 – 22% and 11 – 13% better, respectively, for 25- and 75-job queues—the lookahead had similar results with 18 – 20% and 9 – 11% improvements, respectively. Because of the sequential policy evaluation method adopted and just using 10-second solutions, the project, without any doubts, can claim that the optimization methods were, on average, much better than convention, often by about 20% if 25-job queues were standard, or about 10% better when prominent peaks occur.

Summary of Results by Project Objectives

The following summarizes the project results in the practical context of project objectives:

1. Although the clustering method (i.e., AI LPT) did not conclusively provide 10% targeted improvements on average for the 25-job queue, it statistically outperformed the traditional method (Table 7); this is interpreted as a practical success.
2. The MILP algorithm decisively improved the 25-job queue by 20% on average compared to the traditional way. Similarly, with 10% improvements, it proved scalable for the 75-job queue; thus, it was an all-around success practically and literally from the initial project objectives.
3. The lookahead algorithm failed to provide 30% baseline improvements on average for both the standard and peak queue sizes; therefore, it was considered a failed implementation.
4. The initial objective of consistently near-optimal solutions (i.e., 99.9% and 95% of 25- and 75-job queues within 5% of optimality) proved impossible. This was due to the computational infeasibility of perfectly optimizing 400 problems within the project timeline and its focus on relative improvements rather than absolute optimality. Instead, the project pivoted to a time-expensive iterative scheduling approach, setting a 10-second solve time as the MILP policy to prioritize real-time speed. Similarly, scalability objectives were evaluated using the 75-job queues for objectives 1, 2, and 3, achieving statistically significant gains (Table 7). As a result, with the redesigned scalability and real-time objectives, the project achieved practical success.

Discussion of Implications for Business

The significant performance of every scheduling algorithm compared to traditional methods (Table 7) tells a crucial story for manufacturing: complacency and prioritizing other continuous improvement projects risk missing substantial factory-wide gains in profitability. The common idiom “keep it simple, stupid” (KISS) often justifies reliance on simplistic scheduling methods, but this case study challenges that notion. Sophisticated problems require sophisticated solutions, and oversimplifying complexity may be why manufacturers continue to miss achievable profit gains. Instead, this project encourages manufacturers to heed the immortal wisdom of Albert Einstein: “Everything

should be as simple as it can be, but not simpler" (O'Toole, 2011). Even when perceived as black-box tools, sophisticated algorithms are well-suited to addressing complexities and delivering transformative improvements. This paper serves as a call to action for manufacturers to welcome advanced data science tools and reimagine their approach to high-stakes scheduling problems.

Another important business effect of the project was how standard-queue MILP solutions consistently achieved 20% gains within 10 seconds in a real-time fashion. Getting a schedule so quickly allows for quicker deployment, testing, stakeholder "buy-in", and application. Likewise, it will enable the frontline staff (i.e., David and Lisa) to have more real-time decision support, which can adapt swiftly rather than taking minutes (or hours) to solve. Real-time decision analytics allows direct manufacturing staff to solve quickly, grab the expected next jobs, be more prepared for setting up, and be more efficient as they learn what works.

Lastly, this project demonstrates the implication of an emerging paradigm of business strategy: "decision science". Evolving from operations research in the 1950s and data science over the last few decades, decision science offers the tools to tackle complex, high-stakes problems. For manufacturers, realizing a 20% relative improvement in average throughput at a bottleneck, as shown by this study's MILP policy, could translate into millions—or even billions—of dollars in profit gains or sales opportunities. This implication should create a fear of missing out (FOMO) for any business relying on basic decision methods instead of exploring advanced ones. Thus, by investing in decision science—through staff, upskilling, and tools—companies take a tangible step toward monetizing data into profitable applications, driving innovation, and securing a competitive edge in today's dynamic global market.

Limitations of Results and Future Research Suggestions

The limitations of this project were plenty (too many to mention all), but so also were the opportunities for improvement:

DES and Clustering Limits and Next Steps

Clustering Method Lacked Foresight on the Sequential Policy. The clustering method evaluation was limited because the project statically applied the policy over time. The project had designed every problem to be pre-solved only at the beginning of the iteration. This meant only using one policy statically rather than dynamically on the fly, which, in hindsight, will always be worse than more flexibility. Therefore, the project's DES script did not fairly represent the policy the clustering algorithm would have offered (i.e., underestimate bias).

The DES script would need some new features for a fairer evaluation of the clustering method. For one, it would need to on-the-fly recursively call itself and enumerate among the cluster sequences possible on every iteration (e.g., up to 36 or 33 for most effective), given only expected information (i.e., expected arrival times). This would also mean parallel processing within the script (with fault tolerance) to simulate the simulation policy properly. Likewise, the code must ensure that past decisions and true arrival times are locked in to maintain fairness when comparing policies.

With this clustering method suggestion, measuring the final objective value of the sequentially applied policy would allow a fairer evaluation of itself compared with other decision policies.

Hyperparameter Tuning Limits and Next Steps

Overlapping Sampling and Unexplored Boundaries Hindered Optuna's Potential. In hindsight, the project's hyperparameter tuning suffered wasted trials and implementation potential because its random search function overlapped significantly with the Bayesian search's (i.e., TPE). Furthermore, endpoint boundaries were not fully explored; thus, the TPE likely could not fully use the full breadth of

the hyperparameters' dependencies when sampling new parameters to solve with. In hindsight, the Bayesian optimization hyperparameter tuning would have benefited from using a boundary grid search, explicitly focusing on the lower and upper bound combinations first to enumerate the edge domain.

Consider a scenario with six hyperparameters[19], as in the lookahead policy: a boundary-only[20] grid search could evaluate the six combinations—$[\text{lower bound}, \text{upperbound}]^6 = 2^6 = 64$ trials. A pilot boundary-only grid search with 64 trials may boost TPE by pre-processing a more filled-out domain for the search. However, since a boundary grid search grows exponentially (2^p) with the number of parameters, p, parameter selection would be crucial to deciding the most impactful parameters before TPE. Ultimately, well-tuned optimization models, from the literature review in Chapter II, were proven to significantly improve the time to solve by up to 60%.

MILP Model Limits and Next Steps

Infeasible MILP Solves Plagued Time Window Constraints. The endings of the time windows as "soft" are not necessarily strictly applied, but the project's model applied them as if they were rigid, limiting the implementation. Firstly, it wastes time and energy solving impossible trials and runs (See Figures 13, 14, 16, and 17). Secondly, and more importantly, failing to solve means no decisions, thus limiting the value as this is a failure in production. While this did not often occur for the holdout evaluations, the lookahead had two infeasible 10-second solves on the 75-job queue (2% failure rate).

The most straightforward solution for this limitation would be to use a solver that directly applies soft constraints. Soft constraint support is valuable because it ensures the MILP model is flexible

[19] For example, here was Model 75A: {'mip_focus': 1, 'cuts': 2, 'heuristics': 0.5851062188632922, 'presolve': 1, 'nanmean': False, 'delta_start': 2619}

[20] For categorical hyperparameters (e.g., `cuts`), using the most conservative versus the most aggressive may suffice for boundaries.

and robust to less strictly applied constraints. It also ensures that a production application will not fail over a simple constraint.

However, many optimization solvers do not offer soft constraint functionality. Instead, the following formulation would effectively model a soft constraint at the cost of customizing the objective and breaking "apples-to-apples" comparisons. A real-valued, positive penalty variable z_i (and a weight, e.g., $\lambda = 0.0001$) would slack the objective per job i, softening the time window maximum constraint with a weak penalty. In other words, although solutions that violate start time maximums will increase the objective values, minimization reduces this as part of the optimization.

For the deterministic objective:

$$\text{Minimize} \quad w_1 y_1 + (1 - w_1) y_2 + \lambda \sum_i z_i.$$

(Biobjective) (1next)

For the stochastic objective:

$$\text{Minimize} \quad w_1 \sum_w p_w y_{1w} + (1 - w_1) y_2 + \lambda \sum_{w,i} p_w z_{iw}$$

(Expected Biobjective) (1snext)

Similarly, the formerly hard constraint of the end of the time window is relaxed, so the penalty variables act more as a soft constraint. See the revised constraint equations below:

$$a_i \leq t_i \leq b_i + z_i \quad (\text{for } i \in \boldsymbol{J_{1toN}}), \text{ (Time Windows) (13next)}$$

$$a_{iw} \leq t_{iw} \leq b_{iw} + z_{iw} \quad (\text{for } i \in \boldsymbol{J_{1toN}}; w), \text{ (Time Windows) (13snext)}$$

These suggested changes open the solution space, allowing for more exploration, multithreading capability, and feasible solutions to search—at a small cost of adding relatively few real

variables. This aligns perfectly with the project's goal of pinpointing superior schedules quickly. These soft constraints likely promise the most significant performance improvements of all the next steps.

Initial Symmetry-Breaking Constraints Impractical. Specific constraints were tried to tackle tie-breaking equivalent solutions that do not add value (i.e., symmetry-breaking) to improve solving quality (Equation 18). Given prior modeling efforts, these symmetry-breaking constraints were expected to be well-used and "dialed in". However, this was not the case.

Based on Figures 15 and 18, the Hyperparameter Importance plots for the deterministic and stochastic models showed that the symmetry constraints did not help. Often, when coupled with the best 75-job queue models not using the symmetry-breaking hyperparameter, it indicates they actively hindered better objective values. This makes sense in hindsight as the symmetry-breaking constraints more complex form (i.e., quadratic) thwarts the solving algorithm rather than simplifies it.

Instead, the recommendation for the next steps would be to use a more straightforward symmetry-breaking constraint (i.e., linear) that simplifies the problem fundamentally in a minimally invasive way. Namely, by greatly simplifying the furthest and farthest future jobs in the queue, the model can simplify job arrivals that were too uncertain to plan and instead focus on short and medium-term job optimization. The following shows the recommendations for tie-breaking:

Sets:

- $\boldsymbol{S_3}$: Be a spaced triple tuple of jobs (i, j, q) (e.g., $\{(10, 11, 12), (12, 13, 14), \dots\}$), to enforce a Longest Processing Time (LPT) tie-breaking among same-recipe jobs, constrained by a threshold τ_a (e.g. 4). Individual, same-recipe jobs in queue steps near the end of the scheduling horizon ($\text{Queue step} \geq \tau_a$) are unioned into recipe groups to simplify the problem and reduce complexity in the furthest, uncertain future.

Subject to:

$$x_{ijk} \leq x_{jqk} \quad (\text{for } (i,j,q) \in \boldsymbol{S_3}; k), \qquad \text{(Improved Symmetry-Breaking) (18next)}$$

The equation above allows combining future, same-recipe jobs to simplify large-scale problems. This tactic shifts the model's 'attention' toward immediate and near-term decisions while parametrizing the model with τ_a, allowing future hyperparameter finetuning like using ML.

Lookahead Model Limits and Next Steps

Ineffective lookahead model application. While the results were positive in meeting the main overall personal project goals and the primary project objectives, the model comparisons tell a different story. Although the scheduling method of clustering aided the MILP models with head starts, the lookahead model suffered greatly. Particularly when subjected to problem complexity (75-job queue)—the lookahead was never significantly different than the standard MILP optimization. In hindsight, this makes sense given a mere 10-second solve time and the greater difficulty in simultaneously solving similar yet nearly identical scenarios in a big model than in a less redundant optimization model.

Analogous to humans, the most significant chunk of time spent solving complex problems is often not on quick heuristics (clustering) or optimizing the decision (MILP) but on imagining uncertain random variables. A human imagining the infinitely different scenarios within their optimization (i.e., lookahead) may even mull over decisions for hours, days, or weeks. Similarly, with this project, future ideas could target how a lookahead policy fits within the algorithm pipeline.

One way to improve the lookahead policy's value would be to delegate it as nice-to-have and optionally applied after the initial 10-second MILP optimization in less real-time cycles. Unfortunately, by doing so, future research may never have enough time to sequentially demonstrate the policy's performance like this project's results (the time for each iteration would be much greater than 10-second solves). Nevertheless, treating a lookahead in production as optional and done after a quick

MILP solve to improve the decision robustness could create added value, for example, by eking out an additional genuine 5 – 15% improvement on the original MILP solution by considering important scenarios that add wrenches into the original plan. Given a system's time waiting for a new decision (e.g., the current bottleneck's schedule not being followed, a job arriving in the queue), looking ahead and considering possibilities may help balance robust and real-time operational decisions.

Moreover, a next step could plug the original MILP solution into the lookahead to conserve computing time with a sophisticated scheduling pipeline such as:

1. Discrete Event Simulation (DES) and finding the best warm-start brute force with fast heuristics (e.g., clustering method used and changing nearest neighbor to Dijkstra's algorithm (Brubaker, 2024))
2. MILP uses warm-start and quickly finds the best feasible solutions within 10 seconds.
3. Lookahead uses MILP as a warm-start to slowly find more robust and risk-averse decisions; then, as time passes—say, three or five-minute increments—new solutions are used as warm-starts for successive iterations.
4. Based on the circumstances, staff decide among the real-time or lookahead schedule[21].

Secondly, besides increasing the solve time from 10 seconds, the lookahead may benefit from exploring more simulation-based scenario generation. For instance, instead of the project's "hard-wired" selection of worst-case scenarios per recipe cluster by longest processing time (LPT), the method could simulate a broader range of possibilities, use sample average approximation (SAA), and apply scenario-reduction techniques—such as identifying scenario clusters and their importance through

[21] A caveat is that lookahead solutions adjusting real-time schedules will likely disrupt staging and frustrate staff. Clear justifications for changes (e.g., a separate displayed screen for the real-time versus the lookahead plan) and tracking decisions—adopted or not—can help the decision system (and support indirect staff in improving software later) while keeping direct labor staff engaged (e.g., David and Lisa).

principal components analysis (PCA) and unsupervised clustering. Also, parametrizing simulation depth, such as the number of incoming queue steps ahead (e.g., 1, 2, 3), allows tuning machine learning hyperparameters (e.g., Optuna or Bayesian search) to automate making fewer representative scenarios.

Miscellaneous Suggestions for Future Experimentation

In no particular order, the following are promising yet miscellaneous future research areas:

- *Rolling horizon parameterization in sequential policy*: See Appendix B (Considerations of Rolling Horizon Policy Methodology).
- *Reinforcement learning (RL) coupled with graph neural network (GNN)*: The literature highlighted the benefits of RL and GNN in large-scale implementations (Shahbazian et al., 2024). This method may vie with the DES implementation for warm-starting larger-scale models.
- *Scaling MILP with "PDLP":* Recent advances (Lu & Applegate, 2024) in GPU-based linear programming solving, replacing matrix factorization with matrix multiplication, offer potential for distributed computing in large-scale scheduling problems.
- *Exploring simulation-based lookahead solvers*: For example, test "InsideOpt" in proof-of-concept trials on pilot problems (e.g., 7-job queues) and gradually scale up (InsideOpt, n.d.).

Conclusion

This project explores the potential of combining optimization, AI, and stochastic modeling for manufacturing scheduling. By exploring alongside clustering methods, the study improved key throughput metrics (average machine work time and max flow time) within real-time limits of 10 seconds, validating a statistical advantage over a traditional scheduling method. Key successes included meeting objectives on scalability, relative improvements of 20% with the MILP (for standard queues), and demonstrating feasible real-time scheduling within 10 seconds for larger-scale 75-job queues.

Despite these achievements, the project exposed limitations. Challenges included an inadequate stochastic optimization implementation (computational infeasibility and shortfalls in 10-second solves), inefficiencies in hyperparameter tuning with boundary exploration gaps, and ineffective symmetry-breaking constraints. Additionally, the clustering method and the lookahead stochastic model fell short of definitively achieving their 10% and 30% improvement targets. However, these limitations will pave the way for future advancements.

Future research should address these limitations. One example is reimagining how lookaheads are deployed in production; this could involve extending its solve time substantially but practically. Future research also includes improving hyperparameter tuning through boundary grid search and integrating soft constraints (for time windows) to solve more robustly and reliably. Lastly, evolving techniques such as reinforcement learning, graph neural networks, GPU-based distributed solving of MILP, and simulation-based lookahead solvers should be explored as they could boost optimization and AI's impact on manufacturing.

From a business perspective, this study shows the tangible value of decision science in manufacturing. As demonstrated, a 20% improvement in bottleneck throughput would translate into substantial gains in profitability. These findings challenge the traditional reliance on simplistic scheduling methods, advocating for sophisticated algorithms suited to complex manufacturing. This project calls on manufacturers to capitalize on decision science as a strategic driver to unlock untapped potential and compete in today’s dynamic global economy.

In conclusion, this project showed how blending optimization, AI, and uncertainty can help schedule real-world problems. By applying decision science, businesses can make smarter, data-driven choices. These results aim to inspire manufacturers to imagine and design “smarter” systems, maximize their resources, and turn challenges into growth.

References

Azadeh, A., Goldansaz, S., & Zahedi-Anaraki, A. (2016). Solving and optimizing a bi-objective open shop scheduling problem by a modified genetic algorithm. *International Journal of Advanced Manufacturing Technology, 85*(5-8), 1603-1613. https://doi.org/10.1007/s00170-015-8069-z

Bertsimas, D., & Dunn, J. (2017). Optimal classification trees. *Machine Learning, 106*(7), 1039–1082. https://doi.org/10.1007/s10994-017-5633-9

Bischi, A., Taccari, L., Martelli, E., Amaldi, E., Manzolini, G., Silva, P., . . . Macchi, E. (2019). A rolling-horizon optimization algorithm for the long term operational scheduling of cogeneration systems. *Energy, 184*, 73–90. https://doi.org/10.1016/j.energy.2017.12.022

Brandinu, G., & Trautmann, N. (2014). A mixed-integer linear programming approach to the optimization of event-bus schedules: a scheduling application in the tourism sector. *Journal of Scheduling, 17*(6), 621–629. https://doi.org/10.1007/s10951-014-0375-z

Breaban, M. (n.d.). *Standard SD-MTSP* [PDF]. Faculty of Computer Science, Alexandru Ioan Cuza University. https://profs.info.uaic.ro/mihaela.breaban/mtsplib/Standard%20SD-MTSP.pdf

Brubaker, B. (2024, October 25). *Computer Scientists Establish the Best Way to Traverse a Graph*. Quanta Magazine. https://www.quantamagazine.org/computer-scientists-establish-the-best-way-to-traverse-a-graph-20241025/

Calculator Academy. (n.d.). *Newton's law of cooling calculator*. https://calculator.academy/newtons-law-of-cooling-calculator-2/

COIN-OR. (n.d.). *Cbc (Coin-or branch and cut)* [Computer software]. GitHub. https://github.com/coin-or/Cbc

Czakon, J. (2023, December 19). *Optuna vs Hyperopt: Which Hyperparameter Optimization Library Should You Choose?* neptune.ai. https://neptune.ai/blog/optuna-vs-hyperopt

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America, 2*(4), 393–410. https://doi.org/10.1287/opre.2.4.393

Desrochers, M., & Laporte, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters, 10*(1), 27–36. https://doi.org/10.1016/0167-6377(91)90083-2

Evreka. (2024, September 30). *Routing ten thousand nodes using capacitated clustering algorithm*. Medium. https://evrekadev.medium.com/routing-ten-thousand-nodes-using-capacitated-clustering-algorithm-aa5b8d057ea5

Frontline Systems releases Analytic Solver® V2022 with optimization performance enhancements. (2021). *In PR Newswire*. PR Newswire Association LLC.

Ghadimi, S., & Powell, W. B. (2022). Stochastic search for a parametric cost function approximation: Energy storage with rolling forecasts. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2204.07317

Goldratt, E. M., & Cox, J. (2014). *The goal: A process of ongoing improvement* (Fourth revised edition, 30th anniversary edition ed.). The North River Press Publishing Corporation.

Gozali, L., Kurniawan, V., & Nasution, S. R. (2019). Design of job scheduling system and software for packaging process with SPT, EDD, LPT, CDS and NEH algorithm at PT. ACP. *IOP Conference Series. Materials Science and Engineering, 528*(1), 12045. https://doi.org/10.1088/1757-899X/528/1/012045

Gurobi Optimization. (2019). *Solving simple stochastic optimization problems with Gurobi* [Video]. YouTube. https://www.youtube.com/watch?v=Jb4a8T5qyVQ

Gurobi Optimization. (2024). *Gurobi optimizer (Version 11.0)* [Computer software]. https://www.gurobi.com

Gurobi Optimization. (n.d.). *Gurobi optimizer documentation*. https://docs.gurobi.com/projects/optimizer/en/current/

Hexaly. (2024, June 2). *Hexaly vs Gurobi on the Traveling Salesman Problem (TSP)*. https://www.hexaly.com/benchmark/hexaly-vs-gurobi-traveling-salesman-problem-tsp

HiGHS. (n.d.). *HiGHS: Linear optimization software*. https://highs.dev/

Hillier, F. S., & Lieberman, G. J. (2015). *Introduction to operations research* (10th ed.). McGraw-Hill Education.

Hosny, A., & Reda, S. (2024). Automatic MILP solver configuration by learning problem similarities. *Annals of Operations Research, 339*(1–2), 909–936. https://doi.org/10.1007/s10479-023-05508-x

IBM. (n.d.). *What is CPLEX?* [Documentation]. https://www.ibm.com/docs/en/icos/22.1.1?topic=mc-what-is-cplex

Infanger, G. (1994). *Planning under uncertainty: Solving large-scale stochastic linear programs / Gerd Infanger.* Boyd & Fraser.

Infanger, G. (2011). *Optimization under uncertainty: The state of the art In honor of George B. Dantzig* (1st ed., Vol. 150). Springer Science + Business Media.

InsideOpt. (n.d.). *Documentation*. https://insideopt.com/pages/documentation

Ishihara, T., & Limmer, S. (n.d.). Optimizing the hyperparameters of a mixed integer linear programming solver to speed up electric vehicle charging control. *Applications of Evolutionary Computation*, (pp. 37–53). https://doi.org/10.1007/978-3-030-43722-0_3

Lei, D. (2009). Multi-objective production scheduling: A survey. *International Journal of Advanced Manufacturing Technology, 43*(9-10), 926–938. https://doi.org/10.1007/s00170-008-1770-4

Lu, H., & Applegate, D. (2024, September 20). *Scaling up linear programming with PDLP*. Google Research Blog. https://research.google/blog/scaling-up-linear-programming-with-pdlp/

Nahmias, S. (2009). Operations Scheduling. In *Production and operations analysis* (6th ed., pp. 417–454). McGraw-Hill/Irwin.

OpenAI. (2024). *ChatGPT (October 2024 version)* [Computer software]. https://www.openai.com

Optuna. (n.d.). *Optuna: A hyperparameter optimization framework*. https://optuna.org/

Ördek, B., Borgianni, Y., & Coatanea, E. (2024). Machine learning-supported manufacturing: A review and directions for future research. *Production & Manufacturing Research, 12*(1). https://doi.org/10.1080/21693277.2024.2326526

Othman, A., Haimoudi, E. k., Mouhssine, R., & Ezziyyani, M. (2019). An effective parallel approach to solve multiple traveling salesmen problem. In *Advanced Intelligent Systems for Sustainable Development (AI2SD'2018)* (Vol. 915, pp. 647–664). Springer International Publishing AG. https://doi.org/10.1007/978-3-030-11928-7_58

Pferschy, U., & Staněk, R. (2017). Generating subtour elimination constraints for the TSP from pure integer solutions. *Central European Journal of Operations Research, 25*(1), 231–260. https://doi.org/10.1007/s10100-016-0437-8

Powell, W. B. (n.d.). *The parametric cost function approximation*. CASTLE: Computational Stochastic Optimization and Learning. https://castle.princeton.edu/cfa/

Powell, W. B. (n.d.). *The parametric cost function approximation*. Retrieved from CASTLE: Computational Stochastic Optimization and Learning: https://castle.princeton.edu/cfa/

Pyomo. (2024, October 4). *Pyomo documentation (latest version)*. Read the Docs. https://readthedocs.org/projects/pyomo/downloads/pdf/latest/

Pyomo. (n.d.). *Working with Pyomo Models: Solving multiple instances in parallel* [Documentation]. https://pyomo.readthedocs.io/en/stable/working_models.html#solving-multiple-instances-in-parallel

Python Software Foundation. (n.d.). *multiprocessing — Process-based parallelism* [Documentation]. https://docs.python.org/3/library/multiprocessing.html

Python Software Foundation. (n.d.). *Python (Version 3.11)* [Computer software]. https://www.python.org

Qiao, W.-B., & Créput, J.-C. (2020). Multiple k−opt evaluation multiple k−opt moves with GPU high performance local search to large-scale traveling salesman problems. *Annals of Mathematics and Artificial Intelligence, 88*(4), 347–365. https://doi.org/10.1007/s10472-019-09679-x

Rasmussen, R. (2011). TSP in Spreadsheets – a Guided Tour. *International Review of Economics Education, 10*(1), 94–116. https://doi.org/10.1016/S1477-3880(15)30037-2

Ratnam, M. (2022, April 23). *Multioutput - multiclass classification*. Medium. https://medium.com/@cactuscode/multioutput-multiclass-classification-b0737a0693ec

Ren, S. C., Chaw, J. K., Lim, Y. M., Lee, W. P., Ting, T. T., & Fong, C. W. (2022). Intelligent manufacturing planning system using dispatch rules: A case study in roofing manufacturing industry. *Applied Sciences, 12*(13), 6499. https://doi.org/10.3390/app12136499

scikit-learn. (n.d.). *Examples/Multioutput: Multilabel classification using a classifier chain*. GitHub. https://github.com/scikit-learn/scikit-learn/blob/main/examples/multioutput/plot_classifier_chain_yeast.py

scikit-learn. (n.d.). *Multiclass and multioutput algorithms* [Documentation]. https://scikit-learn.org/stable/modules/multiclass.html

scikit-optimize. (n.d.). *skopt.BayesSearchCV* [Documentation]. https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html

Sefidian, A. M. (2022, August 1). *A simple tutorial on sampling, importance, and Monte Carlo with Python codes*. Medium. https://medium.com/@amir_masoud/a-simple-tutorial-on-sampling-importance-and-monte-carlo-with-python-codes-8ce809b91465

Shahbazian, R., Pugliese, L. D., Guerriero, F., & Macrina, G. (2024). Integrating machine learning into vehicle routing problem: Methods and applications. *IEEE Access, 12*, 93087–93115. https://doi.org/10.1109/ACCESS.2024.3422479

SimPy Developers. (n.d.). *SimPy: Discrete-event simulation for Python* [Documentation]. https://simpy.readthedocs.io/en/latest/

Solberg, J. J. (2009). *Modeling random processes for engineers and managers / James J. Solberg.* John Wiley.

Torres, R. G., Acharya, S., & Goel, P. (2023, December 23). *Solving Capacitated Vehicle Rerouting Problem with GNNs*. Medium. https://medium.com/gnns-for-cvrp/solving-capacitated-vehicle-rerouting-problem-with-gnns-88f1b90611ae

Upadhya, N. (2022, October 20). *Solving two-stage stochastic programs in Gurobi*. Towards Data Science. https://towardsdatascience.com/solving-two-stage-stochastic-programs-in-gurobi-9372da1e3ba8

Wang, Y., Lu, X., & Shen, J. (2021). Improved genetic algorithm (VNS-GA) using polar coordinate classification for workload balanced multiple Traveling Salesman Problem (mTSP). *Advances in Production Engineering & Management, 16*(2), 173–184. https://doi.org/10.14743/apem2021.2.392

What is Newton's Law of Cooling? (n.d.). WeTheStudy. https://www.wethestudy.com/tree-posts/what-is-newtons-law-of-cooling

XGBoost. (n.d.). *Multiple outputs* [Documentation]. https://xgboost.readthedocs.io/en/stable/tutorials/multioutput.html

Xuanyi, Z. (n.d.). *Optimal classification trees*. GitHub. https://github.com/XuanyiZhao/Optimal_classification_trees

Yıldız, S. N., Okay, F. Y., Islamov, A., & Özdemir, S. (2024). Improved chain-based multi-output classification for packaging planning. *Procedia Computer Science, 231*, 697–702. https://doi.org/10.1016/j.procs.2023.12.159

Appendix A: Consideration of Optimal Decision Trees

The study explored advanced variations of Decision Trees (DT) for their simplicity of interpretation, as they are well-known for their explainability. However, one tradeoff with DTs is that they have lower accuracy than higher-powered extensions, such as XGBoost and Random Forest (XGBoost, n.d.). Before the study, an improvement to DTs' standard nearest neighbor algorithm for splitting leaves and nodes was explored. It used optimization, such as a mixed integer linear program (MILP), to solve an optimal DT machine learning model for classifying data called an Optimal Classification Tree (Bertsimas & Dunn, 2017; Xuanyi, n.d.). These Optimal Classification Tree algorithms significantly improved on the conventional DT by 3 – 8% in accuracy. However, coding it within this project seemed infeasible with the added scope.

Appendix B: Considerations of Rolling Horizon Policy Methodology

Had more time been available in the project timeline, the project may have implemented the following parametrization within the model. The study explored implementing a Rolling Horizon Policy (Bischi et al., 2019)—a method that optimizes decisions incrementally over a moving planning window. By parametrizing the horizon through the queue step size, denoted as τ, this approach allowed for strategic tunings to the planning horizon, speeding up decisions based on the queue size at the cost of optimality. This flexibility could allow Bayesian optimization to fine-tune the model's sequential scheduling performance, letting τ adapt to varying problem characteristics. Notably, when τ equaled the maximum queue size (e.g., four or all queues upstream), the model reverted to the original MILP formulation.

Another aspect of the methodology involved simulating model performance across the job queueing evolution in time, mirroring a Discrete Event Simulation (DES) approach to accurately score how the model would adapt under uncertain arrival times. This simulated scoring method allowed for an "apples-to-apples" comparison with the traditional and clustering models, providing a fair and consistent framework to test each model's effectiveness in an imperfect information queueing system.

The formulation follows below:

Parameters:

- a: Estimated arrival times used in the initial planning.
- a_{true}: Actual arrival times are revealed progressively, defining each $\text{SIMTIME}^{(n)}$.
- $\text{SIMTIME}^{(n)}$: Current simulation time at iteration n, set by the actual arrival time $a_{\text{true}}^{(n)}$. Each nonzero a_{true} value determines a new n iteration of a job arriving at queue step 0.
- τ: Queue Step Horizon. The rolling horizon model uses maximum Queue Steps (0 to τ), $\tau = 1, 2, 3, \ldots$, to restrict the number of future queue steps that x_{ijk} can consider.

Previous Solution Variables:

- t_{ik}^{prev}: Start time of job i on machine k from the previous solution.
- x_{ijk}^{prev}: Previous solution's value for x_{ijk}.

$$x_{ijk} = x_{ijk}^{\text{prev}} \quad \left(\text{for } (i,j,k) \text{ such that } t_{ik}^{\text{prev}} \leq \text{SIMTIME}^{(n)} = a_{\text{true}}^{(n)} \text{ and Queue Step } q \leq \tau\right)$$

(Rolling Horizon) (19r)

The rolling horizon approach breaks down the main scheduling problem into smaller, manageable subproblems, each focusing on only part of the queue. This makes solving faster, as it avoids processing the entire queue simultaneously. In Equation 19r, each iteration moves forward based on unique, nonzero arrival times $a_{\text{true}}^{(n)}$, with the total number of iterations n determined by the number of unique arrival times.

Appendix C: Code URL

This project publically hosted the code (under an MIT License) on GitHub at:

https://github.com/jrwoodwa/Masters-Data-Science-DS785-Capstone-VRP-mTSP.