Sample Efficient Optimization

Optimizing Repair Workshop Operations: A Multi-Objective Bayesian Approach with Variance Reduction

---

Student :
Pedram Peiroasfia (11337867)

Professor:
Dr. Carolina Osorio

Contents

Abstract

This report presents an integrated framework for optimizing a repair workshop by minimizing the maximum queue length and operational costs. Using a custom discrete-event simulator, system behavior was modeled, and data was generated to train Sparse Gaussian Processes, enabling efficient and accurate surrogate modeling. Multi-Objective Bayesian Optimization was employed to identify Pareto-optimal trade-offs between conflicting objectives. Variance reduction techniques were applied to improve prediction reliability, leveraging correlations between maximum queue length and related metrics.

The framework demonstrated its capability to address complex optimization problems by balancing system performance and cost constraints. Solutions generated by the proposed methods were robust and interpretable, providing actionable insights for system enhancement. Limitations such as the absence of multi-task modeling and discrete-aware optimization were identified, offering avenues for future research and improved applicability in dynamic, real-world environments.

Keywords: Sparse Gaussian Processes, Multi-Objective Bayesian Optimization, Repair Workshop Optimization, Varaince Reduction

1 Introduction

Optimizing the performance of complex systems with limited resources is a cornerstone challenge in operations management. This project tackles the intricate dynamics of a repair workshop comprising five stations, each managed by varying numbers of operators. The primary aim is to minimize the maximum queue length ($Q_{\max}$), a critical indicator of bottlenecks and inefficiencies. Excessive queue lengths often result in customer dissatisfaction, reduced throughput, and operational disruptions, making their mitigation essential. However, addressing these challenges incurs costs, such as training operators or increasing station staffing. Balancing these competing objectives necessitates a systematic and efficient approach.

The problem is approached using a multi-faceted framework combining simulation, probabilistic modeling, and optimization techniques. A custom-built discrete-event simulator accurately replicates the operational flow of the workshop, capturing factors like inter-arrival times, service times, and order rework. This simulator forms the foundation for generating data and understanding system behavior.

To efficiently model $Q_{\max}$, Gaussian Processes (GPs) were employed as surrogate models. GPs, known for their flexibility and uncertainty quantification, offer a probabilistic framework to approximate complex relationships between inputs (e.g., staffing levels, service times) and outputs ($Q_{\max}$). Sparse Gaussian Processes were particularly utilized to mitigate computational challenges, enabling scalable modeling of large datasets while maintaining robust predictions.

The optimization phase leveraged Multi-Objective Bayesian Optimization (MOBO) to address two conflicting objectives: minimizing $Q_{\max}$ and minimizing the operational costs. MOBO not only explores the decision space efficiently but also identifies a Pareto front that represents optimal trade-offs. Advanced acquisition functions, such as Batch Noisy Expected Hypervolume Improvement (qNEHVI) and Batch Noisy Pareto Efficient Global Optimization (qNParEGO), guided the optimization process, ensuring efficient exploration and exploitation of the decision space.

Finally, variance reduction techniques were integrated to enhance the reliability of predictions, especially for $Q_{\max}$. By leveraging related metrics, such as the maximum waiting time ($W_{\max}$), the variability in predictions was reduced, leading to more robust and interpretable results. This comprehensive approach provides actionable insights into balancing efficiency and cost in operational systems, offering a framework adaptable to other resource-constrained settings.

The remainder of this report is structured as follows: Section 2 provides a detailed overview of the repair workshop and the simulator developed to model its operations, including the characteristics of its stations and workflow. Section 3 introduces Gaussian Process modeling, discussing the rationale behind the chosen configurations and their application to surrogate modeling. Section 4 focuses on MOBO, explaining the methods employed to balance conflicting objectives and the results obtained. Section 5 delves into the variance reduction techniques applied to improve the reliability of predictions, particularly for $Q_{\max}$. Finally, Section 6 discusses the limitations of the current framework, highlighting areas for future exploration and potential advancements.

2 Simulator

2.1 Workshop Description

The repair workshop consists of five workstations, each with a specific role in the repair process. The system handles two types of orders: **standard orders**, which are regular tasks, and **urgent orders**, which need faster processing due to their priority. Both types of orders arrive at the workshop at different rates and follow a set process depending on their needs.

The process begins at **Station A**, where all orders are received and prepared for repair. From there, the orders move to **Station B**, where parts are replaced if needed. After Station B, most orders go to **Station C** for cleaning and degreasing, but some orders skip this step and go directly to **Station D**, where parts are assembled, and adjustments are made. **Station C** is different from the other stations because it processes all orders in the order they arrive, without prioritizing urgent ones. This station also needs regular maintenance, which can temporarily stop its operation. After cleaning at Station C, the orders continue to **Station D**, but a small percentage of them may require rework. These reworked orders go back to **Station B** for additional repairs, then return to Station D and finally proceed to the next station.

The last step in the process happens at **Station E**, where the orders are packed and shipped out. At this point, the repair process is complete, and the orders leave the system. There are two main paths for orders in the workshop: most orders follow the sequence **A** $\rightarrow$ **B** $\rightarrow$ **C** $\rightarrow$ **D** $\rightarrow$ **E**, while a smaller number follow **A** $\rightarrow$ **B** $\rightarrow$ **D** $\rightarrow$ **E**, skipping the cleaning step at Station C. Moreover, those orders that require rework, follows the route in Figure 1.

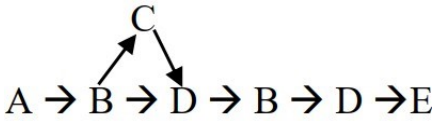

Figure 1: Workflow of orders with rework

The workshop uses shared queues to manage the flow of orders, and urgent orders are prioritized at most stations, except for Station C. This ensures that critical repairs are done as quickly as possible. A scheduling system is used to decide the order of tasks and make sure the system runs smoothly.

In summary, These features mentioned below ensures that the simulator closely resembles the behavior of a real-world repair workshop, providing valuable insights into system dynamics and areas for improvement:

- The simulator captures the sequential nature of the workflow, including the two distinct paths (with and without cleaning) and the possibility of rework.
- It models realistic priority handling, ensuring that urgent orders are processed faster at all stations except Station C.
- Variability in order arrivals and service times is incorporated, reflecting the unpredictable nature of real-world operations.

- Regular maintenance of Station C is included, representing real-world downtime and its impact on system performance.
- The rework logic is implemented for Station D, where orders may loop back for further processing, adding complexity and realism to the system.
- Shared queues and first-come-first-serve policies mimic real-world queuing and scheduling dynamics.
- Key metrics like queue lengths, waiting times, and utilization rates are tracked, enabling performance analysis that mirrors real operational evaluations.

You can also have an overview of the system in Figure 2 assuming that there is 1 machine at workstations A and C, 3 operators at workstations B and E, and 4 operators at workstation D.

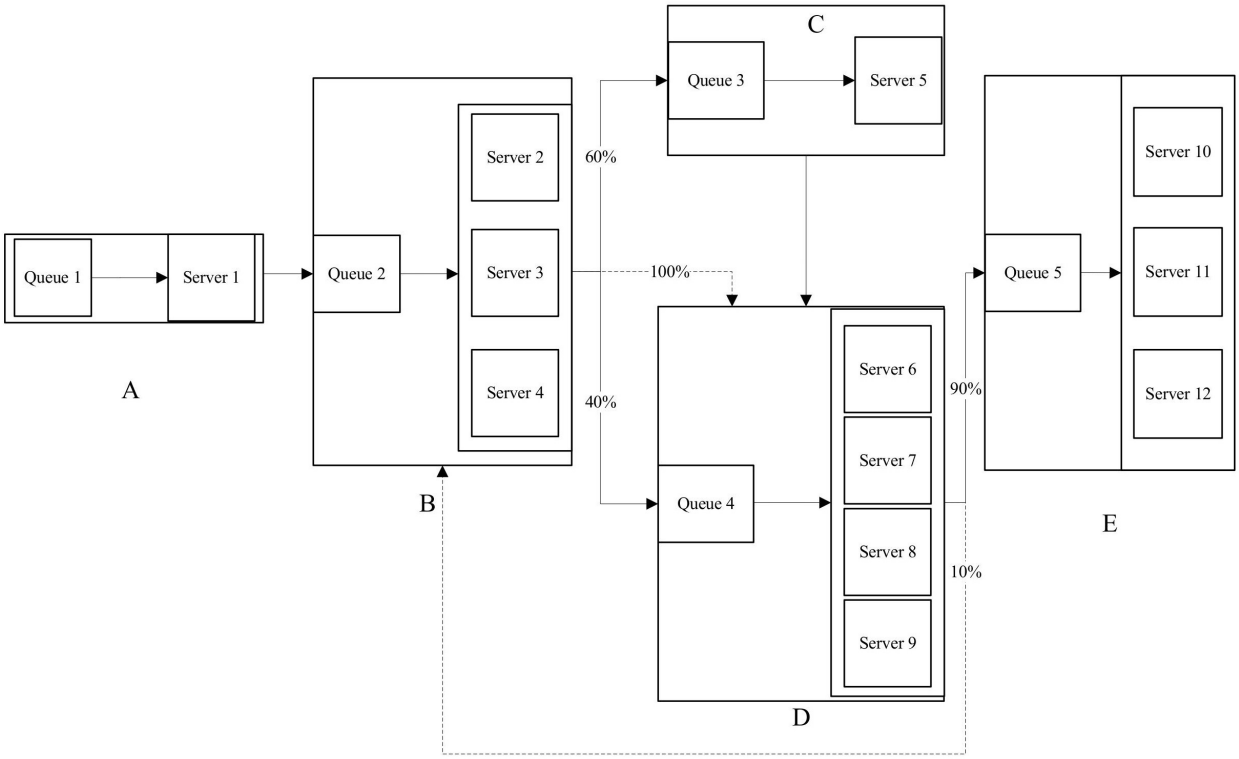

Figure 2: Workflow of orders with rework

It's also worth noting that using our simulator we can have access to the following statistics:

- Average response time of each station between a user-defined time interval
- Average Inter-arrival time for both type of orders (standard and urgent) for different times of day
- Average waiting time of orders in each of the stations
- Average total queue length at each workstation

- Average maximum waiting time in the queue

For further elaboration on how the system were simulated using discrete-event simulation principles, you can read Appendix A.

2.2 Data Generation

To determine the most suitable Gaussian Process (GP) model for our system, we generated data using the simulator by simultaneously incorporating various operational parameters. The data generation process was designed to capture the complexity of the workshop and ensure that all key aspects of its operations were reflected in the dataset. The simulator was run under different settings, including varying inter-arrival times based on the time of day (morning, afternoon, night) and order type (standard or urgent). Additionally, service times for operators were sampled from a range of probability distributions with different parameters, ensuring the dataset captured realistic variability in task durations. The number of operators in each stations was uniformly sampled. At the same time, rework loops and fixing times at Station C were integrated into the simulations, reflecting the delays caused by maintenance and rework processes.

Complete description of the data generation phase and distributions used are detailed in Tables 1, 2, 3 and 4.

Table 1: Bounds for the Number of Operators in Each Station. Random sampling was performed from a discrete uniform distribution within the specified lower and upper bounds. Stations A and C have fixed operator counts due to the nature of their tasks.

Station	Lower Bound	Upper Bound
A	1	1
B	2	6
C	1	1
D	2	6
E	2	6

Each simulation record is generated by running the system for a period of 5 days, repeated 10 times to compute the statistics. Data collection begins from the 8^{th} hour, accounting for the warm-up period, and continues until the end of the 5-day period. Using this, a total of 102,000 records were generated. The columns of the dataset generated are as below:

- number of operators in stations A to E (five columns)
- average service time of operators in stations A to E (five columns)
- inter-arrival time of different orders in different times of day (six columns)
- average queue length of stations A to E (five columns)
- maximum queue length (one column)

Table 2: Distributions and their lower and upper bounds are provided for each station. For each station, a random distribution is selected from the available options. To determine the parameters of the chosen distribution, random values are uniformly sampled within the specified lb and ub for each parameter. For instance, at station E, if the Lognormal distribution is selected, the parameter **mean** is sampled from $U(3.2, 3.5)$, and the parameter **std** is sampled from $U(0.7, 0.9)$. These sampled values are then used to generate a Lognormal random variable (e.g., $\mathcal{LN}(3.3, 0.75)$).

Station	Distribution	Parameters
A	Triangular	{'min_lb': 7, 'min_ub': 8} {'mode_lb': 12, 'mode_ub': 15} {'max_lb': 20, 'max_ub': 22}
	Gamma	{'shape_lb': 3, 'shape_ub': 4} {'scale_lb': 3, 'scale_ub': 4}
	Lognormal	{'mean_lb': 2.5, 'mean_ub': 2.8} {'std_lb': 0.5, 'std_ub': 0.7}
	Uniform	{'min_lb': 8, 'min_ub': 10} {'max_lb': 20, 'max_ub': 22}
	Normal	{'mean_lb': 20, 'mean_ub': 22} {'std_lb': 2, 'std_ub': 3}
B	Triangular	{'min_lb': 6, 'min_ub': 7} {'mode_lb': 14, 'mode_ub': 16} {'max_lb': 22, 'max_ub': 24}
	Gamma	{'shape_lb': 3.5, 'shape_ub': 4.5} {'scale_lb': 3, 'scale_ub': 3.5}
	Lognormal	{'mean_lb': 2.8, 'mean_ub': 3.0} {'std_lb': 0.6, 'std_ub': 0.8}
	Uniform	{'min_lb': 10, 'min_ub': 12} {'max_lb': 22, 'max_ub': 25}
	Normal	{'mean_lb': 20, 'mean_ub': 22} {'std_lb': 2, 'std_ub': 3}
C	Fixed	{'mean_lb': 16, 'mean_ub': 20}
	Normal	{'mean_lb': 16, 'mean_ub': 17} {'std_lb': 2, 'std_ub': 2.5}
D	Triangular	{'min_lb': 8, 'min_ub': 9, } {'mode_lb': 16, 'mode_ub': 18} {'max_min': 24, 'max_ub': 26}
	Gamma	{'shape_lb': 4, 'shape_ub': 5} {'scale_lb': 3.5, 'scale_ub': 4}
	Lognormal	{'mean_lb': 3.0, 'mean_ub': 3.2} {'std_lb': 0.6, 'std_ub': 0.8}
	Uniform	{'min_lb': 10, 'min_ub': 12} {'max_lb': 22, 'max_ub': 24}
	Normal	{'mean_lb': 21, 'mean_ub': 23} {'std_lb': 2, 'std_ub': 2.5}
E	Triangular	{'min_lb': 9, 'min_ub': 10} {'mode_lb': 18, 'mode_ub': 20} {'max_lb': 26, 'max_ub': 28}
	Gamma	{'shape_lb': 4, 'shape_ub': 5} {'scale_lb': 3.5, 'scale_ub': 4.5}
	Lognormal	{'mean_lb': 3.2, 'mean_ub': 3.5} {'std_lb': 0.7, 'std_ub': 0.9}
	Uniform	{'min_lb': 12, 'min_ub': 14} {'max_lb': 25, 'max_ub': 28}
	Normal	{'mean_lb': 22, 'mean_ub': 24} {'std_lb': 2, 'std_ub': 3}

Table 3: Distributions and Parameters for rework and fixing time of the machine C.

Type	Distribution	Parameters
Rework	gamma	{'shape_lb': 3, 'shape_ub': 4, 'scale_lb': 6, 'scale_ub': 8}
	lognormal	{'mean_lb': 3.5, 'mean_ub': 3.8, 'std_lb': 0.8, 'std_ub': 1.0}
fixing Time	fixed	{'mean_lb': 30, 'mean_ub': 40}
	uniform	{'min_lb': 30, 'min_ub': 32, 'max_lb': 38, 'max_ub': 40}

Table 4: Distribution and Bounds of Parameter Mean of Exponential Distribution for Different Time Categories and Order Types.

Time Category	Order Type	Distribution	Lower Bound	Upper Bound
morning	standard	exponential	15	18
	urgent	exponential	18	20
afternoon	standard	exponential	22	25
	urgent	exponential	25	28
night	standard	exponential	14	16
	urgent	exponential	16	18

Now, what's maximum queue length?

The maximum queue length is defined as:

$$Q_{\max} = \max(Q_1, Q_2, \ldots, Q_n)$$

where Q_i represents the queue length at station i.

Why Choose Maximum Queue Length?

The $Q_{\max}$ is chosen as the response variable because it directly captures the system's most critical performance indicator: the bottleneck. In repair workshops, bottlenecks often dictate the system's overall efficiency, as delays at a single station can cascade through the entire workflow. Focusing on $Q_{\max}$ allows us to target and minimize the worst-case scenario, ensuring smoother operations and better system reliability.

This metric is not only intuitive but also computationally simple, making it practical for modeling and optimization. By targeting $Q_{\max}$, we can align system performance with operational goals, such as reducing waiting times and avoiding customer dissatisfaction. Furthermore, $Q_{\max}$ is well-suited for GP modeling due to its continuous and typically smooth behavior (comparing to harmonic mean or simple average which fluctuates a lot and don't capture the full performance of the system) with respect to changes in system parameters, enabling robust predictions and optimization.

While $Q_{\max}$ is a critical metric, it has some drawbacks:

- Limited System-Wide Insight: It focuses solely on the most congested station, potentially overlooking inefficiencies in other parts of the system.
- Neglect of Underutilized Stations: Stations with zero or low queue lengths are ignored, which might result in suboptimal resource allocation.

Alternatives such as the weighted sum of queue lengths, 95th percentile queue length, or harmonic mean of queue lengths could address some of these issues by providing a more balanced view of system performance. However, these metrics may dilute the focus on the most critical bottlenecks, which is often the primary concern in capacity planning and operational optimization.

By incorporating all these factors simultaneously in a unified simulation, the generated dataset represents a comprehensive view of the workshop's dynamics. This integrated approach ensures that the GP models are exposed to realistic interactions between different system parameters, enabling more accurate predictions and robust optimization results.

2.3 Exploring the Generated Data

After generating the data using the simulator, it is crucial to analyze the distributions of the generated columns to understand the behavior and variability of the system parameters. This analysis provides insights into how well the simulator captures the complexities of the workshop's operations and helps validate the realism of the generated dataset.

2.3.1 Server Count Distributions

The server count distributions for stations B, D, and E are shown in Figure 3. As expected, the server counts were sampled uniformly within the defined bounds for each station. This uniform distribution ensures that the dataset captures a wide range of possible staffing configurations, from minimal to maximal operator availability. This variability is critical for testing the impact of staffing levels on the system's performance and identifying configurations that minimize the $Q_{\max}$.

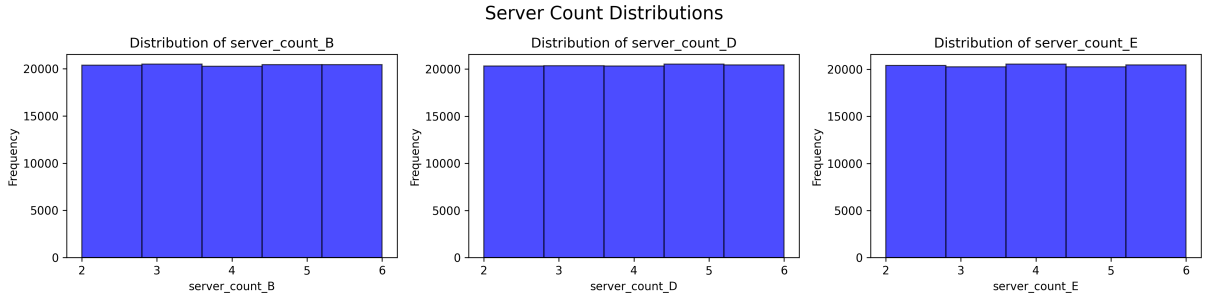

Figure 3: Distributions of server counts for Stations B, D, and E.

2.3.2 Service Time Distributions

Figure 4 shows the distributions of mean service times for stations A to E. These distributions reflect the variability introduced through the use of multiple probability distributions (e.g., triangular, gamma, lognormal). The variations in mean service times capture the differences in operational tasks across stations, ensuring that the simulator models realistic service durations. For instance, the peaks and ranges in the distributions indicate that certain stations, such as D and E, may experience more variability in service times compared to others.

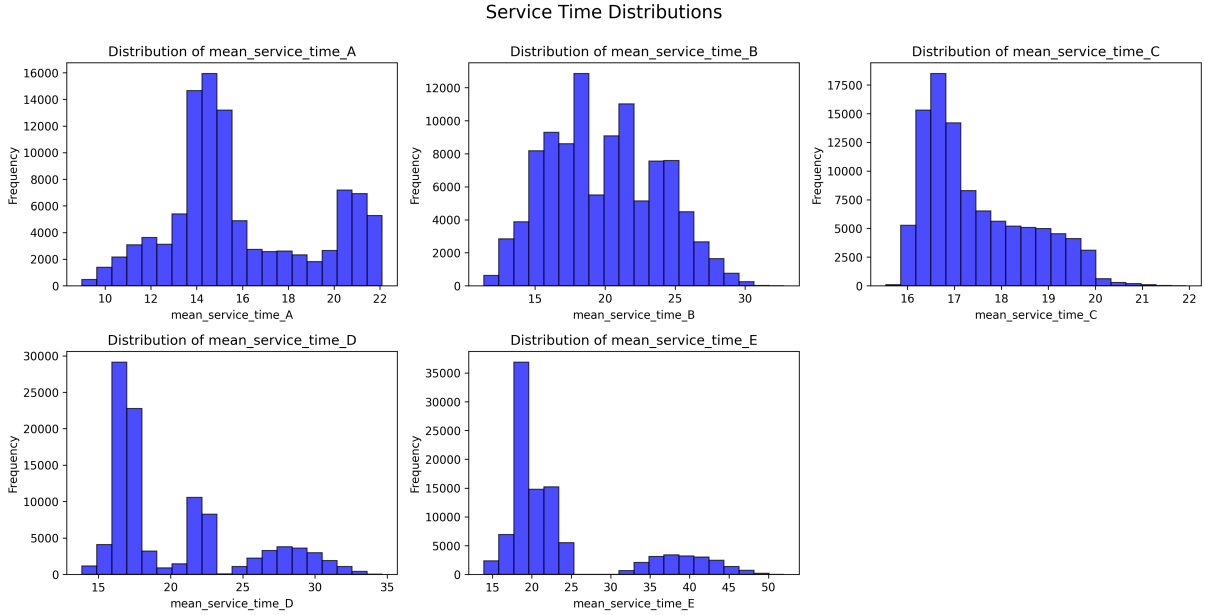

Figure 4: Distributions of mean service times for Stations A, B, C, D, and E.

2.3.3 Inter-Arrival Time Distributions

The inter-arrival time distributions for different order types and times of day are displayed in Figure 5. These distributions highlight the differences in order arrival patterns across standard and urgent orders, as well as the variations between morning, afternoon, and night shifts. The distributions show a balanced representation of the arrival times within their respective bounds, ensuring that the simulator captures realistic demand fluctuations throughout the day.

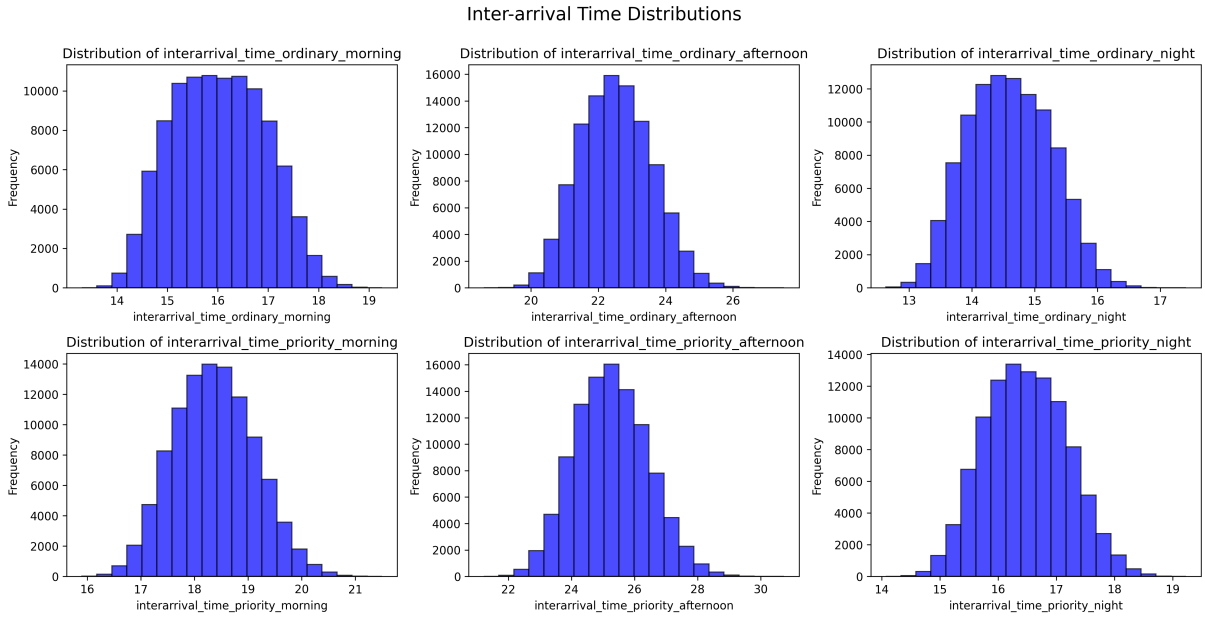

Figure 5: Distributions of inter-arrival times for different order types and times of day.

3 Gaussian Process Modeling for Queue Length Prediction

3.1 Introduction to Gaussian Processes

Gaussian Processes (GPs) (Ebden, 2015) are a powerful and flexible approach to modeling complex systems where the relationships between inputs and outputs are not explicitly defined. At their core, GPs provide a probabilistic framework for learning functions from data. Specifically, a GP is a collection of random variables, any finite subset of which has a joint Gaussian distribution. This property makes GPs particularly well-suited for regression tasks, as they model the underlying function as a distribution over possible functions, rather than a single deterministic output.

Mathematically, a GP is defined as:

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

where:

- $\mu(\mathbf{x})$ is the mean function, which provides the expected value of the function at a given input $\mathbf{x}$.
- $k(\mathbf{x}, \mathbf{x}')$ is the kernel or covariance function, which defines the relationship between function values at different input points $\mathbf{x}$ and $\mathbf{x}'$.

The mean function is often assumed to be zero for simplicity, while the choice of kernel function (e.g., squared exponential, Matérn) determines the smoothness and complexity of the modeled function.

GPs have several advantages that make them highly effective for modeling $Q_{\max}$ in our system (Ebden, 2015):

1. Capturing Complex Relationships: GPs can model non-linear and highly complex relationships between input parameters (e.g., server counts, inter-arrival times) and the response variable ($Q_{\max}$). The flexibility of the kernel function allows GPs to adapt to a wide range of functional forms, ensuring that the model can accurately capture the underlying dynamics of the system.
2. Uncertainty Quantification: Unlike deterministic models, GPs provide a measure of uncertainty for their predictions. This is achieved by estimating a posterior distribution for the function values at unobserved points. The predictive variance provides valuable information about regions where the model is less confident, guiding the exploration of the input space during optimization.
3. Sample Efficiency: GPs are inherently sample-efficient due to their Bayesian nature. By leveraging prior knowledge encoded in the kernel function, GPs require fewer data points to achieve high predictive accuracy, making them ideal for problems where data generation is costly or time-consuming, as in our simulation-based approach.
4. Interpretability: GP models are interpretable, as the kernel function explicitly defines the correlations between inputs, and the posterior distribution provides insights into the model's confidence across the input space.

3.2 GP Model Setup

3.2.1 Input Data

To model the $Q_{\max}$ effectively, we define the input features and response variable as follows:

Input Features: The input features include:

- **Server counts:** The number of operators at each workstation (B, D, and E).
- **Service times:** The mean service times for each station (A to E).
- **Inter-arrival times:** The inter-arrival times for standard and urgent orders across different times of the day (morning, afternoon, night).

These inputs capture the key factors influencing $Q_{\max}$, ensuring the model comprehensively represents the system's behavior.

Response Variable: The response variable is $Q_{\max}$.

3.2.2 Time Complexity of GP

One of the main limitations of GPs is their computational cost (Liu et al., 2019), which becomes significant as the dataset size increases. The time complexity of training a GP model is dominated by the need to invert the covariance matrix, a process with a complexity of $\mathcal{O}(n^3)$, where n is the number of training data points. This cubic scaling arises because the covariance matrix is $n \times n$, and matrix inversion is computationally expensive. Similarly, the memory requirement is $\mathcal{O}(n^2 + nd)$. For large datasets, this becomes a critical bottleneck, making it challenging to apply GPs to problems with thousands or millions of data points.

At prediction time, the complexity of computing the predictive mean and variance is $\mathcal{O}(n^2)$ for each test point, as it requires computing the covariance between the test point and all training points. While this is linear for a single prediction, making predictions for a large number of test points can still be computationally demanding.

For our dataset with $n = 100,000$ training points and $d = 14$ features, for memory requirement:

- The covariance matrix requires n^2 elements. Assuming each element requires 8 bytes (64-bit floating-point numbers), the memory for the covariance matrix is:

$$n^2 = 100,000^2 = 10^{10} \quad \text{elements}, \quad \text{or } 80\,\text{GB}.$$

- The feature data requires nd elements. For 8 bytes per element:

$$nd = 100,000 \times 14 = 1,400,000 \quad \text{elements}, \quad \text{or } 11.2\,\text{MB}.$$

- Total memory requirement:

$$\text{Total memory} = 80\,\text{GB} + 11.2\,\text{MB} \approx 80\,\text{GB}.$$

Moreover, for computation time of training, we have:

- The number of floating-point operations required is:

$$n^3 = 100,000^3 = 10^{15} \quad \text{operations.}$$

- Assuming a computational speed of 10 GFLOPS (10 billion floating-point operations per second) on a standard CPU, the training time is:

$$\text{Time} = \frac{\text{Total operations}}{\text{Performance}} = \frac{10^{15}}{10^{10}} = 10^5 \,\text{seconds} \approx 27 \,\text{hours.}$$

Thus, training a Gaussian Process on this dataset would take approximately 27 hours on a standard CPU.

The memory requirement of 80 GB is something that couldn't be found easily in computers. Therefore, in the next subsection, we try to efficiently sample from our 100,000-record dataset that represents the dataset as much as possible, so we can perform GP and test different kernels and assess their performance.

3.2.3 Sampling from the Dataset

To efficiently reduce the size of the original dataset while preserving its distribution and diversity, several sampling methods were considered (Tan et al., 2006). Random sampling offers a simple and fast approach but often fails to capture the variability or structure of the dataset, resulting in subsets that may not represent the original data well. Stratified sampling ensures representation across predefined groups, but it requires prior knowledge of these strata, which is challenging in high-dimensional datasets. Importance sampling focuses on selecting data points with high variance or relevance; however, this approach risks losing overall diversity and relies on domain-specific insights or prior models.

Among these options, clustering-based sampling emerged as the most effective method. By grouping data points based on feature similarity, clustering captures the underlying structure of the dataset without requiring prior assumptions about its organization. Additionally, clustering ensures that all regions of the feature space are proportionally represented, providing a diverse subset that maintains the original dataset's characteristics.

To perform clustering effectively, the features of the dataset were first standardized using z-score normalization (Grus, 2015). This ensured that all variables contributed equally to the clustering process, which is essential for algorithms like K-Means that rely on distance metrics. The z-score normalization is defined as:

$$z = \frac{x - \mu}{\sigma}$$

where x is the feature value, μ is the mean, and σ is the standard deviation of the feature. This step eliminated biases introduced by differences in feature scales while preserving the data's integrity. The target variable, `max_queue_length`, was not modified as it was not directly used in clustering.

The optimal number of clusters (k) was determined using two metrics: the Elbow Method (Thorndike, 1953) and the Silhouette Score (Rousseeuw, 1987). The Elbow

Method evaluates the inertia, defined as the sum of squared distances between data points and their nearest cluster center:

$$\text{Inertia} = \sum_{k=1}^{K} \sum_{z_i \in C_k} \|z_i - c_k\|^2$$

Where: K is the number of clusters. C_k is the set of data points assigned to the k-th cluster. z_i represents a scaled data point (after z-score normalization), and c_k is the centroid of the k-th cluster. By plotting inertia against k, the "elbow point" was identified as the value of k where additional clusters provided diminishing returns in inertia reduction.

The Silhouette Score measures the quality of clustering by assessing how similar each data point is to its own cluster compared to other clusters. It is defined as:

$$S = \frac{1}{n} \sum_{i=1}^{n} \frac{b_i - a_i}{\max(a_i, b_i)}$$

where a_i is the average distance between $\mathbf{x}_i$ and all other points in its cluster, and b_i is the average distance between $\mathbf{x}_i$ and the nearest cluster to which it does not belong. The score ranges from -1 to 1, with higher values indicating better-defined clusters. Based on these metrics shown in Figure 6 and Figure 7, $k = 5$ was chosen as the optimal number of clusters.

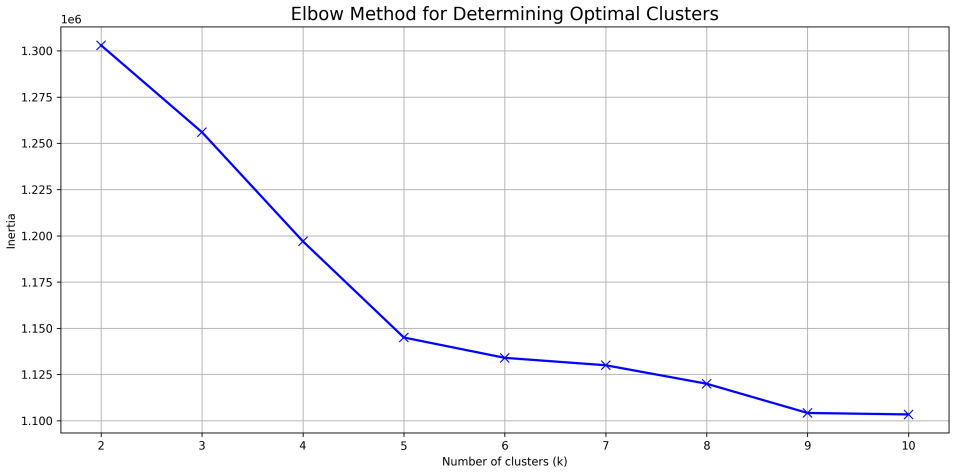

Figure 6: The Elbow Method illustrates the relationship between the number of clusters and the inertia. The optimal k is identified at the point where the reduction in inertia starts to diminish significantly, forming an 'elbow' in the curve.

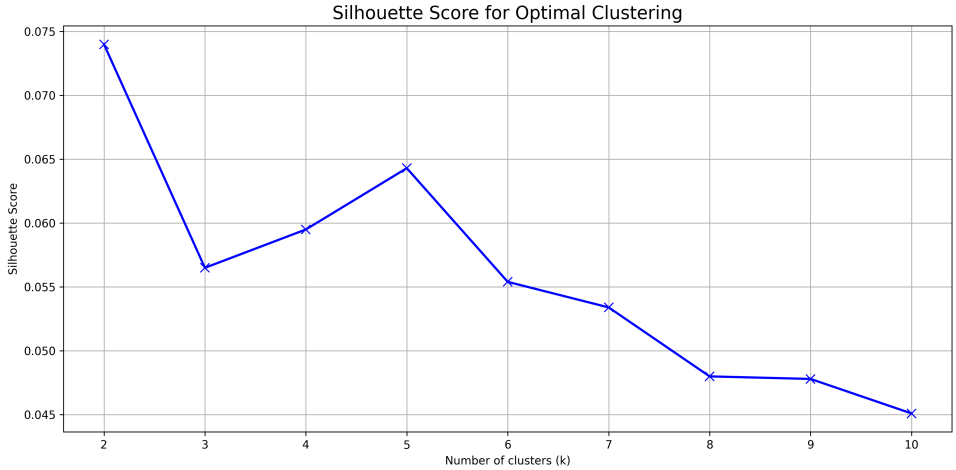

Figure 7: Silhouette scores for varying numbers of clusters. The silhouette score measures the quality of clustering, with higher scores indicating better-defined clusters.

Once k was determined, K-Means clustering was applied (MacQueen, 1967). The algorithm was initialized with $k = 5$. Each data point was assigned a cluster label, and these labels were appended to the original dataset. The objective of K-Means is to minimize the within-cluster sum of squares (WCSS):

$$\text{WCSS} = \sum_{j=1}^{k} \sum_{i \in C_j} \|\mathbf{z}_i - \mathbf{c}_j\|^2$$

where C_j is the set of data points in cluster j, and $\mathbf{c}_j$ is the centroid of cluster j.

To create a representative sample from the clusters, the size of each cluster and its proportion relative to the total dataset were computed. Sampling within each cluster was performed based on these proportions to preserve the overall distribution of the data. This ensured that the sampled subset maintained both the diversity and the balance of the original dataset.

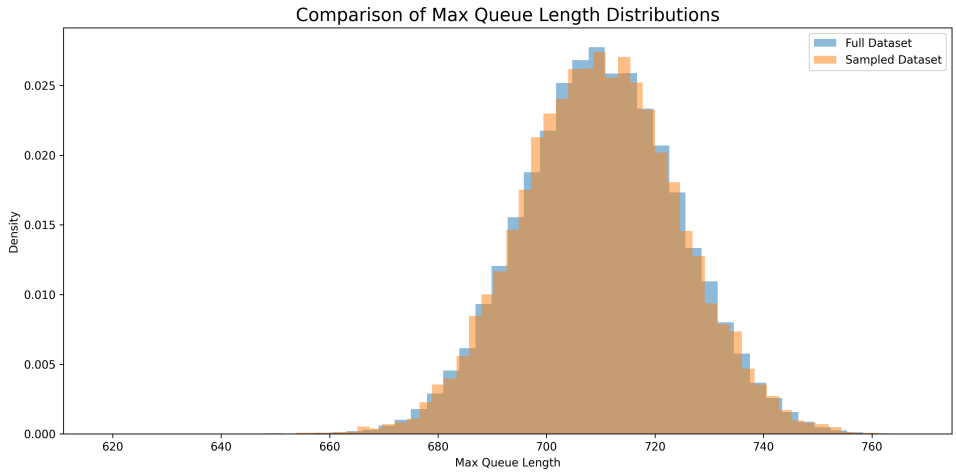

Figure 8: Overlaid density histograms of Q_{max} for the full dataset and the sampled subset.

Validation of the sampled subset was conducted by comparing the distribution of the target variable (Q_{max}) between the original dataset and the sampled subset in Figure 8. The histograms were normalized to show density, ensuring comparability despite differences in dataset size. This visualization confirmed that the sampled subset retained the overall shape and variability of the original dataset, making it suitable for subsequent modeling and analysis. Overall, 10000 records were smapled from the original dataset using this method.

3.2.4 Approaches to Mitigate Complexity

Although we sampled from our dataset using clustering methods, but still the memory requirement plays a key role in restricting us checking different models. To address this limitation, several techniques exist to reduce the computational burden of GPs:

- **Sparse Gaussian Process:** This method (Quiñonero-Candela & Rasmussen, 2005) approximate the full covariance matrix by using a subset of the training data, significantly reducing the computational cost while maintaining reasonable accuracy. For instance, the use of inducing points reduces the complexity to $\mathcal{O}(m^2 n)$, where m is the number of inducing points and $m \ll n$.
- **Structured Kernel Interpolation (SKI):** This approach (Wilson & Nickisch, 2015) leverages kernel approximations to reduce complexity to $\mathcal{O}(n \log n)$, making it feasible to work with larger datasets.
- **Mini-Batch Training and Approximation Methods:** Techniques like stochastic gradient descent (SGD) and variational inference are used to train GPs in batches, allowing them to scale to larger datasets (Chen et al., 2021).

In this project we leveraged Sparse GP and mini-batch training. SKI assumes that the inducing points lie on a grid, which can be restrictive for high-dimensional or irregularly distributed data. Additionally, SKI heavily relies on specific kernel approximations, limiting its applicability in cases where non-stationary kernels or other custom kernel functions are required. Sparse GPs, on the other hand, allow inducing points to be placed freely, providing greater flexibility and adaptability to the underlying structure of the dataset.

The central concept behind Sparse GPs is to approximate the full covariance matrix using a smaller set of inducing variables. These inducing variables are a subset of the latent function values, corresponding to selected input locations X_m, which summarize the information from the full dataset. Mathematically, the approximate posterior in Sparse GPs can be expressed as:

$$q(f_*) = \int p(f_*|f_m)\phi(f_m)\, df_m,$$

where f_m are the inducing variables, $\phi(f_m)$ is their approximate posterior, and $p(f_*|f_m)$ is the conditional distribution of the test predictions given the inducing variables. The optimal inducing inputs, kernel hyperparameters, and posterior distributions are determined by maximizing the Variational Evidence Lower Bound (ELBO) (Kingma & Welling, 2022) on the log marginal likelihood.

The ELBO is given by:

$$\mathcal{L} = \log \mathcal{N}(y|0, Q_{nn} + \sigma_y^2 I) - \frac{1}{2\sigma^2}\mathrm{Tr}(K_{nn} - Q_{nn}),$$

where $Q_{nn} = K_{nm}K_{mm}^{-1}K_{mn}$ is the low-rank approximation of the full covariance matrix K_{nn}, and σ_y^2 represents the noise variance. This expression captures both the data fit and a regularization term that prevents overfitting, ensuring that the inducing points are optimally positioned to capture the underlying structure of the data.

By using inducing variables and employing efficient matrix operations such as Cholesky decomposition, Sparse GPs significantly reduce memory requirements and computational cost. This enables working with large datasets without compromising the predictive performance of the model. For example, the computational cost of deriving the approximate posterior mean and covariance for a test input reduces to $O(m^2)$, making it feasible to handle practical scenarios with substantial datasets.

3.3 Implementation of Sparse Gaussian Processes

To effectively model the maximum queue length ($Q_{\max}$) and address computational constraints, following the last subsection, we implemented Sparse GPs using mini-batch training and GPU acceleration (NVIDIA RTX A2000 12GB Graphics Card). This approach allowed us to efficiently handle the dataset while exploring different kernel and mean function configurations to identify the optimal model for our problem.

3.3.1 Data Preparation

The dataset, consisting of 10,000 records after sampling, was split into training, validation, and test sets with an 80-10-10 split. The features were standardized using z-score normalization to ensure that all variables contributed equally to the model training. The target variable, $Q_{\max}$, was left unscaled to preserve its original scale and interpretability.

3.3.2 Model Configuration

We constructed a Sparse GP model with the following key components:

- **Inducing Points:** To reduce computational complexity, we selected inducing points representing 10% of the training data. These inducing points were shared across all implementations to ensure a fair comparison between models. By holding the inducing points constant, the variability in performance could be attributed solely to the models' ability to fit the data, rather than differences in the selection of inducing points.
- **Kernel Functions:** We experimented with various kernel functions to capture different aspects of the data's structure. The kernels considered include:
  - *Squared Exponential Kernel (RBF):* Suitable for smooth and continuous functions (Rasmussen & Williams, 2006).
  - *Matérn Kernels (with $\nu = 1.5$ and $\nu = 2.5$):* Capture less smooth functions and provide flexibility in modeling (Genton, 2002).
  - *Spectral Mixture Kernel:* Capable of modeling complex patterns by representing the spectral density of the data (Wilson & Adams, 2013).

  - *Combination of RBF and Linear Kernels:* To capture both linear trends and non-linear variations in the data.

- **Mean Functions:** We explored different mean functions (Rasmussen & Williams, 2006) to account for varying baseline behaviors in the data:
  - *Zero Mean:* Assumes the data is centered around zero.
  - *Constant Mean:* Learns a constant offset from zero.
  - *Linear Mean:* Models linear trends in the data.

3.3.3 Description of Kernels Used

In our experiments, we explored multiple kernel functions for their ability to capture different aspects of the data's underlying structure:

Squared Exponential Kernel (RBF): The Squared Exponential Kernel, also known as the Radial Basis Function (RBF) kernel (Rasmussen & Williams, 2006), is a commonly used kernel in Gaussian Processes. It is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{l^2}\right),$$

where σ_f^2 is the signal variance, and l is the length scale. The RBF kernel assumes that the underlying function is smooth and continuous, making it suitable for problems where the target variable changes gradually with respect to the input features. This kernel is often considered a default choice due to its simplicity and generality.

Matérn Kernels (with $\nu = 1.5$ and $\nu = 2.5$): The Matérn kernel generalizes the RBF kernel by introducing a parameter ν, which controls the smoothness of the function (Genton, 2002). It is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{\|\mathbf{x} - \mathbf{x}'\|}{l}\right)^{\nu} K_\nu \left(\sqrt{2\nu}\frac{\|\mathbf{x} - \mathbf{x}'\|}{l}\right),$$

where K_ν is the modified Bessel function of the second kind, $\nu > 0$ is the smoothness parameter, l is the length scale, and σ_f^2 is the signal variance. For $\nu = 1.5$, the kernel allows for less smooth functions with continuous first derivatives, while for $\nu = 2.5$, it allows for more smoothness with continuous second derivatives. These kernels provide flexibility in modeling functions that may not be perfectly smooth, such as those with small discontinuities or rough transitions.

Spectral Mixture Kernel: The Spectral Mixture Kernel is a flexible and expressive kernel designed to model a wide range of stationary patterns in data (Wilson & Adams, 2013). It is constructed by representing the spectral density of a GP as a mixture of Gaussian components, allowing it to capture diverse structures in the data. The kernel is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \sum_{q=1}^{Q} w_q \prod_{p=1}^{P} \exp\left(-2\pi^2 (\mathbf{x}_p - \mathbf{x}'_p)^2 \nu_q^{(p)}\right) \cos\left(2\pi (\mathbf{x}_p - \mathbf{x}'_p) \mu_q^{(p)}\right),$$

where Q is the number of mixture components, w_q are the weights of the components, $\nu_q^{(p)}$ are the length scales, and $\mu_q^{(p)}$ are the frequencies of the Gaussian components for each dimension p.

Combination of RBF and Linear Kernels: The combination of RBF and Linear kernels provides a way to model both non-linear and linear relationships in the data. The combined kernel is expressed as:

$$k(\mathbf{x}, \mathbf{x}') = k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') + k_{\text{Linear}}(\mathbf{x}, \mathbf{x}'),$$

where k_{RBF} is the squared exponential kernel, and k_{Linear} is the linear kernel defined as:

$$k_{\text{Linear}}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'.$$

This combination captures the smooth variations modeled by the RBF kernel and the linear trends modeled by the linear kernel, making it effective for datasets where both types of relationships exist.

3.3.4 Training Procedure

The model training was carried out using the following procedure:

- **Mini-Batch Training:** To handle the dataset efficiently, we implemented mini-batch training with a batch size of 64. This approach reduces memory usage and allows for parallel processing on GPUs.
- **Optimization Algorithm:** The model parameters, including kernel hyperparameters and variational parameters, were optimized using the Adam optimizer (Kingma & Ba, 2017) with learning rates of 0.01 and 0.1 to evaluate the effect of the learning rate on convergence.
- **Loss Function:** We utilized ELBO as the loss function.
- **Early Stopping:** To prevent overfitting and reduce training time, early stopping was implemented with a patience of 5 epochs. Early stopping is a popular approach to mitigate overfitting in ML theory (Girosi et al., 1995). The model's performance on the validation set was monitored, and training was halted if the validation loss did not improve over consecutive epochs.

3.3.5 Model Evaluation

After training, the models were evaluated on the validation and test sets. The primary evaluation metric was the Mean Squared Error (MSE) between the predicted $Q_{\max}$ and the true values. The best-performing model for each configuration was identified based on the lowest validation MSE, and the corresponding test MSE was recorded. The models and their performances are demonstrated in Table 5.

Table 5: Performance of Sparse GP Models with Different Configurations.

Kernel	Mean Function	Learning Rate	Epoch Early Stopping	Validation MSE	Test MSE
RBF	ZeroMean	0.01	100	49.6222	52.2798
RBF	ZeroMean	0.1	81	24.4911	31.4375
RBF	ConstantMean	0.01	100	47.0436	48.1311
RBF	ConstantMean	0.1	53	34.2235	36.8142
RBF	LinearMean	0.01	100	46.1212	47.8172
RBF	LinearMean	0.1	77	24.8992	29.4812
Matern32	ZeroMean	0.01	93	19.1730	22.4384
Matern32	ZeroMean	0.1	35	21.9726	30.8018
Matern32	ConstantMean	0.01	95	19.3227	21.8539
Matern32	ConstantMean	0.1	32	22.3555	26.0424
Matern32	LinearMean	0.01	96	18.8618	22.0729
Matern32	LinearMean	0.1	54	20.3002	24.0517
Matern52	ZeroMean	0.01	100	17.6796	20.9039
Matern52	ZeroMean	0.1	40	21.4827	24.7209
Matern52	ConstantMean	0.01	100	17.5265	21.2686
Matern52	ConstantMean	0.1	39	20.7177	23.7426
Matern52	LinearMean	0.01	96	18.1505	21.5263
Matern52	LinearMean	0.1	40	20.7336	29.2265
SpectralMixture	ZeroMean	0.01	7	503320.2930	504899.4316
SpectralMixture	ZeroMean	0.1	6	503320.7090	504899.4473
SpectralMixture	ConstantMean	0.01	100	325304.0938	326573.5176
SpectralMixture	ConstantMean	0.1	90	199.9836	200.9489
SpectralMixture	LinearMean	0.01	100	324052.4238	324165.1387
SpectralMixture	LinearMean	0.1	100	17.2088	20.8141
RBF+Linear	ZeroMean	0.01	31	603.2171	676.5113
RBF+Linear	ZeroMean	0.1	34	73.6373	98.9554
RBF+Linear	ConstantMean	0.01	10	17.4540	21.1662
RBF+Linear	ConstantMean	0.1	19	16.8815	20.9214
RBF+Linear	LinearMean	0.01	10	17.1631	20.7619
RBF+Linear	LinearMean	0.1	15	17.5727	21.8462

The results of the experiments demonstrated that the choice of kernel and mean function significantly influenced the model's performance. Models using the Spectral Mixture Kernel and the combination of RBF and Linear Kernels generally achieved superior results, indicating that the ability to model complex, multi-scale patterns and trends is crucial for accurately predicting $Q_{\max}$. The Linear Mean function also contributed to better performance, suggesting the presence of underlying linear relationships in the data. Additionally, the combination of the Linear and RBF kernels substantially enhanced performance compared to using only the RBF kernel, illustrating that mixing kernels can capture diverse characteristics of the data more effectively.

An important observation during training was the tendency of some models, particularly those using the Spectral Mixture Kernel, to underfit when a smaller learning rate (0.01) was employed. This underfitting likely arose from the model's inability to effectively explore the parameter space within the training epochs, leading to suboptimal convergence. Conversely, the RBF + Linear models exhibited faster convergence, even with smaller learning rates, highlighting their efficiency in adapting to the under-

lying data structure. Furthermore, across most configurations—except in cases of clear underfitting—there was no evidence of overfitting, as the MSE values for the validation and test sets remained close, demonstrating strong generalization capabilities.

To contextualize these MSE values, it is important to note that $Q_{\max}$ in the sampled dataset ranges between 649 and 763, with a mean of 709.96 and a standard deviation of 14.51. This implies that an MSE around 17 corresponds to predicting $Q_{\max}$ with an average error of approximately four units. This level of accuracy is highly satisfactory for the problem at hand, as it provides reliable estimates of the maximum queue length.

Given these results, we further analyzed the top three models with the best MSE performance on the validation set. These models, in order of their performance, are:

- `RBF + Linear` kernel with `ConstantMean` mean function and learning rate (lr) of 0.1.
- `RBF + Linear` kernel with `LinearMean` mean function and lr of 0.01.
- `SpectralMixture` kernel with `LinearMean` mean function and lr of 0.1.

To ensure the robustness of these models and verify that their performance was not merely due to randomness, each configuration was implemented 10 times, using the same set of inducing points across all repetitions to reduce variance. The goal was to assess the stability of their performance metrics. The mean ($\hat{\mu}$) and standard deviation ($\hat{\sigma}$) of MSE_{val} for these models are as follows:

- `RBF + Linear` + `ConstantMean` + lr $= 0.1$: $\hat{\mu} = 17.54$, $\hat{\sigma} = 0.96$.
- `RBF + Linear` + `LinearMean` + lr $= 0.01$: $\hat{\mu} = 17.72$, $\hat{\sigma} = 0.71$.
- `SpectralMixture` + `LinearMean` + lr $= 0.1$: $\hat{\mu} = 20.22$, $\hat{\sigma} = 6.35$.

Interestingly, the Spectral Mixture model demonstrated a significantly higher variance in MSE, making it less robust compared to the RBF + Linear models. This variability may stem from the model's sensitivity to initialization or its complexity in parameter optimization. On the other hand, the first two models exhibited consistently low variance and similar performance levels, indicating both stability and reliability. Thus, the RBF + Linear models with appropriate mean functions emerge as the most robust and effective configurations for predicting $Q_{\max}$.

Figure 9 illustrates the predicted values versus the actual values of Q_{max} for the first model (`RBF + Linear` kernel with `ConstantMean` mean function and lr=0.1). The red dashed line represents the ideal perfect prediction, where predicted values equal actual values. Most data points lie close to this line, indicating that the model performs well in predicting Q_{max}. Marginal histograms along the axes further show the distributions of the predicted and actual values, confirming the alignment between the two distributions.

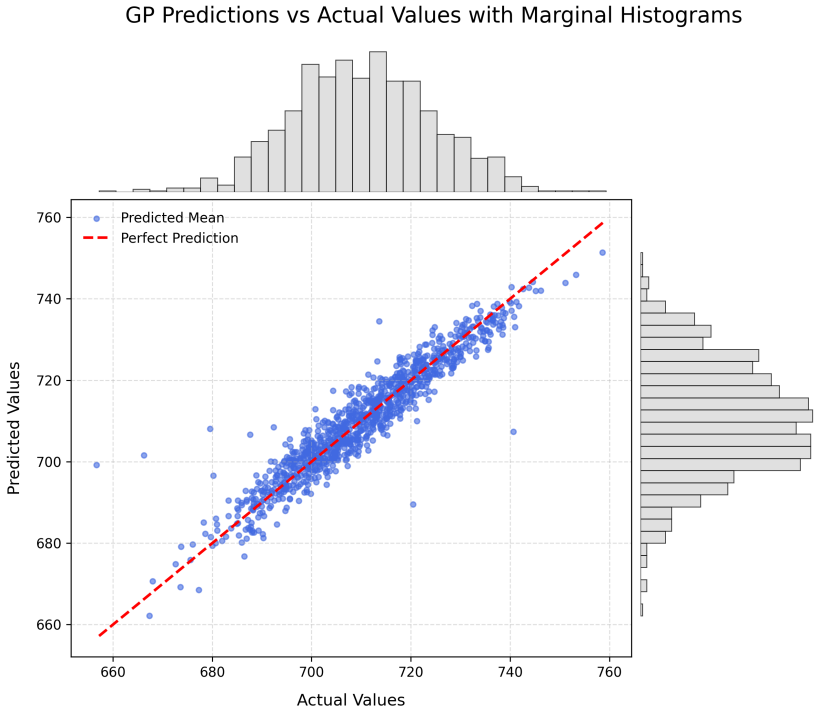

Figure 9: Scatter plot of predicted versus actual values for Q_{max} , with marginal histograms showing the respective distributions. The red dashed line represents perfect predictions.

3.4 Conclusion

The best-performing model in the Sparse GP experiments was identified as the combination of the `RBF + Linear` kernel with the `ConstantMean` mean function. This result is particularly insightful when interpreted in the context of the queue system being modeled. The success of this model stems from its ability to capture both linear trends and non-linear variations in the data effectively.

The `RBF` kernel captures the smooth, non-linear dependencies inherent in the system, such as the interaction between station workloads and queue dynamics. Meanwhile, the addition of the `Linear` kernel accounts for the underlying linear relationships in the queueing system, such as the proportional effects of increasing operators or processing times on $Q_{\max}$. Together, these kernels provide a balanced representation of the system, which exhibits both smooth variations in queue length across different operating conditions and direct, linear effects of key features like staffing and arrival rates.

The `ConstantMean` function further contributes to the model's performance by providing a flexible yet stable baseline prediction that is well-suited for the overall distribution of $Q_{\max}$. This stability likely helped the model generalize better across the validation and test datasets, as reflected in its consistently low MSE and robustness during repeated experiments. Now that we have captured our best model, we resume to use this knowledge as a surrogate in our Bayesian Optimization.

4 Bayesian Optimization for Multi-Objective Problems

Bayesian Optimization (BO) is a probabilistic framework designed to optimize objective functions that are expensive to evaluate, non-convex, or black-box in nature (Garnett, 2023). By combining surrogate modeling with acquisition functions, BO provides an efficient mechanism for identifying optimal solutions while minimizing the number of function evaluations. The surrogate model, often a GP, serves as a probabilistic representation of the objective function, capturing uncertainty and guiding the search process. The acquisition function, in turn, balances the trade-off between exploration of uncertain regions and exploitation of known promising areas.

For systems like ours, where the objectives involve balancing complex trade-offs such as minimizing $Q_{\max}$ while simultaneously reducing operational costs, BO becomes a natural choice. In such multi-objective settings, the goal shifts from identifying a single optimal point to discovering a Pareto front—a set of non-dominated solutions where no objective can be improved without compromising another.

Multi-Objective Bayesian Optimization (MOBO) extends the principles of single-objective BO to handle multiple objectives (Garnett, 2023). This is achieved by using surrogate models for each objective and acquisition functions designed to optimize the Pareto front. MOBO is particularly well-suited for scenarios like ours, where evaluating objectives (e.g., simulating $Q_{\max}$) is computationally expensive and requires careful allocation of resources to explore the decision space effectively.

In this section, we detail the implementation of MOBO for our system, leveraging the GP model developed in the previous section as the surrogate for the primary objective, $Q_{\max}$. The cost of operations is incorporated as a secondary objective, enabling the discovery of efficient trade-offs between queue management and cost minimization.

4.1 Costs Considered in the Optimization Problem

To effectively optimize the system using MOBO, it is essential to incorporate a realistic cost function. Without considering costs, the optimization process could enhance the system indefinitely, disregarding practical constraints. In our setup, costs are associated with two primary factors: (1) enhancing the service time of operators or machines at various stations, and (2) increasing the number of operators in stations B, D, and E (remember that it was assumed that stations A and C have fixed number of machines). These costs are evaluated against a baseline configuration representing the current, non-optimal behavior of the system.

4.1.1 Baseline Configuration

The baseline configuration assumes the following:

- Stations B, D, and E each have 2 operators.
- The mean service times at all stations follow a normal distribution with the following parameters:

$$\mu = [10, 15, 16, 18, 20] \quad \text{and} \quad \sigma = 2\,\text{minutes}, \quad \text{for Stations A to E, respectively.}$$

4.1.2 Bounds for System Parameters

To ensure realistic optimization and prevent indefinite system enhancement, lower and upper bounds are defined for the decision variables:

- **Number of Operators:** The lower bound (lb) is set to 2 and the upper bound (ub) is set to 6 for stations B, D, and E.
- **Mean Service Times:** The bounds for the mean service times (μ) are as follows:

  $\mu_{\text{lb}} = [4, 7, 10, 10, 7]$ and $\mu_{\text{ub}} = [10, 15, 16, 18, 20]$ for Stations A to E, respectively.

  The upper bounds reflect the current performance levels, ensuring operators and machines cannot perform worse than their baseline configuration on average.

4.1.3 Cost Functions

The cost functions are designed to capture the trade-offs associated with system improvements:

Service Time Cost: The cost of improving the mean service time at a station is calculated based on the reduction in service time ($\Delta\mu_i$), a base cost coefficient (a_i), and a growth rate parameter (b_i) for the station:

$$\Delta\mu_i = \mu_{i,\text{base}} - \mu_i, \quad \text{where } \mu_{i,\text{base}} \text{ is the baseline service time.}$$

The cost is proportional to this reduction and reflects the additional effort or resources required to improve operator or machine efficiency. It is calculated as $f_i(\Delta\mu_i) = a_i \exp(b_i \Delta\mu_i)$

Operator Count Cost: The cost of increasing the number of operators in stations B, D, and E is modeled as a linear function:

$$\text{Cost} = c_i \cdot (\text{new number of operators} - \text{baseline number of operators}),$$

where c_i is the cost coefficient per additional operator for station i. This linear relationship accounts for the incremental costs of hiring or allocating additional personnel to these stations.

4.1.4 Fixed Interarrival Distributions

For the optimization phase, the interarrival times for standard and urgent orders are fixed as follows:

$$\texttt{interarrival\_time\_config\_BO} = \begin{cases} \text{Morning:} & \text{ordinary: } \text{Exp}(\lambda = 17), \quad \text{priority: } \text{Exp}(\lambda = 15), \\ \text{Afternoon:} & \text{ordinary: } \text{Exp}(\lambda = 20), \quad \text{priority: } \text{Exp}(\lambda = 25), \\ \text{Night:} & \text{ordinary: } \text{Exp}(\lambda = 10), \quad \text{priority: } \text{Exp}(\lambda = 14). \end{cases}$$

This fixed distribution ensures consistency in the arrival rates during the optimization process, allowing the cost and performance evaluations to focus on the decision variables (service times and operator counts).

4.1.5 Setup Summary

In summary, the lower bound and upper bound of the number of operators, average service time (in minutes), parameters of cost functions are detailed in table 6.

Table 6: Constraints and Cost Parameters for Operators and Service Times at Stations

Station	lb_operators	ub_operators	lb_ST	ub_ST	a_ST	b_ST	c_operators
A	-	-	4	10	10	0.1	-
B	2	6	7	15	20	0.3	20
C	-	-	10	16	30	0.3	-
D	2	6	10	18	40	0.5	25
E	2	6	7	20	15	0.2	10

4.2 Multi-Objective Bayesian Optimization (MOBO)

Multi-Objective Bayesian Optimization (MOBO) is a framework designed to optimize multiple conflicting objectives simultaneously, particularly when evaluating these objectives is computationally expensive. Unlike single-objective optimization, MOBO focuses on identifying the Pareto front—a set of optimal trade-offs where no objective can be improved without worsening another. GPs are commonly employed as surrogate models in MOBO, providing both predictive estimates of objective functions and associated uncertainties, which are then leveraged by acquisition functions to guide the optimization process.

4.2.1 Acquisition Functions in MOBO

Acquisition functions play a pivotal role in MOBO, balancing exploration (sampling uncertain areas to improve the model) and exploitation (focusing on regions with promising solutions). In our implementation, we used two advanced acquisition functions: *qNEHVI (q-Expected Hypervolume Improvement)* (Daulton et al., 2021) and *qNParEGO (Pareto-based Weighted Aggregation)* (Knowles, 2006).

The qNEHVI acquisition function (Daulton et al., 2021) extends the concept of Expected Hypervolume Improvement (EHVI) (Daulton et al., 2020) to batch optimization, enabling the simultaneous evaluation of multiple candidate solutions. Hypervolume is a key measure in multi-objective optimization, representing the volume of the objective space dominated by the Pareto front. The improvement in hypervolume (ΔHV) for a candidate point x is given by:

$$\Delta\text{HV} = \text{HV}(\text{Pareto} \cup f(x)) - \text{HV}(\text{Pareto}),$$

where $f(x)$ is the vector of objectives for x, and Pareto is the current Pareto front. qNEHVI evaluates the expected contribution of a batch of points to the hypervolume improvement, prioritizing solutions that expand the Pareto front efficiently.

qNEHVI is particularly effective in refining the Pareto front by selecting points that maximize the hypervolume dominated by Pareto-optimal solutions. Its integration of

uncertainty through the GP posterior ensures robust exploration and exploitation, even in noisy or high-dimensional settings.

The qNParEGO acquisition function builds upon the ParEGO framework, which scalarizes multiple objectives into a single objective using an augmented Tchebycheff function:

$$\phi(\mathbf{x}, \mathbf{w}) = \max_i [w_i | f_i(\mathbf{x}) - z_i^* |] + \rho \sum_i w_i | f_i(\mathbf{x}) - z_i^* |,$$

where $\mathbf{w}$ is a weight vector sampled to emphasize different trade-offs between objectives, z_i^* represents the ideal point in the objective space, and $\rho > 0$ ensures smoothness. qNParEGO leverages these scalarized objectives to approximate the Pareto front effectively, ensuring diverse exploration of the objective space. qNParEGO is particularly advantageous in the early stages of optimization when the Pareto front is sparse (Knowles, 2006).

4.3 Implementation of MOBO for Optimizing Queue Systems

To effectively implement the MOBO framework for our problem, the input data was first scaled using z-score normalization to ensure all features contributed proportionally during optimization. As the optimization process involves minimization of conflicting objectives, lower bounds were set for each objective function to avoid unrealistic solutions. The Gaussian Process model employed for modeling the objectives utilized the kernel and mean function combination identified in previous sections (*RBF + Linear Kernel* with *Constant Mean*).

The MOBO framework was initialized with 50 observations to establish an initial understanding of the objective space. Subsequently, a batch size of 16 was used for each epoch during optimization. The two algorithms implemented, *qNEHVI* and *qNParEGO*, were each run for 20 epochs. To evaluate the robustness of the methods and to account for the possibility of converging to local optima, the optimization process was repeated three times for each algorithm.

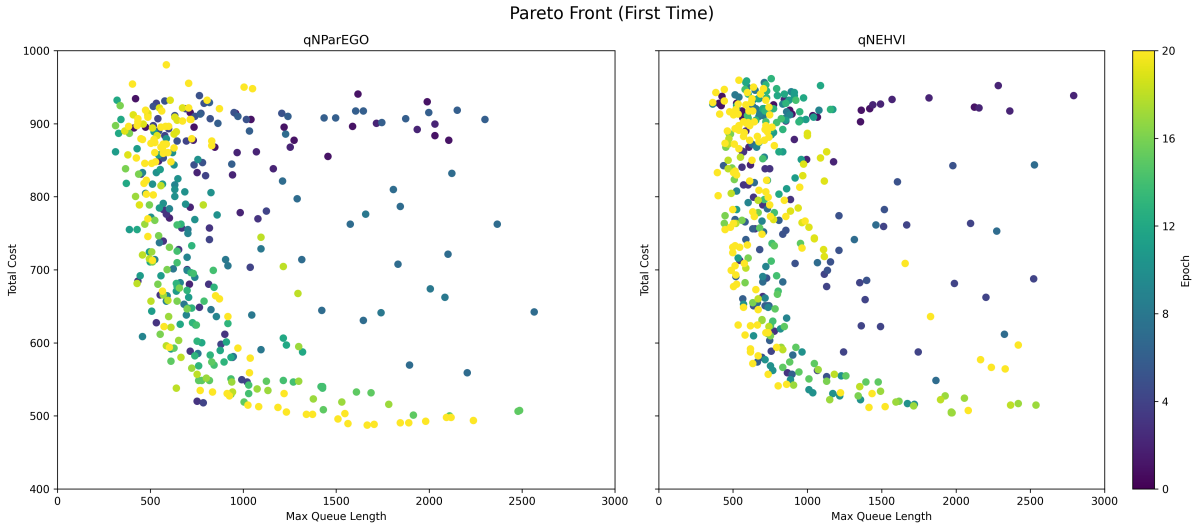

Figure 10: Pareto Front for the first run of qNEHVI and qNParEGO. The plots demonstrate the trade-off between the two objectives: minimizing *Total Cost* and $Q_{\max}$.

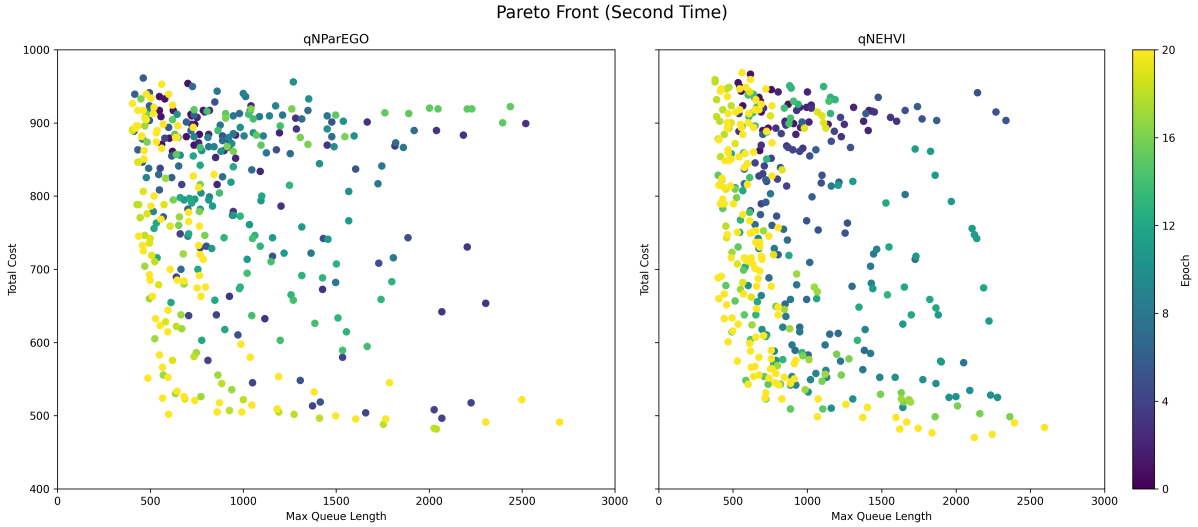

Figure 11: Pareto Front for the second run of qNEHVI and qNParEGO.

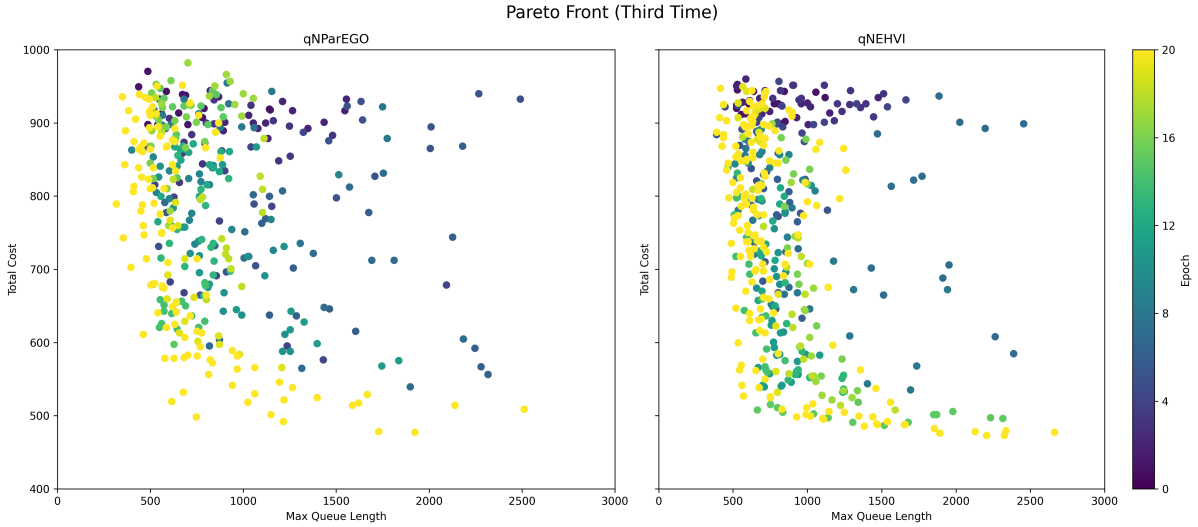

Figure 12: Pareto Front for the third run of qNEHVI and qNParEGO.

The results in Figure 10, Figure 11, and Figure 12 illustrate the Pareto fronts obtained across different runs for both algorithms. Each point on the Pareto front represents a feasible trade-off between the two objectives: *Total Cost* and $Q_{\max}$. The color intensity indicates the epoch at which a point was discovered, revealing the evolution of the optimization process.

The qNEHVI algorithm exhibited a more consistent and dense Pareto front, capturing a broader range of trade-offs compared to qNParEGO. In contrast, qNParEGO often concentrated on certain regions, reflecting its inclination to favor particular trade-offs. Notably, the Pareto fronts showed a clear improvement in solutions over successive epochs, with the algorithms progressively identifying better compromises between minimizing $Q_{\max}$ and Total Cost. Another noteworthy observation is that the majority of points for both algorithms are clustered in the upper-left corner of the plots. This is primarily due to the distribution of our objective functions, which are concentrated in that region, rather than being a limitation of the algorithms themselves. Consequently, most of the sampled points fall within this area, resulting in a Pareto frontier that is more densely

populated and defined with greater confidence in this region.

4.4 Final Results and Optimal Configuration

After thoroughly analyzing the results, we selected the qNEHVI acquisition function as the most suitable approach for our multi-objective optimization. This decision was based on its ability to effectively balance both objectives and consistently generate a robust Pareto front. To determine the optimal configuration, we imposed a trade-off with weights of 0.4 on total cost and 0.6 on $Q_{\max}$, reflecting the priority of minimizing queue length while maintaining reasonable costs.

The resulting configurations for the number of operators and average service times at the stations are listed and their corresponding objective functions is demonstrated in Table 7. Each configuration represents a viable solution, with slight variations in the the number of operators assigned to each station and enhancement of their service time.

Table 7: Summary of the Optimal Number of Operators and Average Service Time in each of the Stations with Their Corresponding Total Cost and Maximum Queue Length

#opps B	#opps B	#opps B	ST A	ST B	ST C	ST D	ST E	Total Cost	Q_{max}
3	3	5	4.15	12.83	15.37	17.38	13.55	273.2	507.6
3	5	3	4.04	14.14	15.53	17.16	14.44	285.99	514.1
4	5	3	4.43	14.08	15.09	17.57	15.3	291.86	511.3

In the next section, using the optimal configuration obtained, we use variance reduction method to reduce the variance of our prediction in the maximum queue length.

5 Variance Reduction Methods

Variance reduction techniques are essential in simulation and optimization problems to achieve more precise and reliable estimates without significantly increasing computational cost. These methods focus on reducing the variance of the estimator while preserving its unbiasedness, thereby ensuring faster convergence and more stable predictions. Variance reduction is particularly crucial when dealing with stochastic systems, where variability in outputs can obscure the true effects of input changes (Ross, 2002).

Several widely-used variance reduction methods include:

- **Control Variates:** This method exploits correlation with auxiliary variables whose expectations are known. By adjusting estimates based on these variables, the variance of the estimate is significantly reduced.
- **Antithetic Variates:** Negative correlation is introduced between paired simulations, ensuring that the fluctuations in one simulation are counterbalanced by its pair.

- **Common Random Numbers:** Synchronizing random inputs across compared systems helps isolate the true effects of system changes by minimizing noise shared between simulations.

In this project, variance reduction was implicitly applied during the training of Sparse GPs. By using shared inducing points across different kernel and mean function combinations, we minimized variability in model evaluations. This strategy allowed us to isolate the impact of kernel and mean function choices on performance, ensuring fair comparisons and robust model selection.

Building upon these principles, we now explicitly apply variance reduction techniques to the evaluation of the $Q_{\max}$ under the optimal configuration derived from MOBO. By reducing the variance in $Q_{\max}$, we aim to enhance the reliability and interpretability of our predictions, ultimately supporting more effective decision-making for system improvements.

5.1 Variance Reduction Using Maximum Waiting Time

In addition to the maximum queue length ($Q_{\max}$), one of the metrics extracted from our simulator is the maximum waiting time ($W_{\max}$). Intuitively, these two metrics are highly related as the maximum waiting time is often a consequence of excessive queue length. For example, as the queue length increases, it leads to prolonged waiting times for subsequent entities in the system which is also supported according to Figure 13. Due to this inherent relationship, a potential high correlation between $Q_{\max}$ and $W_{\max}$ can be leveraged for variance reduction.

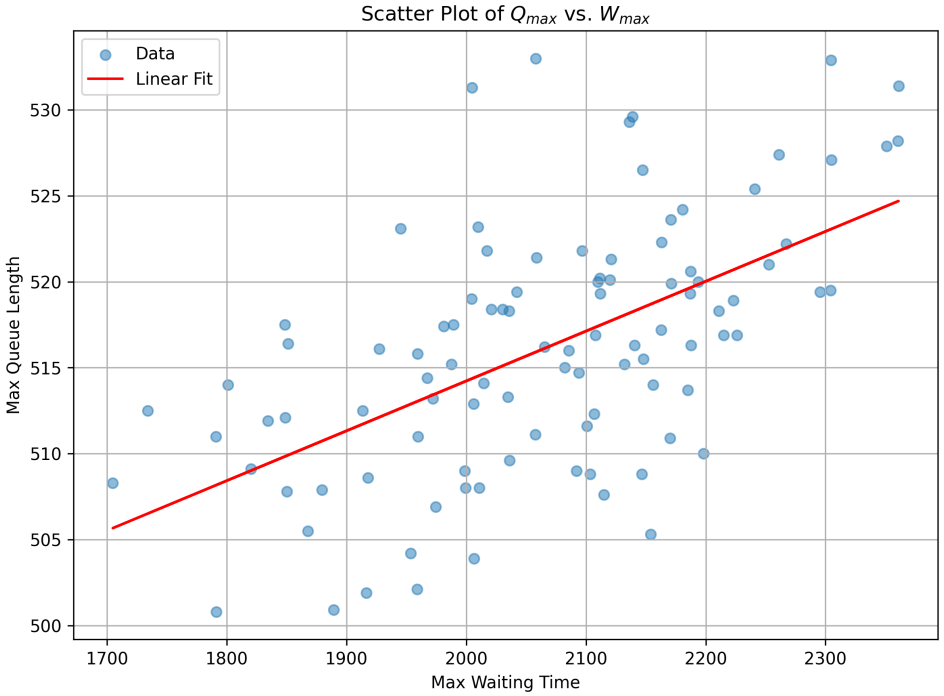

Figure 13: Scatter plot of maximum waiting time and maximum queue length

To incorporate this relationship, we use the *Control Variates* method, which adjusts

the estimator of $Q_{\max}$ using the correlation with $W_{\max}$. The formula for the adjusted estimator, Y^*, is given as:

$$Y^* = Y - c^*(Z - \mathbb{E}[Z]),$$

where:

- Y: The observed value of $Q_{\max}$,
- Z: The observed value of $W_{\max}$,
- $\mathbb{E}[Z]$: The expected value of $W_{\max}$,
- $c^* = \frac{\text{Cov}(Y,Z)}{\text{Var}(Z)}$: The optimal coefficient for variance reduction.

Using the simulation data, the correlation between $Q_{\max}$ and $W_{\max}$ was calculated as 0.5403. The optimal coefficient c^* was derived accordingly:

$$c^* = \frac{602.57}{20771.441} = 0.0290,$$

5.2 Results With and Without Variance Reduction

The variance reduction technique was applied to the simulation results, and the following metrics were computed, assuming having only 100 simulated data (the results hold and even gets better with more data on hand):

Metrics Without Variance Reduction

- Estimate of $Q_{\max}$: 516.124
- Estimated Variance ($\hat{\text{Var}}$): 53.2074
- Estimated Standard Error ($\hat{\sigma}$): 7.294
- Half-width of the Confidence Interval: 1.4296

Metrics With Variance Reduction

- Estimate of $Q_{\max}$: 516.124
- Estimated Variance ($\hat{\text{Var}}^*$): 35.726
- Estimated Standard Error ($\hat{\sigma}^*$): 5.977
- Half-width of the Confidence Interval: 1.1715

The application of the variance reduction method resulted in a significant improvement. Specifically, the half-width of the confidence interval was reduced by 18.06%, demonstrating the effectiveness of this approach in improving the precision of the estimates for $Q_{\max}$. Also as demonstrated in Figure 14, there is no sign of bias as the distribution of the original Q_{max} and its controlled version resembles each other.

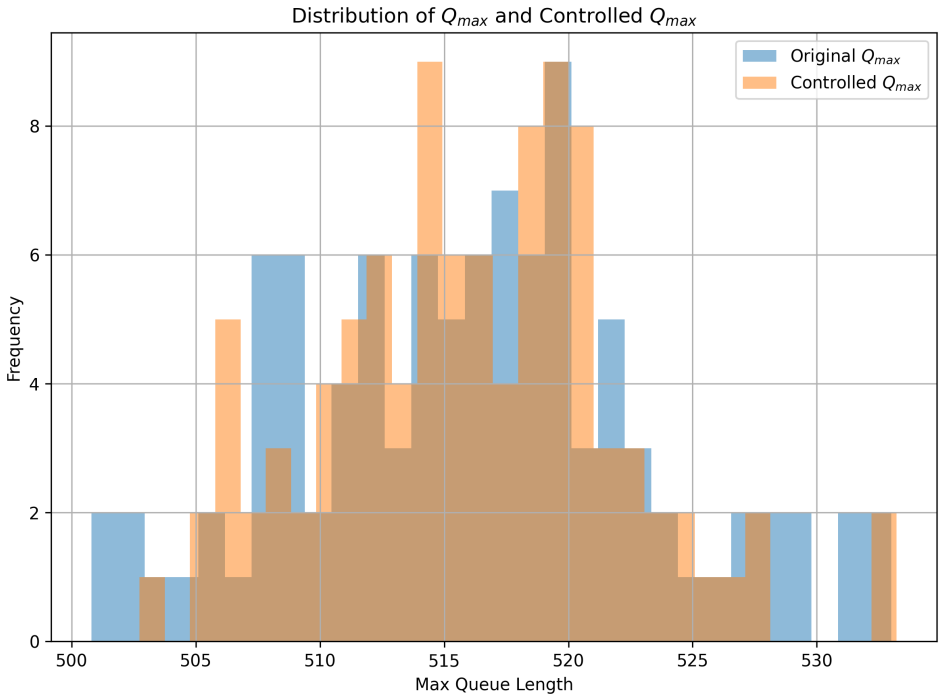

Figure 14: Distribution of maximum queue length with and without variance reduction methods

6 Limitations and Future Directions

While this study demonstrates the effective application of Gaussian Processes, Multi-Objective Bayesian Optimization, and variance reduction techniques to optimize the repair workshop system, there are several limitations and areas for improvement that merit discussion:

1. **Simplification of Queue Length to Maximum Queue Length:** The use of the $Q_{\max}$ as a summary metric effectively captures bottlenecks but overlooks the performance of individual stations. Employing a Multi-Output Gaussian Process (also known as Multi-Task Gaussian Process) could allow for the joint modeling of queue lengths across all stations. This approach would provide a more detailed understanding of system behavior, enabling more targeted optimizations and generalizing the model to diverse scenarios.

2. **Discrete Variables in MOBO:** Our current MOBO implementation treated the number of operators as continuous variables, requiring post hoc rounding to the nearest integer. This approximation introduces potential inaccuracies and inefficiencies. Adopting a MOBO variant designed to handle discrete variables directly could yield more accurate and practical solutions by naturally accounting for the integer nature of such decision variables.

3. **Uncertainty in Input Distributions:** The optimization framework assumes fixed and known input distributions for inter-arrival times and service times. In reality, these inputs may themselves be uncertain. Using Bayesian Hierarchical Models or

Input Uncertainty Propagation techniques could account for this variability and improve the robustness of predictions.

4. **Bias Toward Predefined Trade-Offs:** The optimization process relied on a fixed trade-off between cost and $Q_{\max}$. This static priority allocation may not align with real-world scenarios where decision-makers dynamically shift preferences based on changing conditions. Incorporating Preference Learning or Adaptive Weighting schemes could address this issue, allowing the system to dynamically adjust to evolving priorities.

5. **Incorporation of Real-World Constraints:** Many practical constraints, such as budget limits, workforce regulations, or physical space limitations, were not explicitly incorporated into the optimization framework. Techniques such as Constraint Bayesian Optimization (CBO) could address these factors directly, ensuring solutions are feasible in real-world settings.

7 Conclusion

This project successfully tackled the complex challenge of optimizing a repair workshop's performance under resource constraints by integrating advanced simulation, probabilistic modeling, and optimization techniques. The core of this work revolved around the prediction and minimization of the maximum queue length while balancing operational costs, ultimately addressing bottlenecks and enhancing system performance.

Gaussian Process Modeling

GPs served as the backbone for our surrogate modeling. By leveraging Sparse GPs, we addressed the computational challenges associated with large-scale datasets. Inducing points were employed to approximate the covariance matrix efficiently, significantly reducing the computational complexity while maintaining predictive accuracy. A combination of kernels, including the RBF and Linear kernels, coupled with a constant mean function, emerged as the optimal configuration. This model effectively captured both non-linear dependencies and linear trends inherent in the workshop's operations. The GP's ability to quantify uncertainty further enhanced its utility, enabling robust predictions and guiding optimization.

Multi-Objective Bayesian Optimization

Using the GP surrogate model, we implemented Multi-Objective Bayesian Optimization to balance two conflicting objectives: minimizing $Q_{\max}$ and reducing operational costs. Two advanced acquisition functions, Batch Noisy Expected Hypervolume Improvement (*qNEHVI*) and Batch Pareto Noisy Efficient Global Optimization (*qNParEGO*), were employed. *qNEHVI* demonstrated superior performance by efficiently expanding the Pareto front using expected hypervolume improvement, while *qNParEGO* effectively explored early-stage sparse Pareto fronts through scalarized objectives.

The optimization process was conducted over multiple runs and epochs, ensuring robustness and avoiding convergence to local optima. The Pareto fronts generated showcased clear trade-offs between the two objectives, with configurations reflecting diverse solutions to meet varying priorities. The results highlighted *qNEHVI* as the preferred acquisition function due to its consistent performance and dense Pareto front coverage.

Variance Reduction Techniques

To improve the reliability of predictions, variance reduction techniques were applied. Specifically, the Control Variates method leveraged the correlation (0.57) between $Q_{\max}$ and the maximum waiting time ($W_{\max}$) to reduce the variance of $Q_{\max}$ predictions. This adjustment significantly reduced the half-width of the confidence interval by 18.06%, providing more precise and stable estimates. Additionally, variance reduction was implicitly incorporated during the training of Sparse GPs by using shared inducing points across different configurations, ensuring robust model evaluation.

References

Chen, H., Zheng, L., Kontar, R. A., & Raskutti, G. (2021). Gaussian process inference using mini-batch stochastic gradient descent: Convergence guarantees and empirical benefits. https://arxiv.org/abs/2111.10461

Daulton, S., Balandat, M., & Bakshy, E. (2020). Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. https://arxiv.org/abs/2006.05078

Daulton, S., Balandat, M., & Bakshy, E. (2021). Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. https://arxiv.org/abs/2105.08195

Ebden, M. (2015). Gaussian processes: A quick introduction. https://arxiv.org/abs/1505.02965

Garnett, R. (2023). *Bayesian Optimization*. Cambridge University Press.

Genton, M. G. (2002). Classes of kernels for machine learning: A statistics perspective. *J. Mach. Learn. Res.*, *2*, 299–312.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural Computation*, *7*(2), 219–269. https://doi.org/10.1162/neco.1995.7.2.219

Grus, J. (2015). *Data science from scratch*. O'Reilly Media.

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980

Kingma, D. P., & Welling, M. (2022). Auto-encoding variational bayes. https://arxiv.org/abs/1312.6114

Knowles, J. (2006). Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, *10*(1), 50–66. https://doi.org/10.1109/TEVC.2005.851274

Liu, H., Ong, Y.-S., Shen, X., & Cai, J. (2019). When gaussian process meets big data: A review of scalable gps. https://arxiv.org/abs/1807.01065

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations.

Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, *6*(65), 1939–1959. http://jmlr.org/papers/v6/quinonero-candela05a.html

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. The MIT Press. http://www.GaussianProcess.org/gpml

Ross, S. M. (2002). *Simulation* (3rd). Academic Press.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65. https://doi.org/https://doi.org/10.1016/0377-0427(87)90125-7

Tan, P.-N., Steinbach, M., & Kumar, V. (2006). Data mining introduction. *People's Posts and Telecommunications Publishing House, Beijing*.

Thorndike, R. L. (1953). Who belongs in the family? *Psychometrika*, *18*(4), 267–276. https://doi.org/10.1007/BF02289263

Wilson, A. G., & Adams, R. P. (2013). Gaussian process kernels for pattern discovery and extrapolation [arXiv preprint arXiv:1302.4245]. *Proceedings of the 30th International Conference on Machine Learning*, *28*. https://arxiv.org/abs/1302.4245v3

Wilson, A. G., & Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp). https://arxiv.org/abs/1503.01057

A Appendix – Details of Discrete Event Simulation of the Repair Workshop

In this appendix, we will discuss the simulator that this project is based on. Here, it is assumed that the workshop is only composed of 1 machine for stations A and C, 3 operators for station B and E and finally 4 at station D. Moreover, we assume that the simulation only works until $T = 600$.

Variable Description

Table 8: Variables and Their Initial Values Used in the Simulation Model.

Variable	Description	Initial Value
serverA	Number of servers in workstation A	1
serverB	Number of servers in workstation B	3
serverC	Number of servers in workstation C	1
serverD	Number of servers in workstation D	4
serverE	Number of servers in workstation E	3
arr_ID	The ID of the machine entered the system	0
MTOT	Total number of machines entered between 120 till 600	0
Status[i]	Status of server i (0 = idle, 1 = busy)	0
Q[i]	Number of machines waiting in the i-th queue	0
Q_A[i]	Machines waiting in the queue of server A (ID, time, priority)	0
Q_B[i]	Machines waiting in the queue of server B (ID, time, priority)	0
Q_C[i]	Machines waiting in the queue of server C (ID, time, priority)	0
Q_D[i]	Machines waiting in the queue of server D (ID, time, priority)	0
Q_E[i]	Machines waiting in the queue of server E (ID, time, priority)	0
TotalQ[i]	Total number of machines waited in the i-th queue	0
SVR[i]	Service time of server i	0

Variable	Description	Initial Value
Tnow	Simulation clock	0
T1	Start point of collecting statistics	120
T2	Endpoint of collecting statistics	600
TWT	Total waiting time of machines	0
NF	Number of failures	0
Max_WT	Maximum waiting time	0
N[i]	Total number of served machines by server i	0
RespTime[i]	Response time (departure time - arrival time)	0
FEL	Future events (Code, Time, Priority, Rework, Machine ID)	[(0,0,1,0,1), ...]
FEL_backup	All past and future events	Empty
Demographic	Machine demographics (ID, order type, rework, arrival time)	Empty
Demographic_list	Saves all demographics in each iteration	Empty
TWT_list	Saves TWT variable in each iteration	Empty
N_list	Saves N variable in each iteration	Empty
SVR_list	Saves SVR variable in each iteration	Empty
TotalQ_list	Saves $TotalQ$ variable in each iteration	Empty
max_WT_list	Saves Max_WT variable in each iteration	Empty
FEL_total	Saves FEL_backup variable in each iteration	Empty
MTOT_list	Saves $MTOT$ variable in each iteration	Empty
rcounter	Counter for using random numbers	0

Modeling

State variables: $(Q_A, Q_B, Q_C, Q_D, Q_E, \text{Status}(1), \text{Status}(2), \ldots, \text{Status}(12))$

Q_i: Information about machines that are waiting in the i-th station's queue.

$$Q_i[j] = (\text{ID}_{(j\text{-th part in the queue})}, \text{Arrival Time}, \text{priority or not}, \text{rework or not})$$

Status(i): Status of server i.

$$\text{Status}(i) = \begin{cases} 0 & \text{if server } i \text{ is idle,} \\ 1 & \text{if server } i \text{ is either busy or down (for server 4 in station C).} \end{cases}$$

Events:

- **Arrival Event:** Code 0
- **Departure From A:** Code 1
- **Departure From B:** Codes 2, 3, 4
- **Departure From C:** Code 5
- **Departure From D:** Codes 6, 7, 8, 9
- **Departure From E:** Codes 10, 11, 12

Controller

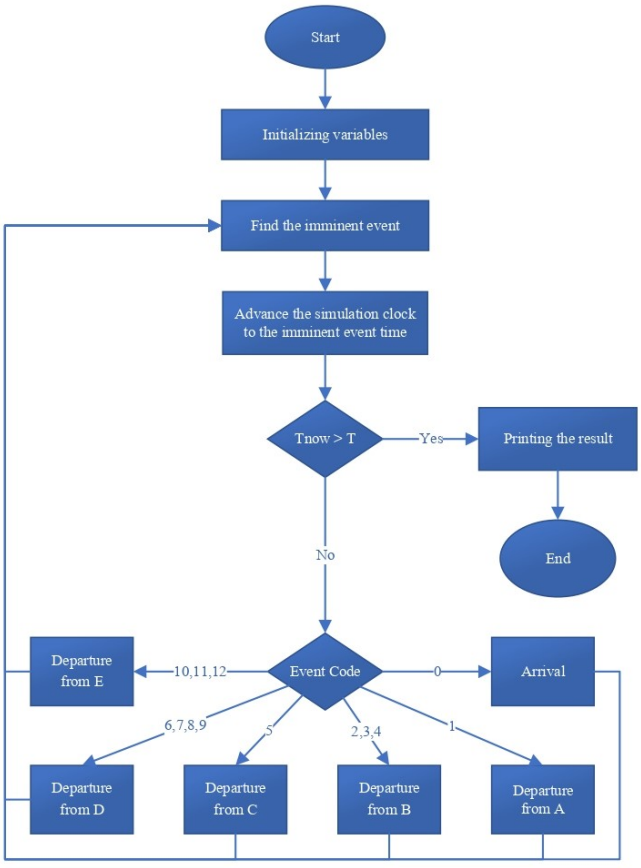

Figure 15: Controller of the Simulator

Arrival Event

Since we have two types of orders (standard and urgent) and the distribution of their inter-arrival time differs from each other, we have to compute their inter-arrival time independently (a* and b*). We first check the `code_type`; 0 means that the imminent event is a standard order and due to this, we compute the arrival time of the next standard order, 1 means the imminent event is an urgent order and due to this we compute the arrival time of the next urgent order. If the server in A is busy, we add the machine to `Q_A` otherwise, we assign the machine to the server.

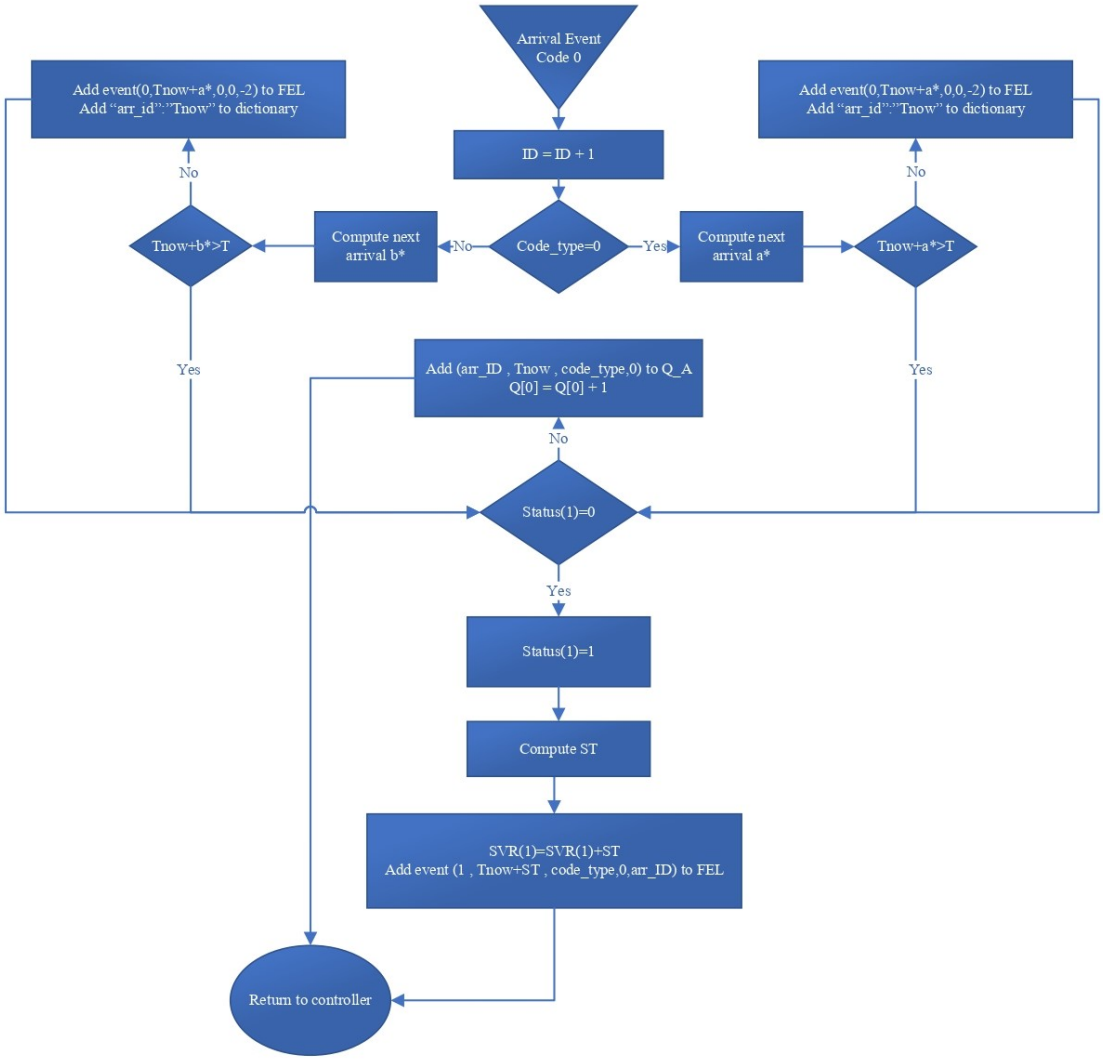

Figure 16: Flowchart of arrival event

Departure From A

We have two types of *departure from A* event; first, for the machines that were sent to rework from station D for which we should not check `Q_A` for the next arrival, we just have to check the status of servers in station B and so on.

Second for the machines that have immediately departed from station A for which we

should check the `Q_A` to assign the next machine (due to the priority) to station A's server and after that check the status of servers in station B.

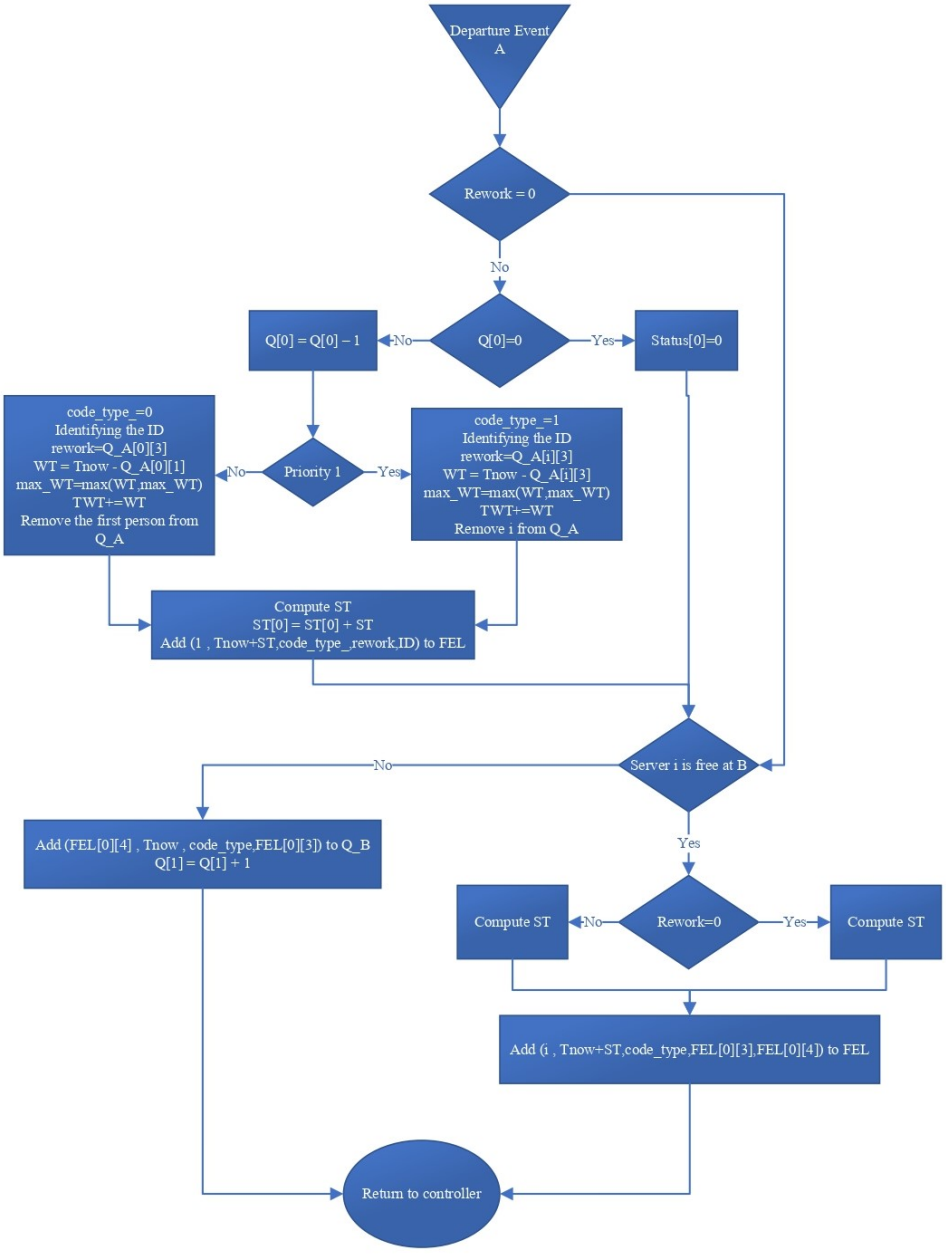

Figure 17: Departure from A Event

Departure From B

First, we check the `code_type`; `code_type` = -1 means it's breakdown time for station C's server (we modeled the breakdown event as a machine whose `code_type`=-1). If `code_type`= -1, check the status of station C's server; if busy, we put the breakdown event at the top of `Q_C` and if it was idle, we change the status to 1.

If `code_type` $\neq$ -1, we first check `Q_B` and assign the next machine to the server, after

that, we create a random digit from which we decide to which station we should send the machine.

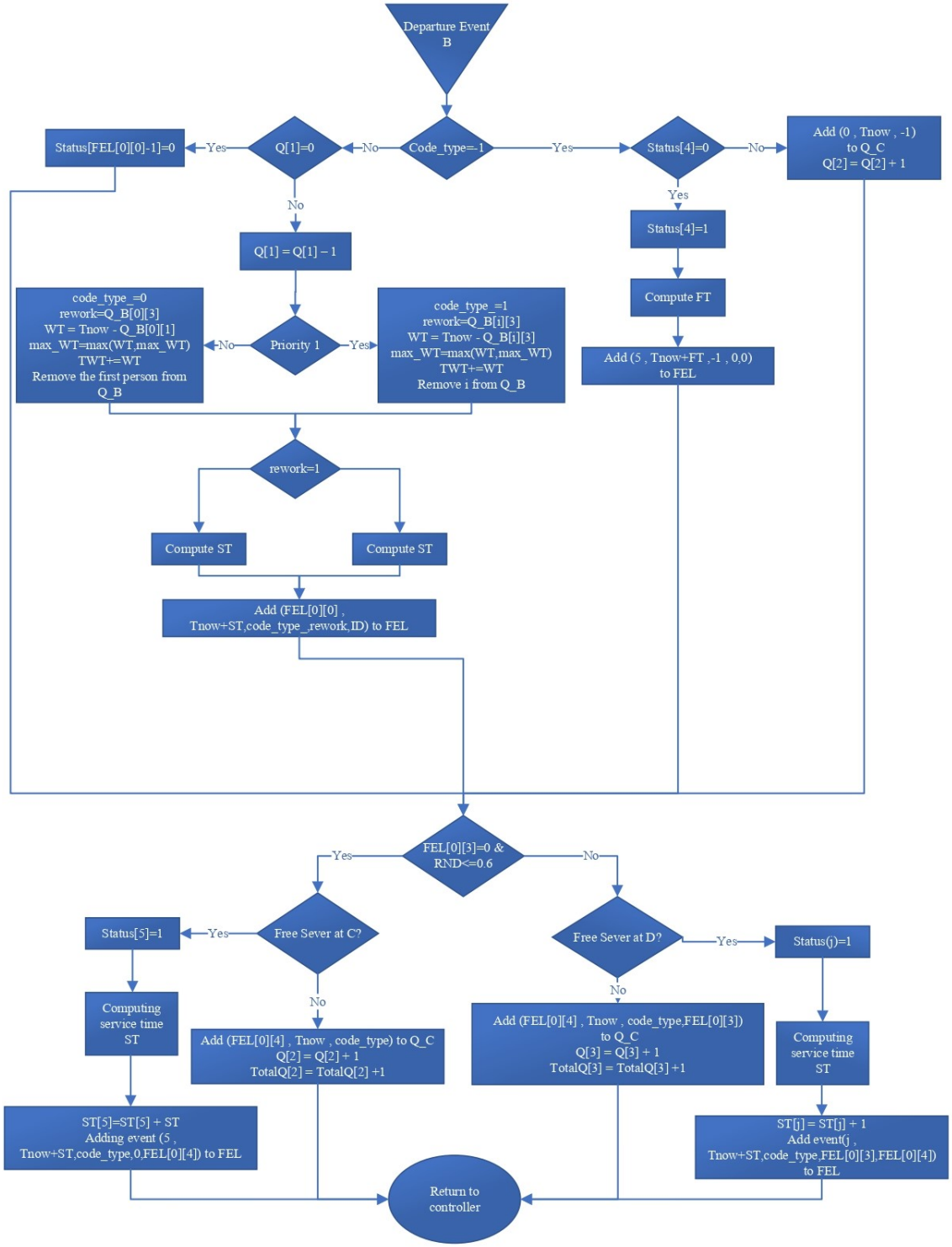

Figure 18: Departure from B Event

Departure From C

First check `Q_C` , if empty, we set status [4] =0, otherwise, we should decide whether the item at the top of the list is a breakdown event or a machine that was sent to C. If it was the breakdown event, we compute the fixing time and append it to the FEL otherwise we check the status of station D's servers and so on.

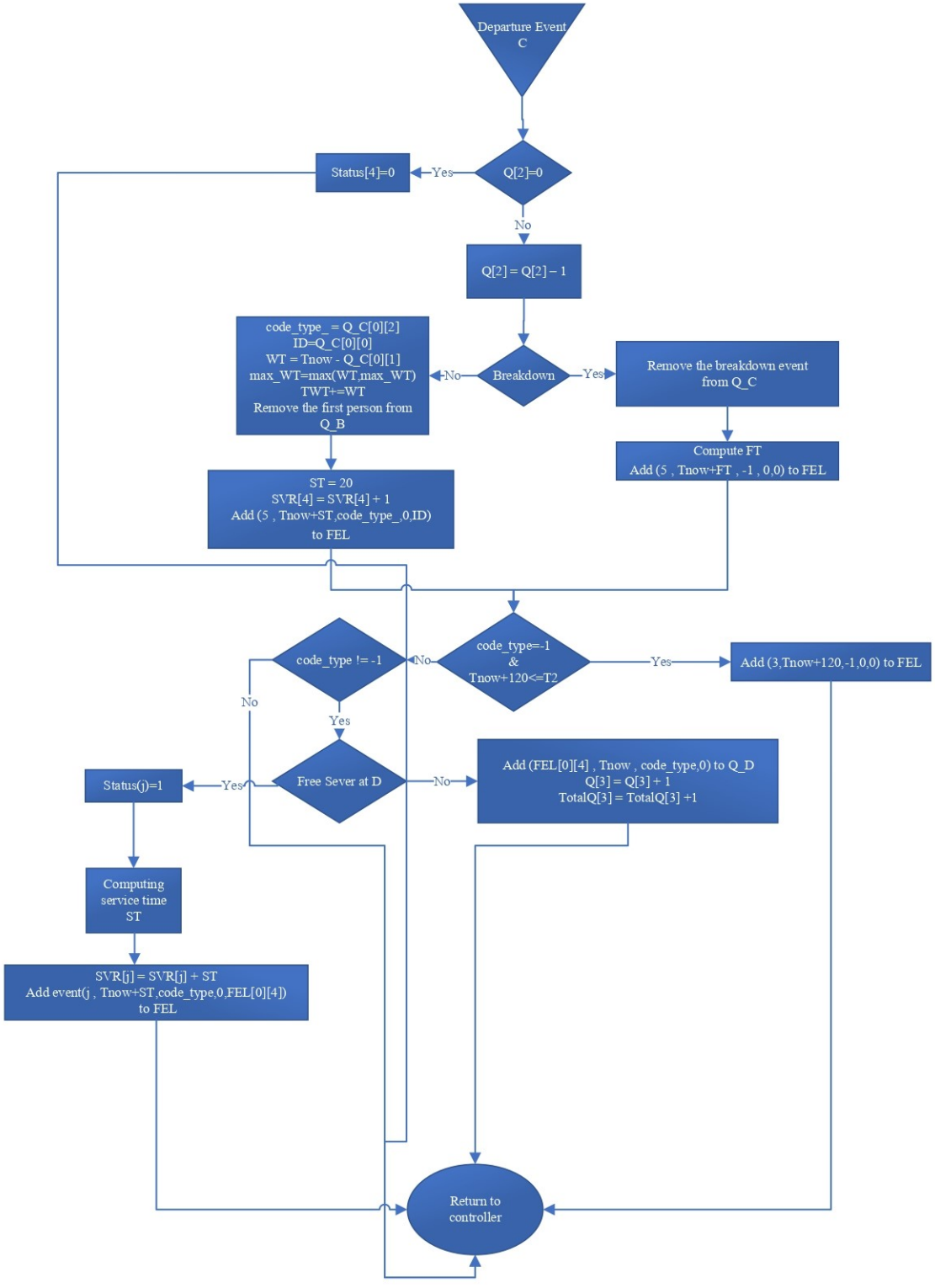

Figure 19: Departure from C Event

Departure From D

First, we check the queue and assign the next machine to the sever after that we create a random digit to decide whether to send the machine for a rework (to station B) or to send it to station E.

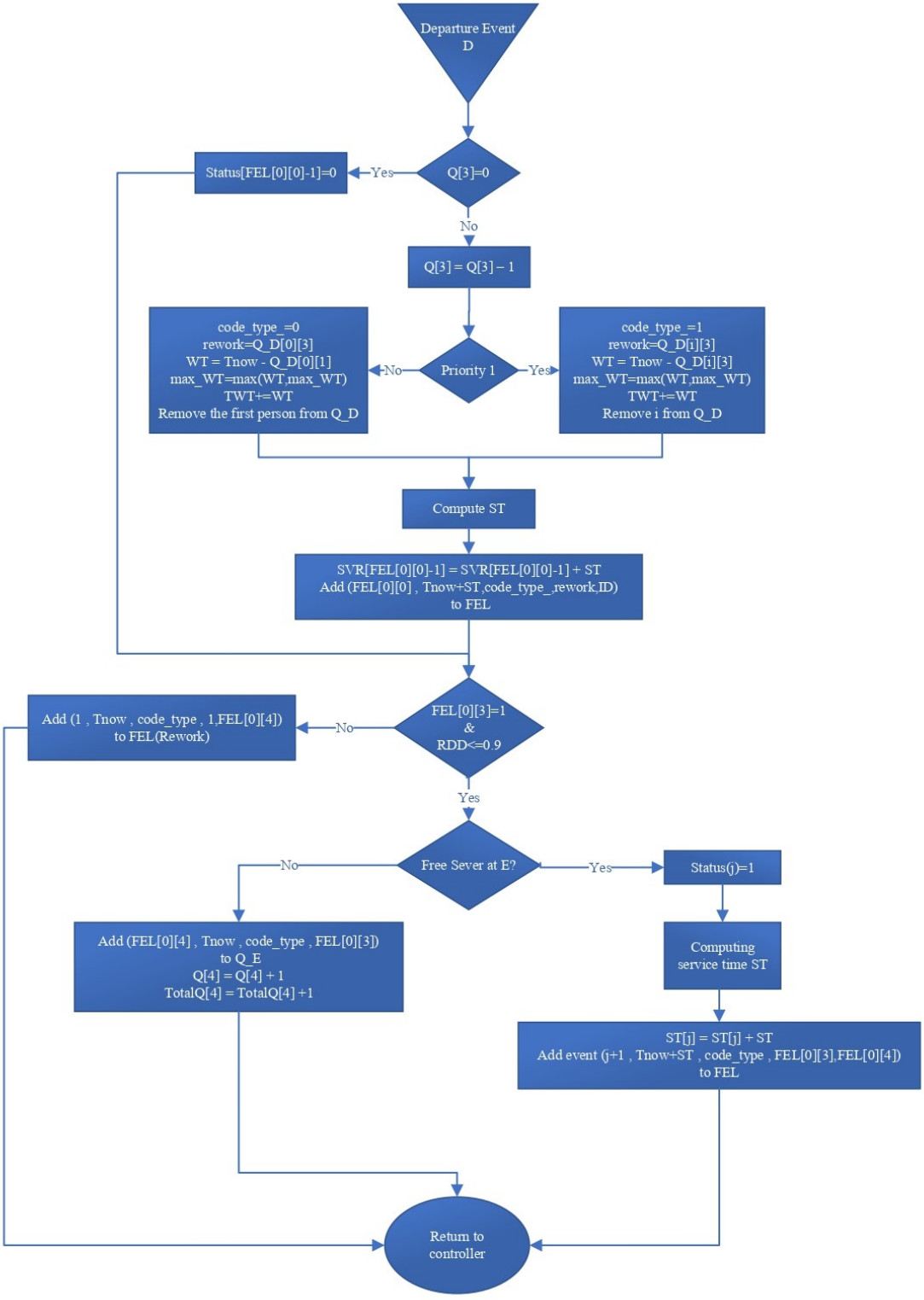

Figure 20: Departure from D Event

Departure From E

We just have to assign the next machine to the server.

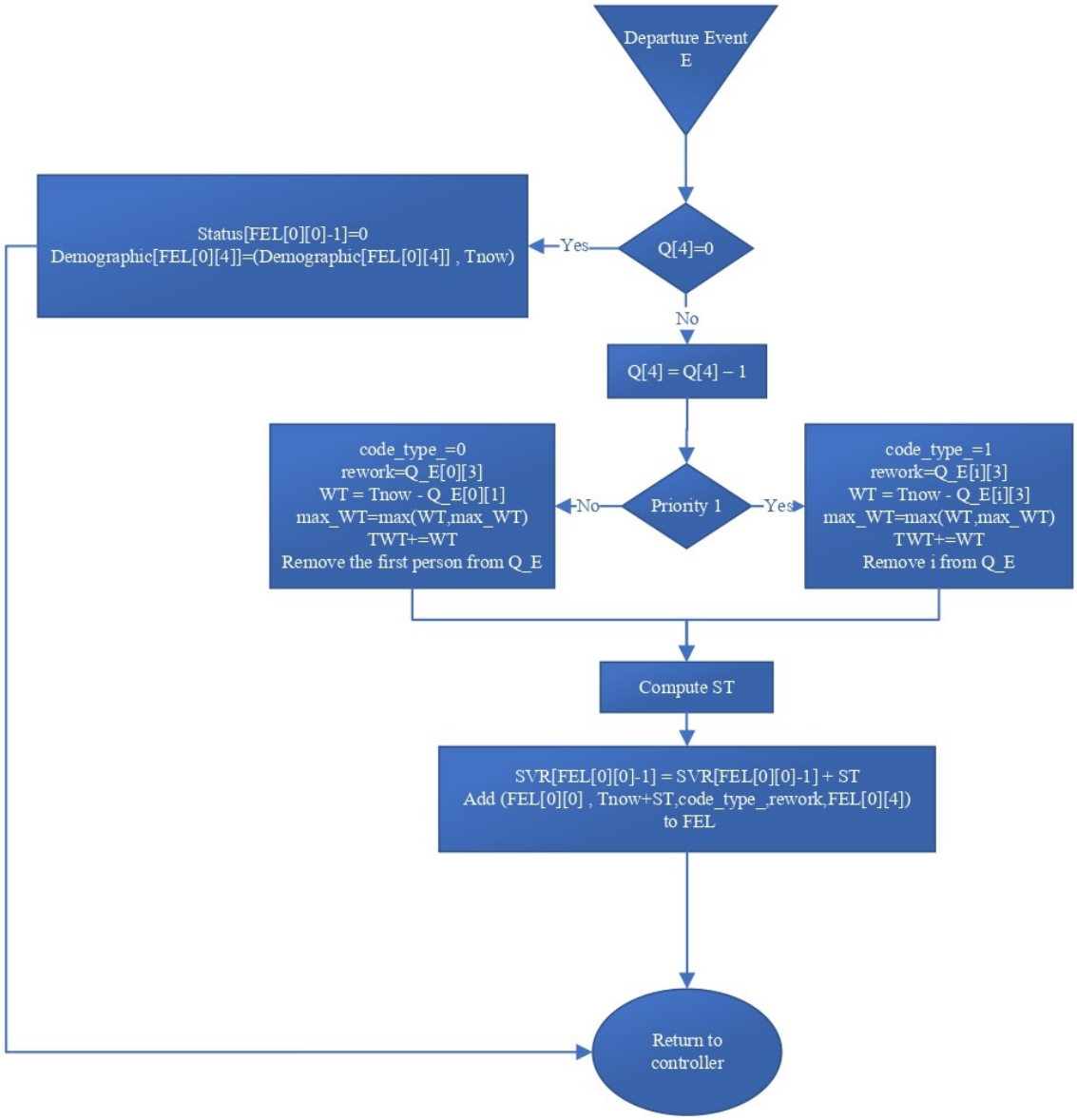

Figure 21: Departure from E Event

Simplifying Assumptions

- The first standard and urgent orders enter the system at $T = 0$.
- Tasks are assigned to servers by the numerical order of the servers in a station.
- All machines that enter the system will be served.
- The first breakdown of station C occurs at $T = 180$ and repeats every two hours without considering the service time of the station.
- If the server at station C is busy at the breakdown time, we wait until it becomes idle and then proceed to break it down.
- After $T = 600$, the system does not accept any new entrances.
- Statistics are computed for the time interval between $T = 120$ and $T = 600$. For example, if a new arrival occurs at $T = 590$ and the machine enters a queue, its waiting time is at most 10 minutes, even if it is still being served at $T = 610$.