MACHINE LEARNING BASED ALLOCATION IN A LOT SIZING GAME

A Thesis

by

Ömer Berkay Sarıoğlu

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master's Degree

in the
Department of Data Science

Özyeğin University
December 2022

MACHINE LEARNING BASED ALLOCATION IN A LOT SIZING GAME

Approved by:

Prof. O. Örsan Özener, Advisor
Dept. of Industrial Eng.
Ozyegin University

Assistant Prof. Başak Altan
Dept. of Economics
Ozyegin University

Associate Prof. Uğur Çelikyurt
College of Administrative Sciences and Economics
Koc University

Date Approved: 13 December 2022

To my lovely family

ABSTRACT

Supply chains often have confliction objectives and operate in a finite resource setting. Traditionally, companies focused on their internal processes to generate cost reduction opportunities in order to increase their profitability. However, recent studies suggest that collaboration is the key to have sustained benefits among supply chain partners. Supply chains, which usually have conflicting objectives, can form coalitions and take advantage of collective payoffs. In this paper, we analyze a collaborative production setting where several companies facing varying demands throughout a finite planning horizon attempt to reduce their procurement costs by ordering from a common supplier. The participants exploit the synergy among themselves by maximizing the capacity utilization of the supplier. We design a novel cost allocation method using various machine learning techniques with the goal of generating a cost-allocation mechanism that ensures the sustainability of the collaboration. We conduct a computational study to compare and contrast our proposed method with the generic methods in the literature. We discuss the advantages and disadvantages of these methods in terms of solution quality and computation time.

ÖZETÇE

Tedarik zincirleri genellikle çelişkili hedeflere sahiptir ve sınırlı bir kaynak ortamında çalışır. Geleneksel olarak şirketler, kârlılıklarını artırmak amacıyla maliyet azaltma fırsatları yaratmak için kendi iç süreçlerine odaklandılar. Bununla birlikte; son çalışmalar, tedarik zincirinin paydaşları arasında sürekli fayda sağlamanın anahtarının işbirliği olduğunu göstermektedir. Genellikle birbiriyle çelişen hedeflere sahip olan tedarik zincirleri, koalisyonlar oluşturabilir ve toplu kazançlardan yararlanabilir. Bu çalışmada, sınırlı bir planlama ufku boyunca değişen taleplerle karşı karşıya kalan birkaç şirketin ortak bir tedarikçiden sipariş vererek tedarik maliyetlerini düşürmeye çalıştığı işbirlikçi bir üretim ortamını analiz ediyoruz. Katılımcılar, tedarikçinin kapasite kullanımını maksimize ederek kendi aralarındaki sinerjiden yararlanırlar. İşbirliğinin sürdürülebilirliğini sağlayan bir maliyet dağıtım mekanizması oluşturmak amacıyla çeşitli makine öğrenimi tekniklerini kullanarak yeni bir maliyet dağıtım yöntemi tasarlıyoruz. Önerdiğimiz maliyet dağıtım yöntemini literatürdeki genel yöntemlerle karşılaştırıyor ve çözüm kalitesi ve hesaplama süresi açısından bu yöntemlerin avantaj ve dezavantajlarını tartışıyoruz.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisor, Prof. Örsan Özener. You always had time to answer my questions not only while we were working on this paper, but also during your lectures. Your openness to discuss everything with a challenging mindset was truly inspiring. I feel extremely lucky to be able to learn from you both academically and personally. In addition to your academic and personal contributions, I am extremely grateful for our friendly chats at the end of our meetings. I hope to watch an F1 race with you.

Also, thank you mom and dad for teaching me at a very young age the importance of learning. Thank you for supporting me throughout my entire life. I always felt your love, your support, and comfort of being able to talk to you on everything.

Last but not least, I would also like to thank my wife Deniz. Deniz, thank you for your incredible support throughout this journey. When I needed to work long hours in between my work and my Master's degree, you were always there for me even when I didn't need anything. Thank you for the motivation talks and and your endless support. Thanks to you, I never felt alone. I am sorry for the holidays we needed to skip. I promise we will make up for them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In this paper, we aim to distribute lot sizing costs to multiple players in a lot sizing game using optimization and machine learning algorithm. Our objective is to allocate costs in a way that reduces the instability of the game and that ensures the long term sustainability of the coalition. In this chapter, we explain the lot sizing game and the previous work in the literature. In addition to that, we briefly explain the Bayesian optimization and random forest modelling approach that is used in this study.

1.1 Context

In cooperative game theory literature, a game refers to a situation where a group of decision makers join their effort to achieve certain objectives and finally acquires certain benefits resulted from those objectives. In most cases, such objectives are very difficult or impossible to achieve by individual efforts. In that aspect, cooperative efforts, or more commonly used term collaborations, supposedly benefits all parties involved in the process. The problem, however, is ensuring the sustainability of the collaboration to realize these benefits as in most cases these individual decision makers seek to maximize their of walfare rather than the welfare of the group.

Under these selfish agenda of the individuals, the goal of the collaboration is to design collaboration governing mechanism to sustain the collaboration. This mainly entails developing a cost/benefit sharing mechanism among participants. as in most cases the conflicts are resulted from and hence can be solved by reallocation of benefits (transferable assets such as money). Unfortunately identifying a good/fair allocation mechanism might be extremely challenging as these conflicts among participants presents constraints that are quite difficult to satisfy.

In our collaborative structure, the companies forms a joint procurement structure in order to reduce the procurement cost by eliminating the inefficiencies in production. The common supplier is assumed to have a capacity and unstationary non-linear cost structure both of which complicates the problem. By complication, we not only mean the difficulty in identifying the optimal solution but also the difficulty in determining a fair allocation method. The problem is proved to be an NP-Hard problem, however the optimal solution is relatively easy to identify using commercial solvers as long as the number of collaborators are reasonable.

The goal in the economic lot-sizing game as in other cooperative games studied in literature is to identify a fair allocation. In most cases, this is equivalent to finding the *core* of the game. The core refers to a budget balanced and stable allocation. As we are allocating the joint procurement costs, the allocation should cover all the joint costs (budget balanced condition) and the allocated costs cannot be greater than alternative allocations. If a core solution cannot be found, close approximations to the core could be sought. Given that the capacitated economic lot-sizing game might have an empty core, as proved in the literature, our objective is to identify an allocation that has the minimum instability values.

In the lot sizing problem, as the number of periods increases, the relation between the total cost allocations and respective percentage of stability decreases. The percentage of stability is calculated by calculating the percentage of different cases where with the given costs, the percentage cases that would continue collaborating. The methodology is further detailed in Chapter 2. Increase in the number of period introduces increase in number of inputs to the optimization and modelling approaches. This results in curse of dimensionality and makes the problem hard to solve with the optimization and modelling approaches [1].

Table 1: Impact of Total Cost Allocation Change over Target

Index	Cost Alloc P1	Cost Alloc P2	Cost Alloc P3	Cost Alloc P4	Target
1	13.35858	15.00829	12.4912	14.41755	0.06556
2	13.14545	15.48449	12.22472	14.42179	0.088661
3	15.15042	15.3972	12.12697	12.1469	0.161747

In addition to the curse of dimensionality, due to the nature of the problem, small changes in the total cost allocation in a given period, the target value changes significantly. Table 1 shows the impact of the change in total cost allocations over target variable. The columns Cost Alloc P1, Cost Alloc P2, Cost Alloc P3, and Cost Alloc P4 columns indicate the total lot sizing cost in the periods 1,2,3, and 4 respectively. Although the difference between the total cost allocations of the first and the second cost allocation scenarios are minimal, the target of the second cost allocation scenario is 35% more than the first cost allocation scenario. In addition to that, the target value is sensitive to the small changes in the input values as it can be observed when the third cost allocation scenario is compared to the rest. For these reasons, the lot sizing problem is also a difficult problem to solve using optimization and machine learning algorithms.

1.2 Literature Review

The lot-sizing problem was introduced by [2], which study the problem without capacity limitation and proposed a polynomial solution approach. There are survey papers summarizing the related work on lot sizing problems such as [3] and [4].

The capacitated dynamic lot-sizing problem is proven to be NP-Hard [5], [6]. For NP-Hard variants of the problem (i) pseudo-polynomial solution methods [7], [8], [9], and [10], (ii) branch and bound algorithms [11], [7] , [12], and (iii) branch and cut

algorithms [13] have been proposed.

From the aspect of collaborative games, there is a vast literature. The *core* concept was introduced by [14]. As the core might be empty, the least core proposed by Maschler et al.(1979)[15] or gamma-approximate core proposed by April et al. (2007) can be considered for cooperative games. The closest studies to our works is the literature on lot-sizing games. [16] study the lot-sizing game without any capacity restrictions. [17] propose an approximate budget balanced allocation method for the game with backordering.[18] prove that the core is non-empty and an allocation can be computed by solving a modified dual problem. Finally, [19] examine the cooperative lot-sizing problem under many constraints including the capacity constraint.

Contrary to the studies above, we assume a capacity restriction in production in each period. We also propose a machine learning based solution methodology for allocating the costs and to the best of our knowledge this is the first study that uses ML methodology in cooperative games.

1.3 Bayesian Optimization

Bayesian optimization is an optimization technique based on Bayesian statistics which mainly utilises the procedure of updating a prior belief to produce a posterior belief in a probabilistic manner when new information is encountered. Like many other optimisation techniques, Bayesian optimization aims the find the global minima of an objective function in smallest number of steps.

In Bayesian optimization, a "surrogate function" that approximates the objective function is initialised and iteratively updated. This function is formed by sampling points on the objective function and is updated by determining the regions that may contain the global minima. By updating the surrogate function according to the promising areas of global minima, the Bayesian optimisation aims to eventually reach the global minima of the objective function.

In Bayesian optimization, these surrogate functions are represented as probability distributions by the "Gaussian Process" and they can be updated with Bayesian processes as new information is gathered. Gaussian Process assigns probabilities of representing the current data points to surrogate functions and the probability of representation is updated with each iteration.

The surrogate functions are updated with the utilisation of an "acquisition function". The acquisition function selects new data points in each iteration whilst balancing an exploration-exploitation trade-off. It exploits areas where the surrogate function predicts a promising objective and also explores areas where the uncertainty is high. This way, Bayesian optimisation aims to avoid being stuck in a local minima and find the minima of the objective function.

The evolution of the modern day Bayesian optimization took almost a decade, with many researchers contributing to the development of the technique. The exploration-exploitation trade-off was first studied by William Thomas in 1930s. Later in 1960s Kushner studied the global and local optimization trade-off and utilised an acquisition function with the intention of maximising the probability of improvement. In 1978, O'Hagan mentioned the use of Gaussian Processes as prior over functions. Around the same time, Močkus and colleagues coined the term "Bayesian Optimisation" and developed a multidimensional Bayesian optimization approach. See [20], [21], [22], [23], [24], [25], [26], and [27] for more details on Bayesian optimization.

1.4 Random Forest Model

Random forest models (RF) are popular machine learning models that are widely used for classification and regression problems. RF models consists of an ensemble of another machine learning model called the decision trees. Decision trees are predictive models that break down a data set into smaller subsets in a hierarchical manner. They learn simple decision rules (i.e. splitting criteria) that divide the data into subsections

(also called leaf nodes) with the aim of predicting the value of a target variable. To divide the data into leaf nodes, they ask sequence of questions about the features. The questions which define the splitting criteria are selected based on the "pureness" of the subsections. The splitting value that result in "purest" or the most homogeneous leaf is selected and this process is repeated until a stopping criterion is reached. An example of a stopping criterion could be nodes containing less than 5 data points.

RF models consists of an ensemble of another machine learning model called the decision trees. Decision trees are predictive models that break down a data set into smaller subsets in a hierarchical manner. They learn simple decision rules (i.e. splitting criteria) that divide the data into subsections (also called leaf nodes) with the aim of predicting the value of a target variable. To divide the data into leaf nodes, they ask sequence of questions about the features. The questions which define the splitting criteria are selected based on the "pureness" of the subsections. The splitting value that result in "purest" or the most homogeneous leaf is selected and this process is repeated until a stopping criterion is reached. An example of a stopping criterion could be nodes containing less than 5 data points.

Decision tree models are prone to overfitting the data set. RF models on the other hand utilise multiple decision trees that are built by using randomly selected subsamples of the data, which makes them less prone to overfitting and better generalize to the unseen data. RF models introduce this randomness by randomly sampling data points and by considering a random set of features when splitting the nodes. They then average the predictions of each decision tree (i.e. a process called bagging). This way RF models ensure that although each decision tree may have high variance, the overall ensemble model has low variance and an acceptable bias. In addition, because each decision tree can be built with parallel processes they offer an efficient training process.

The idea to use a randomly selected sample of the data to build each decision tree

was introduced by Breiman in 1996 [28] and the technique was named "Bootstrap Aggregation" or in short "Bagging. Selecting a random set of features to build each tree to achieve maximum accuracy while avoiding overfitting was mentioned separately by Amit and Geman in 1997 [29] and Ho in 1998 [30]. Later in 2001 [31], Breiman combined the bagging technique with random feature selection technique and introduced the Random Forest model scheme. For more information, we refer the reader to [32].

In this paper, the methodology of our Bayesian optimization and random forest models are explained in Chapter 2. The results for each method are presented in Chapter 3.

CHAPTER II

DETAILED PROBLEM DESCRIPTION

In this chapter, we share the objective and the design of the study along with the methodologies followed in each design approach. For each method, the design constructs and the steps followed will be shared in detail.

2.1 General Approach and Solution Design

In this study, our objective is to find a novel cost allocation method using optimization and machine learning techniques to better allocate costs to different players in a cooperative lot sizing game that will ensure maximum stability for the cooperation. Bayesian optimization and machine learning models are compared with the existing methods of cost allocation over different time horizons. Due to time and computational complexity limitations, only random forest models are developed for comparison. We compare the cost allocations obtained with our proposed methods with the pattern based cost allocation method proposed in the reference [33]. The main objective of the pattern based cost allocation is to focus on forming coalitions for the conditions where the capacity of the suppliers are fully utilized. Since those situations represent the cases that maximizes efficiency and that are less tolerant to changes, creating stable coalitions for those cases ensures the stability in all cases [33].

Base pattern allocation (BPA) is a patern based algorithm which tries to assess the synergy among participants in an exclusive manner. For more detailed information on different existing cost allocation methods, we refer the reader to [33].

In order to generate multiple collaboration scenarios, we define the required problem setup parameters using different seeds each time. The required parameters for

the cooperative game theory setup are: back order cost, capacity, demand, fixed cost, holding cost, and variable cost. Back order cost represents the cost of supplying a demand that originates from the past. Capacity indicates the total production capacity in a given period of time. Demand indicates the total customer demand in a given period. Fixed cost indicates the total fixed costs regardless of the number of products demanded, while the variable cost indicates the cost per product requested. If the product is requested ahead of its required time, than a holding cost is applied for the number of periods it is stored. Each of these problem setting variables are generated using a random integer generator with varying lower and upper bounds. Additionally, since these values are dependant on the number of periods the game is expected to be played, the parameters are vectors with the shape of $[1, NumberofPeriods]$.

Back order cost variable is generated within the interval of $[2.4, 4.2)$ using a uniform distribution. Capacity variable is generated within the interval of $[11, 21)$ using a discrete uniform distribution. Demand variable is generated within the interval of $[6, 11)$ using a discrete uniform distribution. Fixed cost variable is generated within the interval of $[50, 76)$ using a discrete uniform distribution. Holding cost variable is generated within the interval of $[0.8, 1.4)$ using a uniform distribution. Variable cost variable is generated within the interval of $[6, 11)$ using a discrete uniform distribution.

During our study, we have compared the maximum percentage deviation from stability of each allocation within 1000 different customer demands. To generate the 1000 different customer demands, we first use a random number generator with the length of number of periods of the collaboration process to generate a base demand. This process returns a demand vector that contains customer demand on each period. Then, using another random number generator and a cumulative distribution function, we multiply the base demand with the multipliers obtained by the second random number generator to add noise and to create variability between demand values for different customers. By repeating the same process 1000 times, we obtain

a pool of 1000 customers with their synthetically generated demands.

Once we generate the synthetic customers with their demand vectors, we then calculate the optimum lot sizing cost for each customer demand scenario. To calculate that, we use exacts methods available in CPLEX implementation of Python. The goal of the objective function is to minimize the cost of collaboration that satisfies the given demands for each player. To restrain the function from assigning zero cost to each player and to satisfy the problem needs, a total of 5 constraints are introduced. First constraint enforces the demands to only be originated from either current period or the previous periods. Second constraint ensures the total production in a given period is strictly less than the capacity of that period. Third constraint tightens the LP relaxation. The fourth constraint is the binary constraint to represent if the production is required in the given period. The last constraint ensures non-negative cost. For more detailed information, we refer the reader to [33].

After calculating the optimum lot sizing cost for each demand, we are able to calculate for each allocation, the maximum percentage deviation from stability. This value indicates the percentage of the cases where the allocation is unstable given a cost allocation vector. By testing each cost allocation in different demand scenarios, we are able to better identify if a cost allocation will be stable.

To calculate the maximum percentage deviation from stability, for each cost allocation to be evaluated, we first normalize the cost allocations so that the sum product of the cost allocations vector with the demand vector is equal to the optimum lot sizing cost value. This normalization process is done by calculating the sum product of the cost allocations with the demand vector which is followed by finding the ratio between the optimum lot sizing cost value and the total cost obtained by the sum product operation. Later, each cost allocation is divided by this ratio to ensure that the sum of the products are equal to the lot sizing cost. This step is done to ensure that the cost allocations are calculated for the scenario where the maximum

production capacity is utilized. Since those cases are the ones with most probability of deviating from stability, solving the problem for that ensures solving for the rest of the cases as it was described in [33].

After the normalization step, the first step in calculating the maximum percentage from stability is to calculate the percentage from stability. To calculate the percentage from stability, we first calculate the cost of each customer demand and allocation pairs. Once the cost of each demand and allocation pair is calculated, we find the percentage difference of the cost of the pair compared to the optimum value of the pair. If the optimum is smaller than the cost of the customer demand and cost allocation pair, then the percentage difference of the cost of the pair is assigned as the percentage from stability. Otherwise the percentage difference of the cost of the pair is assigned as zero, meaning that the pair is stable. Once we calculate the percentage difference of the cost for each customer and allocation pair, we take the maximum of the percentage difference of the cost for each allocation to calculate the maximum percentage from stability.

In our study, we aim to find the cost allocation setting for a given scenario that have the lowest maximum percentage from stability value than the one generated with the base pattern allocation method. By finding the cost allocation that minimizes the maximum percentage from stability, we aim to find the allocations that ensure stability for a given scenario. Since existing methods does not guarantee global optimum, we aim to find a better cost allocation method that is more effective and more accurate than the base pattern allocation method. To find that, we try Bayesian optimization method along with a machine learning based approach. Due to the ability of the random forest algorithm's non-sequential approach and the simplicity in training and tuning processes, only random forest models are used [34].

The following sections explains in detail the used methods and the solution designs in relative methods.

2.1.1 Bayesian Optimization

In this subsection, we share the solution design using the Bayesian optimization process in more detail along with the detailed settings used in the Bayesian optimization algorithm. In addition to that, we share the data generation steps used in the solution design using Bayesian optimization.

With the Bayesian optimization process, our objective is to find the cost allocations that minimizes the maximum percentage deviation from stability. To do that, we use the Python implementation of the Gaussian process on continuous target setting in the scikit-learn library, called "GaussianProcessRegressor". The GaussianProcessRegressor method aims to find the independent x_n variables that maximizes the dependent variable y. Since our objective is to minimize the maximum percentage deviation from stability(y), we configure our target variable within the optimization process to $-y$ so that the Gaussian process finds independent variables that minimizes the maximum percentage deviation from stability.

The Bayesian optimization process takes an initial distribution and using that distribution, predicts the next best point in space that reduces the variance in the distribution and aims to predict the actual function better over time. This process requires us to first generate an initial sample for the Bayesian optimization process to start on. Then, for each iteration, generate a number of samples for the algorithm to search and find the next possible allocations that is estimated to decrease the variance in the surrogate function. This means that our solution approach using the Bayesian optimization process has two main steps: initialization and optimization.

Initialization step consists of 3 phases; data generation, normalization, and model fit. In the data generation phase, for each cost allocation in a given period, we define the interval in which the cost allocations are generated. In addition to that, we define the number of initial observations to be generated for the Gaussian process' initialization. In our application, we defined the cost allocations in each time period

to be between $[0, 20)$ interval and to originate from a uniform distribution. For example, in a setting where the objective is to initialize cost allocations within a 3 period horizon, the initial cost allocation values can be $[12.45, 16.85, 10.95]$ with each value in the list represents the total cost allocation in a different time period. This process is repeated for the number of initial observations requested. In our study, we tried different number of initial observations to see the effect of initial observations on the rate of convergence.

Secondly in the initialization step, once the requested number of allocations are generated, the allocations are normalized so that the sum of the cost allocation times demand values are equal to the optimum lot sizing cost of the game. The details on the normalization process are explained in the General Approach and Solution Design section.

Finally for the initialization step, using the normalized cost allocations and the 1000 different demand settings explained in the General Approach and Solution Design section, hereinafter referred to as the actual model, the actual maximum percentage from stability for each allocation, hereinafter referred to as y_true, is calculated. Using the normalized cost allocations as the independent variables of the Gaussian process and negative of the the actual maximum percentage from stability score as the dependent variable of the Gaussian process, the optimization model is initialized and trained with the initial number of samples. With the training of the optimization model with the initial samples, the initialization step is finalized.

Next step in the Bayesian optimization process is the optimization step. In the optimization step, the initialized Gaussian process is used as the surrogate function of the Bayesian optimization process. A number of different instances are created in each optimization loop and using the surrogate function, the expected maximum percentage from stability for each allocation values, hereinafter referred to as y_pred, are calculated. After that, using an acquisition function, the next best independent

value set, i.e. next best cost allocation, that will reduce the variance in the surrogate function is generated. Once the next best cost allocation is identified, that allocation setting is evaluated through the actual model to get the y_true value in addition to the y_pred and the surrogate function of Gaussian processor refit using also the next best cost allocation and its y_true.

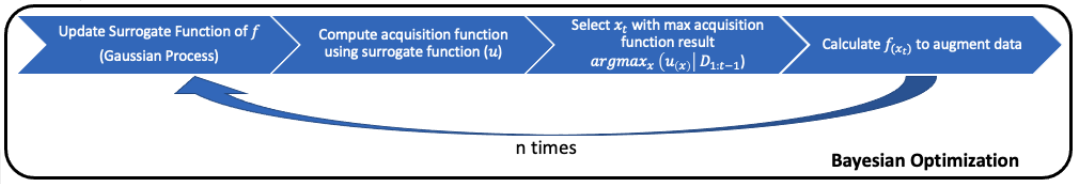

Figure 1: Bayesian Optimization Process

Figure 1 shows the Bayesian optimization process of a function f. First a surrogate function is defined, which is the Gaussian process in our study. Then, the acquisition function is calculated to find the next best independent variables that maximize the acquisition function. The function f is calculated for the best independent variables and used to augment the surrogate functions representation.

To iterate within the optimization step, we define 5 parameters; number of optimization loops, number of instances to be generated in each loop, cost allocation range in each instance, optimization function, and optimization function's multiplier. In the following paragraphs, we explain in more detail the parameters and the optimization process.

The number of optimization loops defines how many times the optimization process is going to run. A single optimization process consists of data generation, scoring through surrogate function, selecting next best allocation using an acquisition function, running the actual model with the next best allocation, and retraining the surrogate function with the next best allocation and its y_true. In our research, we try different number of optimization loops within different intervals to better understand the number of optimization loops.

The number of instance in each loop determines the number of instances generated in each loop from which the next best allocation is selected. This means that as the number of instance in each loop parameter increases, there will be more cost allocation instances to be scored by the surrogate function and more instance to be select the next best allocation from. In our study, we used 1000 instances in each loop as standard.

Cost allocation range in each instance determines the interval from which the cost allocation instances will be generated. This parameter defines for each loop and for each instance within the loop, the distribution from which the cost allocations are generated. In our study, we used different intervals to better understand the impact of the exploration and the exploitation. We first generate allocations using a uniform distribution from $[0, 20)$ interval. Since we also use the same $[0, 20)$ interval in the initialization phase, we call this approach as the "Entire Feature Space" approach. In addition to that, we have tried to optimize the Bayesian optimization around the best allocation obtained up to a given point. We achieve this optimization process by taking the allocation with the minimum y_true until the given point and reduce the cost allocation range from $[0, 20)$ to $[(Best\_allocation - n), (Best\_allocation + n))$ while still using uniform distribution. We call this approach as the feature space reduced approach and within our Bayesian optimization approach trials, we set n value to 5,3,and 1 to better understand if the conversion of the algorithm is impacted by the search space.

3 Period Setup	Feature Search Space
Allocation 1	min — max
Allocation 2	min — max
Allocation 3	min — max

Figure 2: Data Generation Process in a 3 Period Setup Using Entire Feature Space

Figure 2 visualizes the feature search space for the entire feature space approach in a 3 period game theory setup. For a 3 period game, 3 cost allocation parameter needs to be generated. As the figure illustrates, each cost allocation parameter is generated using the entire future space, which is $[0, 20)$.

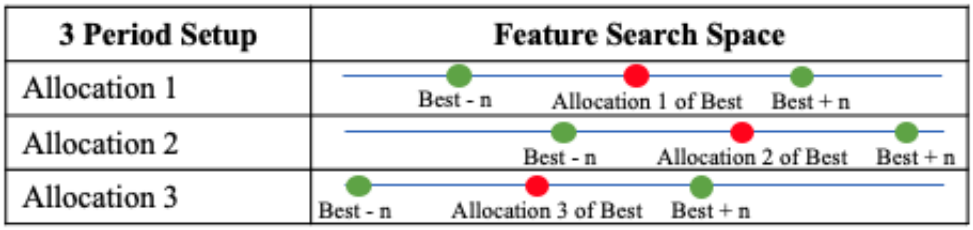

Figure 3: Data Generation Process in a 3 Period Setup Using Reduced Feature Space by n

Figure 3 visualizes the feature search space for the reduced feature space in a 3 period game theory setup. For a 3 period game, each cost allocation parameter is generated using the $[-n, +n)$ neighborhood of the best allocation value for the respective cost allocation. As the figure illustrates, best allocation value for each period can be different from each other.

An important parameter in the optimization phase is the definition of the acquisition function. In our study, we use the upper confidence bound acquisition function (UCB). The upper confidence bound acquisition function allows the user to take into consideration the variance in the predictions in addition to the prediction that maximizes the expected output. This is done with the help of explicit β parameter in the function. Equation 1 below shows the structure of the function. As the β parameter gets a high value, the function priorities the values where the error is higher, i.e. exploration, and when it gets a lower value, the function priorities around the current best observation, i.e. exploitation [35]. We chose to utilize upper confidence bound as our acquisition function because of its ability to allow explicit modification between the exploration and exploitation trade-off by trying different β parameters. We use

different beta values in different problem settings to be able to observe the impact of the acquisition function with regards to the problem setup in our study.

$$x_t = \underset{x \in D}{argmax} \mu_{t-1}(x) + \beta_t^{1/2} \sigma_{t-1}(x) \tag{1}$$

Once we define the optimization process' settings, the optimization process begins. In each optimization loop, the optimization process starts with generating the specified number of instances in a loop using specified cost allocation parameter range. In our study, we use multiple different number of instances in a loop parameter along with multiple cost allocation parameter ranges. Once the instances are generated, we then score the instances using our surrogate function, Gaussian process. After scoring the instances, we then score each instance with our acquisition function to identify which cost allocation should be evaluated through the actual model to complement the Gaussian process' predictions the most. After deciding the next best instance, we then calculate the y_true value of that allocation using our actual model. Then, by adding the next best instance to our previous data with independent variables and their respective y_true values, we refit the Gaussian process regressor object so that it better learns the actual distribution of y_true. After that, the optimization process repeats as many times as the number of optimization loops defined. For each optimization loop, different number of instances, different range for cost allocation parameters, different optimization functions, and different β parameters can be used. In our study, we keep number of instances to 1000, optimization function to upper confidence bound same with varying β parameters and varying range for cost allocation parameters.

The Bayesian optimization process described above can be used with different data sets and on cooperative game theory setups with varying time horizons. In order to observe the impact of the data set and the time horizon on the performance of the Bayesian optimization, we run the same process with different data sets generated

with the same approach. In addition to that, we try our proposed methodology in different time horizons. In our study, we try to find the best cost allocations in a 3 period game, 4 period game, and in a 10 period game. For each period setup, we use multiple data sets to better observe if the process changes as the data changes.

After finalizing the Bayesian optimization process, we evaluated the optimization performance by evaluating the prediction error between the Gaussian process(y_pred) and the actual model results(y_true). We expect that the difference between y_true and y_pred to decrease as the number of optimization loops increase. In addition to that, we expect the standard deviation to decrease as well since the Gaussian process should learn as the iterations increase. We compare the different optimization processes within their respective period setting. In other words, the optimization process results for a 3 period game is only compared between the results of a 3 period game.

With this setup, we aim to observe the Bayesian optimization's conversion performance in different game period settings and using different optimization settings.

2.1.2 Random Forest

Our objective with the random forest approach is to develop a model that learns the relation between the cost allocations and the maximum percentage deviation from stability, which is our target. With that model, we can evaluate the feature space with minimal time cost and find cost allocations that result in smaller y_true values than the y_true obtained using the base pattern allocation approach. As the Random Forest algorithm is not an optimization algorithm but a machine learning modelling technique, we define a methodology to concentrate on the cost allocations with the minimum y_true. To achieve that, we first generate a data set for the model to learn from. Secondly, we develop a random forest model using different hyper parameters. Finally, we evaluate the model results on a data set with more instances to be able

to find in the search space, the allocations that result in minimum y_true value. As the model prediction process takes an insignificant amount of time, we were able to evaluate a wider parameter space once we have a model. In this section, we explain each step in more detail along with the game periods and seeds used to evaluate our results.

First step in using the Random Forest algorithm is generating data to develop the model. In this study, we aim to develop a stable and predictive model with as less data as possible. The reason behind this approach is that data generation process also requires time and effort. By minimizing the data required for the model development process, we aim to make the random forest modelling approach more attractive. The data generation process for the random forest algorithm and the initialization step of the Bayesian optimization is nearly identical. For both the random forest data generation process and Bayesian optimization's initialization step, data needs to be generated and normalized. Only difference between the approaches is that in Bayesian optimization, a Gaussian process needs to be defined, whereas for the random forest model approach it is not necessary. So, the same method used in the Bayesian optimization's initialization step has been used to generate an initial data set that consists of 1000 instances.

In addition to that, since the random forest model can not tune the model around the optimum, we specifically generated samples around the proximity of the minimum y_true value. To do that, we used the same approach we followed in the Bayesian optimization process' optimization approach. First, we identify the instance with the minimum y_true in the initial 3000 instance. Then, by limiting the cost allocation generation range for each instance to the $[-5, +5]$ neighbourhood we create additional 3000 instances. After that, we repeat the same process using the 6000 instances and limit the cost allocation range to $[-3, +3]$ to generate additional 3000 instances. And finally, repeat the process for the cost allocation range of $[-1, +1]$.

In summary, we initialize 3000 instances using $[0, 20)$ interval for each cost allocation. Then, by reducing the cost allocation interval to $[(Best\_allocation-5), (Best\_allocation+5))$, we create additional 3000 instances. Then, we create additional 3000 instances with the cost interval of $[(Best\_allocation - 3), (Best\_allocation + 3))$. Finally, we generate 3000 instances with $[(Best\_allocation - 1), (Best\_allocation + 1))$ interval. This iterative approach allows us to sample points from different points and to teach the model better around the optimum values. In the end, we generate a total of 12000 data points from different allocation ranges.

Once we generate instances to develop models on, we split the data into train, test, and validation data sets. Training data set has 800 instances, test data set has 3200, and validation data set has 8000 instances. In addition to that, the number of independent variables in each data set is equal to the period of the game theory setup as each column represent the cost associated in that period. While splitting the data into different data sets, the data generation range, i.e. allocation range interval design, is used as a stratification parameter so that each data set has representation from different data generation methods. Also, as we aim to develop a stable model with the least amount of data possible to decrease the data generation costs for future problems, we define our training data set with a lower number of instances than the test data set. In few cases, we increase the training data set size and decrease the test data set size to see if we can observe a better model that can identify cost allocation instances that beats the base pattern allocation.

After splitting the data into different data sets, we start the modelling process. In our study, we develop random forest models to identify the relation between cost allocations in time horizons and the y_true. Using that information, we then evaluate the feature space for each cost allocation without any additional cost. Assuming that we develop an accurate representation of the relation between the cost allocations and the target, our methodology has the advantage of finding the maximum percentage

deviation from stability value for each allocation with minimal additional time loss.

With this objective, using training data set, we create a pipeline that first scales the input variables using a standard scaling and then develops a random forest model with a wide hyper parameter combination. We also use a 3 fold cross validation to evaluate our model performance. The model with the minimum mean absolute error in the test data sets of the cross validation process is selected. Except for the $max\_features$ hyper parameter, the remaining hyper parameters remained same across 10 period game and 20 period game. In 10 period game, since we have 10 input variables with each one representing the cost allocation in a single period, we use 9 and 10 as the $max\_features$ hyper parameter.

To evaluate the model performance, in addition to the mean absolute error, we also analysed the correlation between the target and the prediction. Since our objective is to find the cost allocations with minimum target value, we are more interested in the order of the predictions than the exact accuracy of the predictions. This is why, we also evaluated the correlation between target and prediction in each of the data sets.

Once we select the best model, we use that model to predict the maximum percentage deviation from stability value for a wider set of instances. For this step, we use the validation data set with 8000 instances. Since this data set has 2000 instances generated for each of the data generation range, it has a higher chance of containing the instance with the minimum of the maximum percentage deviation from stability value.

After calculating the model predictions for the validation data set, we then find the instance with the minimum predicted maximum percentage deviation from stability, hereinafter referred to as y_pred_RF. Then, using base pattern allocation method, we identify the instance that has the minimum predicted maximum percentage deviation from stability value. By comparing the y_true values of the instance obtained using

random forest and base pattern allocation, we then determine which approach is able to find the minimum actual maximum percentage deviation from stability value. If the y_true value of the random forest model's instance with y_pred_RF is smaller than the y_true value of the instance obtained with the base pattern allocation (BPA), then we conclude that the random forest model performs better than the BPA approach for that setting. If not, we conclude that the BPA performs better than the random forest model approach.

To better conclude if random forest modelling approach performs better or worse than the BPA approach, we test our hypothesis on 10 different data sets using 10 different seeds to generate data for the 10 period game theory setup. For 20 period game theory setup, we obtain conclusive results after 6 different data sets. Hence, we keep the number of different data sets for 20 period to 6. For 20 period game theory setup, we also created a scenario where instead of creating a 20 period game from scratch; a scenario where we add 2 different 10 period games back to back and compare the model performance with the BPA allocations in that scenario.

Table 2 below shows an example cost allocations in a 10 period horizon game. Each row with index $ALLOC\_n$ represents the total cost allocation in the given period. For the example below, total cost allocation for all players on first period is 12.1729386 and for second period it is 9.51044468. With the given total cost allocations, the maximum percentage deviation from stability (y_true) is calculated as 0.61253817.

Table 2: Cost allocation example in a 10 period horizon setting with maximum deviation from stability(y_true)

ALLOC_0	12.1729386
ALLOC_1	9.51044468
ALLOC_2	16.842075
ALLOC_3	23.1619437
ALLOC_4	10.4747727
ALLOC_5	19.8129805
ALLOC_6	19.1945124
ALLOC_7	5.1675646
ALLOC_8	10.6445909
ALLOC_9	7.03173788
y_true	0.61253817

Table 3 below shows a similar example as the Table 2, but in a 20 period game. Similarly, rows with $ALLOC\_n$ values represent the total cost allocation in the given period. For the example below, total cost allocation for all players on first period is 9.46675567 and for second period it is 15.4083293. With the given total cost allocations, the maximum percentage deviation from stability (y_true) is calculated as 0.82839586.

Table 3: Cost allocation example in a 20 period horizon setting with maximum deviation from stability(y_true)

ALLOC_0	9.46675567
ALLOC_1	15.4083293
ALLOC_2	17.3774348
ALLOC_3	21.494456
ALLOC_4	18.2923944
ALLOC_5	20.78205
ALLOC_6	8.15733615
ALLOC_7	20.8626675
ALLOC_8	19.947825
ALLOC_9	5.27439779
ALLOC_10	10.0964381
ALLOC_11	2.99483334
ALLOC_12	6.53614583
ALLOC_13	17.8488217
ALLOC_14	7.27704363
ALLOC_15	8.17208113
ALLOC_16	4.28112061
ALLOC_17	14.3733841
ALLOC_18	0.4126977
ALLOC_19	1.0504527
y_true	0.82839586

CHAPTER III

RESULTS

In this chapter the results of the Bayesian optimization and Random Forest modelling will be shared.

3.1 Bayesian Optimization Results

In this section, the results of the Bayesian optimization process will be shared. In each sub-section, the process results for a different time period will be shared.

3.1.1 Results for 3 Period

For a lot sizing game with 3 periods, we evaluate a total of 13 different optimization scenarios. All trials with their respective optimization setup parameters are presented in Table 4 and Table 5. On each columns, the setups for different trials are represented with the initial column describing the scenario and hyper-parameter type.

The generic parameters are "TRIAL_ID", "Game Period", "# Samples in Each Opt", and "# Initial Samples". The "TRIAL_ID" row represents the unique identifier for each of the scenarios in the given scenario. "Game Period" row represents the time horizon in the given setting. In this sub-section, the results for a 3 period game are presented. "# Samples in Each Opt" row represents the number of samples in each optimization row as described as the number of instance in each loop in Section 2.1.1. "# Initial Samples" row represents the initial number of instances that are used to initialize the optimization process to reduce cold start.

In addition to the generic parameters, we also define optimization step specific parameters such as "Opt Data Generation Space", "# Optimization Loop", "Opt

Function", and "Opt Function Beta". These parameters are defined in each optimization loop and each different loop is represented with the suffix "#n" with n indicating the id of the optimization step. As described in Section 2.1.1, each optimization step is run "# Optimization Loop" number of times with "Opt Data Generation Space", "Opt Function", and "Opt Function Beta" staying constant within each step. "Opt Data Generation Space" parameter represents the data generation space for each optimization step in a given loop. This parameter is described in more detail in Section 2.1.1 as cost allocation range in each instance. "# Optimization Loop" parameter represents the number of times the loop is run with given parameters. This parameter is also explained in Section 2.1.1 under the number of optimization loops name. "Opt Function" row represents the acquisition function used in the optimization process. As described in the Section 2.1.1, we only use upper confidence bound acquisition function. Finally, "Opt Function Beta" parameter represents the β parameter of the upper confidence bound acquisition function as described in Section 2.1.1.

Table 4: Bayesian Optimization Process Setup Part 1 in a 3 Period Game

TRIAL_ID	1	2	3	4	5	6	7	8
Game Period	3	3	3	3	3	3	3	3
# Samples in Each Opt	1000	1000	1000	1000	1000	1000	1000	1000
# Initial Samples	20	20	300	300	20	20	5000	500
Opt Data Generation Space #1	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space
# Optimization Loop #1	100	100	20	50	20	200	50	100
Opt Function #1	UCB	UCB	UCB	UCB	UCB	UCB	UCB	UCB
Opt Function Beta #1	1000	1000	1000	0.01	10	100	0.01	0.01
Opt Data Generation Space #2	Entire Space	BEST_+-1	Entire Space	BEST_+-2	BEST_+-1	Entire Space	-	-
# Optimization Loop #2	100	50	50	50	50	200	-	-
Opt Function #2	UCB	UCB	UCB	UCB	UCB	UCB	-	-
Opt Function Beta #2	0.01	0.0001	0.01	0.01	0.01	0.01	-	-

Table 5: Bayesian Optimization Process Setup Part 2 in a 3 Period Game

TRIAL_ID	9	10	11	12	13
Game Period	3	3	3	3	3
# Samples in Each Opt	1000	1000	1000	1000	1000
# Initial Samples	20	300	20	20	20
Opt Data Generation Space #1	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space
# Optimization Loop #1	20	50	500	50	50
Opt Function #1	UCB	UCB	UCB	UCB	UCB
Opt Function Beta #1	1000	10	1000	0.01	0.01
Opt Data Generation Space #2	BEST_+-5	BEST_+-5	Entire Space	BEST_+-5	BEST_+-5
# Optimization Loop #2	50	20	500	50	50
Opt Function #2	UCB	UCB	UCB	UCB	UCB
Opt Function Beta #2	10	10	0.01	0.01	0.01

Table 5 continued from previous page

TRIAL_ID	9	10	11	12	13
Opt Data Generation Space #3	BEST_+-3	BEST_+-3	BEST_+-5	BEST_+-3	BEST_+-3
# Optimization Loop #3	50	50	50	50	50
Opt Function #3	UCB	UCB	UCB	UCB	UCB
Opt Function Beta #3	0.01	0.01	0.01	0.01	0.01
Opt Data Generation Space #4	BEST_+-1	-	BEST_+-1	BEST_+-1	BEST_+-1
# Optimization Loop #4	50	-	50	50	50
Opt Function #4	UCB	-	UCB	UCB	UCB
Opt Function Beta #4	0.01	-	0.01	0.01	0.01
Opt Data Generation Space #5	-	-	Entire Space	-	BEST_+-3
# Optimization Loop #5	-	-	50		50

Table 5 continued from previous page

TRIAL_ID	9	10	11	12	13
Opt Function #5	-	-	UCB	-	UCB
Opt Function Beta #5	-	-	0.01	-	0.01

We share in more detail the results for the trials 1,2,7,5,9, and 13 to share our findings on the impact of the initialization, data generation space, and β multiplier. To evaluate the model performance, we analyze the difference between exact solution of y_true obtained with CPLEX library and the prediction of the Gaussian process y_pred. For each trial, we share the graph visualizing the difference between y_true and y_pred.

First, to analyze the impact of the number of initial samples, we compare the trials 1 and 7. In trial 1, we initialize the process with 20 instances and run a total of 200 optimization loops. In Trial 7, we initialize the process with 5000 instances and run a total of 50 optimization loops. In trial 7, we observe that the optimization process predicts the y_true right from the first optimization loop with minimal errors. In Trial 1, the optimization process is not able to predict accurately the actual distribution of y_true values. To better visualize the results, the last 20 initialization instances of Trial 7 is represented in the Figure 4 below. The results show that the number of initial instances increases the model convergence. Lines in the Figure 4 represent the error of the Bayesian optimization process' predictions from the true deviation from stability value. The positive values in the errors mean that the Bayesian optimization process' predictions are lower than the actual values and the negative error values mean that the Bayesian optimization process' predictions are higher. As it can be seen in the Figure 4, the high number of initial examples allows the process to reach

convergence right from the beginning of the optimization process, whereas in Trial 1, due to the low number of initial instances, the model struggles to converge even after many optimization rounds.

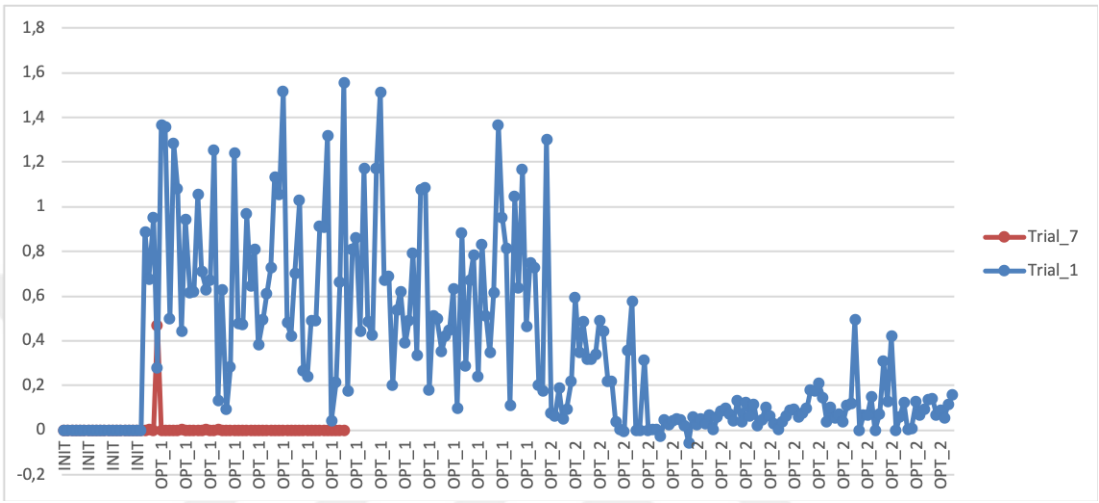

Figure 4: Trial 1 and Trial 7 Conversion Comparaison

Secondly, we analyze the impact of the search space on the convergence of the algorithm. To analyze that, we evaluate the results for Trial 13. In Trial 13, after generating initial 20 instances, we optimize the process with 50 instances in the entire feature space, 150 instances around the $[-5, +5)$, $[-3, +3)$, and $[-1, +1)$ neighbourhoods of the best allocation, with 50 instance each. In addition to that, we also increase the neighbourhood to $[-3, +3)$ again to see if the model actually learns the general distribution or only around the neighbourhood. The results in the Figure 5 show that as the neighbourhood interval decreases, model performance increases. However, increasing the neighbourhood after the optimization with reduced feature interval impacts the model performance and shows that the model only learns the given neighbourhood and is not suitable for generalization. The red line represent the increased feature search space after sequentially decreasing the feature space. The line represents the error of the Bayesian optimization process' predictions from the true deviation from stability value in the same way as it does in the Figure 4. It

can be seen in the Figure 5 that although the optimization process reaches near zero prediction errors at the end of the fourth optimization step, when we increase the feature search space, the model errors increase. Thus, we deduce that increasing feature search space after training the process with smaller feature space reduces the predictive power.

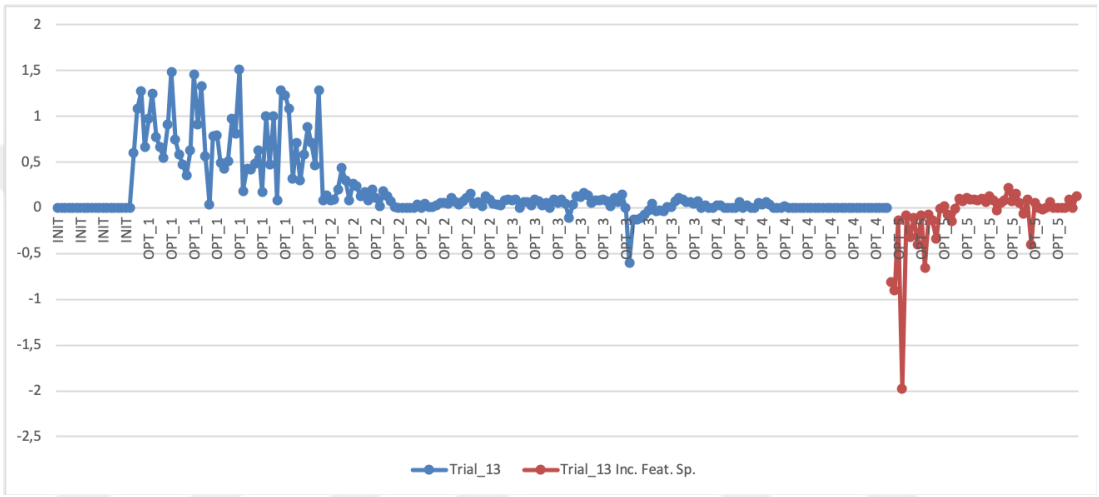

Figure 5: Trial 13 Search Space Impact

Finally, to observe the impact of the β in the upper confidence bound parameter, we compare the Trial 2 and Trial 12's first optimization processes. The trial 2's β parameter is set to 1000 and the trial 12's β parameter is set to 0.01. They both have 20 initialization instances and 50 optimization instances. Although the β of the trial 12 is 100,000 times of the trial 2's β, the results did not show a clear impact of the β parameter of the upper confidence bound acquisition function. The Figure 6 below shows the prediction errors with the Trial 2 and Trial 12. As it can be seen in the Figure 6, it is not possible to make a clear deduction on the impact of β parameter of the upper confidence bound acquisition function.

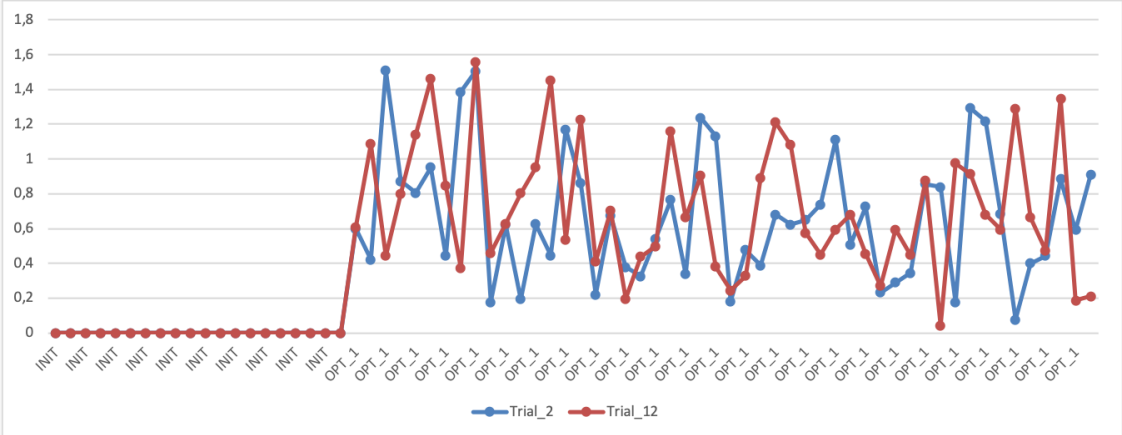

Figure 6: β Impact on Trials 2 and 12

For the 3 period results, we observe that the limitation of the feature search space has a much more significant impact on the model convergence rate in addition to the number of initial instances. In addition to that, β value did not show a clear impact on the model convergence performance.

3.1.2 Results for 4 Period

For the lot sizing problem in a 4 period horizon, we follow the same approach as the 3 period horizon. For 4 period horizon, we generate a total of 8 different data-sets, 5 of them are created using one seed, and the 3 remaining data-sets are created using another to also increase variability. Table 6 below shows the setups of each trial. The data points for trials 1 to 5 are generated using the same seed, while the data points for trials 6 to 8 are generated using another. The table configuration is the same as the Table 4 and the Table 5. For more detailed description of the table, we refer the reader to section 3.1.1.

Table 6: Bayesian Optimization Process Setup in a 4 Period Game

TRIAL_ID	1	2	3	4	5	6	7	8
Game Period	4	4	4	4	4	4	4	4
# Samples in Each Opt	1000	1000	1000	1000	1000	1000	1000	1000
# Initial Samples	20	20	100	5000	1000	5000	200	200
Opt Data Generation Space #1	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	-	Entire Space	Entire Space
# Optimization Loop #1	50	100	100	1	1000	-	1	150
Opt Function #1	UCB	UCB	UCB	UCB	UCB	-	UCB	UCB
Opt Function Beta #1	0.01	0.01	0.01	0.01	0.01	-	0.01	0.01
Opt Data Generation Space #2	BEST_+-5	BEST_+-5	BEST_+-5	Entire Space	Entire Space	-	BEST_+-5	BEST_+-5
# Optimization Loop #2	50	100	150	50	1000	-	150	150

Table 6 continued from previous page

TRIAL_ID	1	2	3	4	5	6	7	8
Opt Function #2	UCB	UCB	UCB	UCB	UCB	-	UCB	UCB
Opt Function Beta #2	0.01	0.01	0.01	0.01	0.01	-	0.01	0.01
Opt Data Generation Space #3	BEST_+-3	BEST_+-3	BEST_+-3	-	Entire Space	-	BEST_+-3	BEST_+-3
# Optimization Loop #3	50	100	150	-	1000	-	150	150
Opt Function #3	UCB	UCB	UCB	-	UCB	-	UCB	UCB
Opt Function Beta #3	0.01	0.01	0.01	-	0.01	-	0.01	0.01
Opt Data Generation Space #4	BEST_+-1	BEST_+-1	BEST_+-1	-	-	-	BEST_+-1	BEST_+-1
# Optimization Loop #4	50	100	150	-	-	-	300	300
Opt Function #4	UCB	UCB	UCB	-	-	-	UCB	UCB
Opt Function Beta #4	0.01	0.01	0.01	-	-	-	0.01	0.01

In a 4 period horizon game, in Trial 4, we observe that even with a 5000 initial samples, the process is not able to converge when the entire parameter search space is used. This is different than what we observe in a 3 period setting. Although in a 3 period game, the algorithm converges immediately after 5000 initial samples, in a 4 period game, the algorithm is not able to converge even after 50 optimization steps. The reason behind this result is that as the number of period increases, the combinatorial complexity of the model increase as well. This means that there are more X_n total lot sizing cost for each period that result in similar y_true values as the number of period increases. This in turn, makes predicting accurate results for a given set of input variables more complicated. The Figure 7 shows the impact of the number of periods in the performance of the Bayesian optimization process. To better visualise the effect of the number of periods, only the last 100 initialization instance is presented in the Figure 7. The blue line in the Figure 7 shows the error of the Bayesian optimization process' predictions from the true deviation from stability value in the Trial 7 of the 3 period setup. The red line represents the error of the Trial 4 of the 4 period setup. We observe that although the same setup in the 3 period setup converges immediately after initialization, it could not converge even after 50 optimization steps in a 4 period game. Thus, we can deduce that the number of period increases the complexity of the problem.

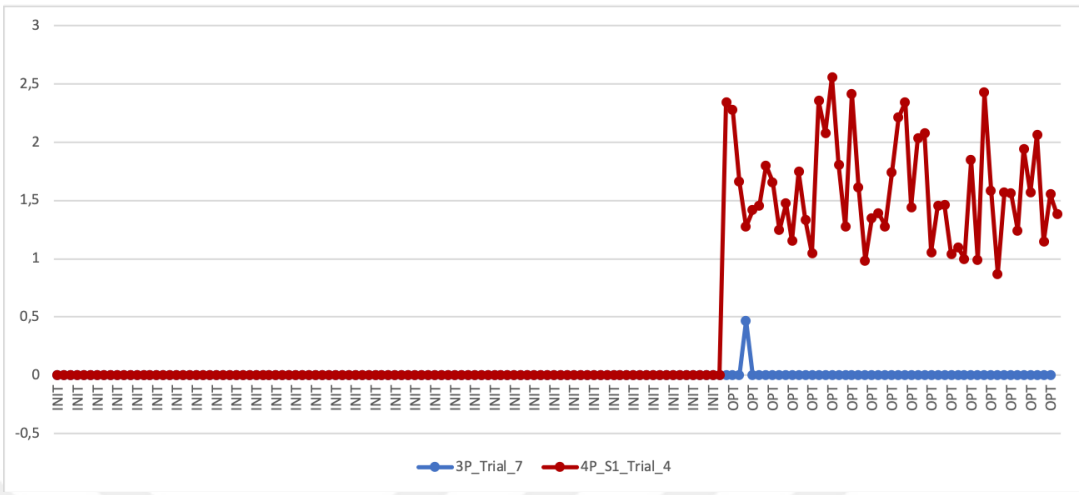

Figure 7: Impact of Number of Periods in Convergence using 3 Period and 4 Period Setups

In addition to the impact of the initial set of instances, the optimization process still could not converge even after a significant amount of optimization rounds using the entire search space. In Trial 5, we generate 1000 initial instances and a total of 3000 optimization rounds in 3 different optimization rounds. The model still is not able to accurately predict the y_true values of a given allocation. Comparing Figure 8 and Figure 9 shows that in a 4 period setup, the model prediction error highly fluctuates even with a much larger optimization instances. In Figure 9, only the first 2 optimization rounds of the trial 11 that uses the entire feature space to optimize the process has been visualized. In Figure 8, the error distribution of the 1000 initialization and 3000 optimization rounds are visualized. The figures show that the model errors in the 3 period setting with 20 initialization and 1000 optimization instances are much lower than the model errors in the 4 period settings with 1000 initialization and 3000 optimization instances.

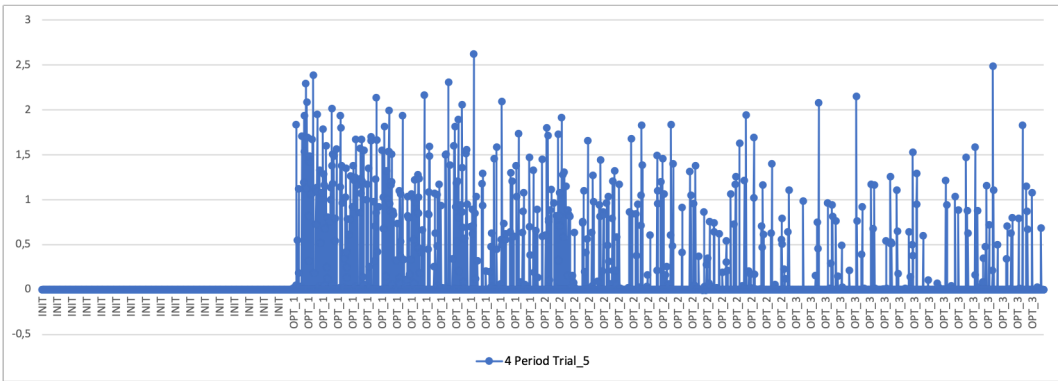

Figure 8: Conversion Difficulty of Bayesian Optimization in a 4 Period Setup

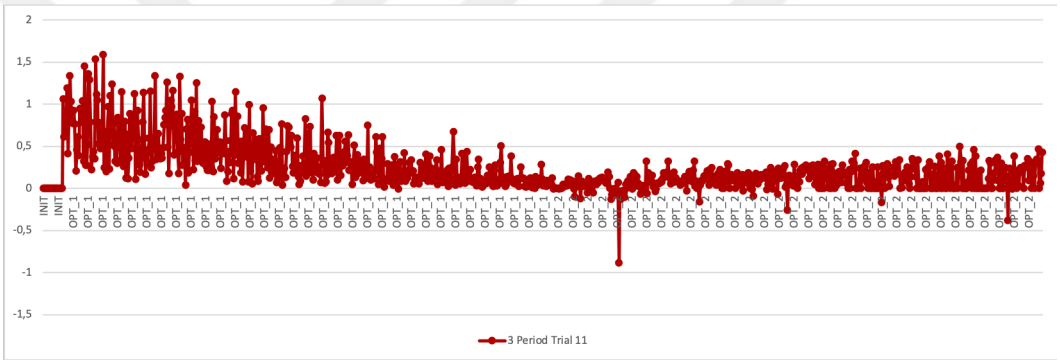

Figure 9: Conversion Performance in a 3P Setting in Trial 11

Finally, in Trial 3 of the 4 period game, we generate an initial set of 100 samples and run an optimization loop using entire feature space for 100 instances. Then, we reduce the feature generation space to the $[-5, +5)$, $[-3, +3)$, and $[-1, +1)$ neighbourhoods of the best allocation, with 150 instance each. We observe that although model converges, it converges to a local minimum. We deduce that by comparing the minimum y_true values obtained in Trial 5 to the minimum y_true obtained in Trial3. As described in the paragraph above, Trial 5 contains 1000 initialization instances and 3000 optimization instances. By comparing the y_true obtained in Trial 3 and Trial 5, we analyze the impact of the reduced search space on the model performance. Trial 3 uses a reduced feature space optimization while trial 5 uses the entire feature space. Table 7 shows that the minimum actual target value obtained in trial 5 is smaller

than that of trial 3. This means that although the trial 3 converges, it converges to a local minimum. Thus, reducing feature space introduces additional problems.

Table 7: Actual Deviation from Stability Comparison in Trials 3 and 5

Trial Reference	**Min Actual Deviation**
4P Trial 3	0.06
4P Trial 5	0.00

3.1.3 Results for 10 Period

After observing the complexity increase with the increased number of periods, for a 10 period game, we only develop an optimization process using the reduced feature space setup in the optimization step. We conduct 9 different trials for the Bayesian optimization approach in a 10 period game. Table 8 shows the Bayesian optimization process setups in each of the trials.

Table 8: Bayesian Optimization Process Setup in a 10 Period Game

TRIAL_ID	1	2	3	4	5	6	7	8	9
Game Period	10	10	10	10	10	10	10	10	10
# Samples in Each Opt	1000	1000	1000	1000	1000	1000	1000	1000	1000
# Initial Samples	100	100	100	500	500	5000	250	500	1000
Opt Data Generation Space #1	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space	Entire Space
# Optimization Loop #1	100	100	100	300	250		250	250	250
Opt Function #1	UCB	UCB	UCB	UCB	UCB		UCB	UCB	UCB
Opt Function Beta #1	0.01	0.01	0.01	0.01	0.01		0.01	0.01	0.01
Opt Data Generation Space #2	BEST _+-5	BEST _+-5	BEST _+-1	BEST _+-1	BEST _+-3		BEST _+-1	BEST _+-3	BEST _+-1

Table 8 continued from previous page

TRIAL_ID	1	2	3	4	5	6	7	8	9
# Optimization Loop #2	150	150	150	300	250		250	250	2000
Opt Function #2	UCB	UCB	UCB	UCB	UCB		UCB	UCB	UCB
Opt Function Beta #2	0.01	0.01	0.01	0.01	0.01		0.01	0.01	0.01
Opt Data Generation Space #3	BEST _+-3	BEST _+-3	BEST _+-1		BEST _+-1			BEST _+-1	BEST _+-1
# Optimization Loop #3	150	150	150		250			500	500
Opt Function #3	UCB	UCB	UCB		UCB			UCB	UCB
Opt Function Beta #3	0.01	0.01	0.01		0.01			0.01	0.01
Opt Data Generation Space #4	BEST _+-1	BEST _+-1	BEST _+-1		BEST _+-1			BEST _+-1	

Table 8 continued from previous page

TRIAL_ID	1	2	3	4	5	6	7	8	9
# Optimization Loop #4	150	150	150		250			500	
Opt Function #4	UCB	UCB	UCB		UCB			UCB	
Opt Function Beta #4	0.01	0.01	0.01		0.01			0.01	
Opt Data Generation Space #5					BEST _+-1				
# Optimization Loop #5					250				
Opt Function #5					UCB				
Opt Function Beta #5					0.01				

In Trial 9, we initialize 1000 samples with entire feature space. After that, we run the optimization step for 250 iterations. After that, we reduce the feature search space to the $[-1, +1)$ neighbourhood of the best lot sizing cost allocation result and run the optimization step a total of 2500 times. Since Bayesian optimization process predicts a distribution with the mean and the standard deviation, in addition to the prediction mean, we also analysed the standard deviation in the predictions. By looking at the means and their difference from actual target values, we observe that the model seems to converge and reduce its errors. Figure 10 shows that the prediction increases as the number optimization rounds performed increases.

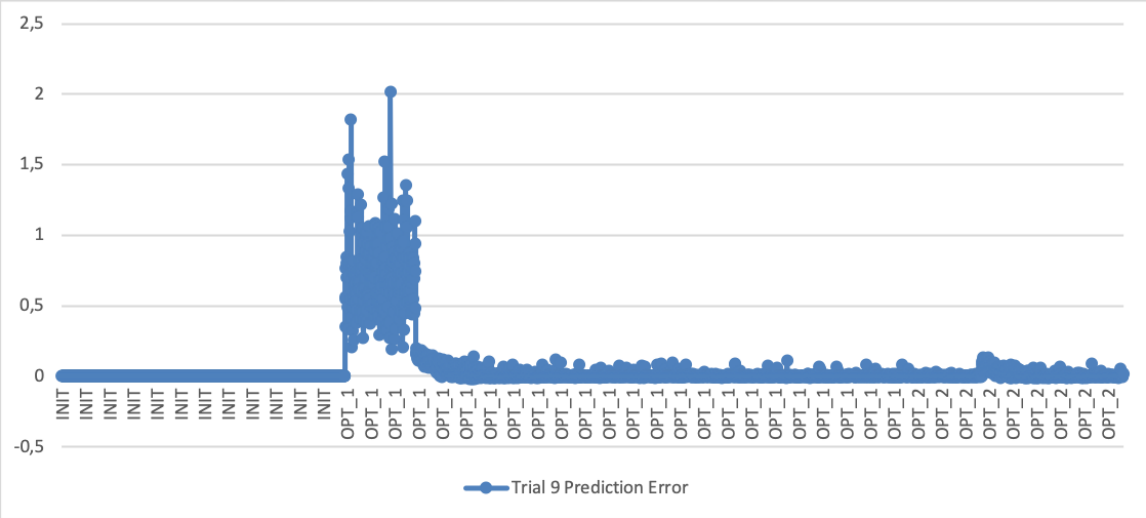

Figure 10: Bayesian Optimization Process Prediction Error Distribution in 10 Period Setting

However, when we analyse the standard deviations in the Bayesian optimization process' predictions, we observe a different result. Figure 11 shows the evaluation of the prediction standard deviation as the optimization rounds increases. Although in Figure 10 we observe an improvement with the optimization, the standard distribution in Figure Figure 11 shows that the model predictions contain high variability. Thus, we conclude that the model requires further optimization.

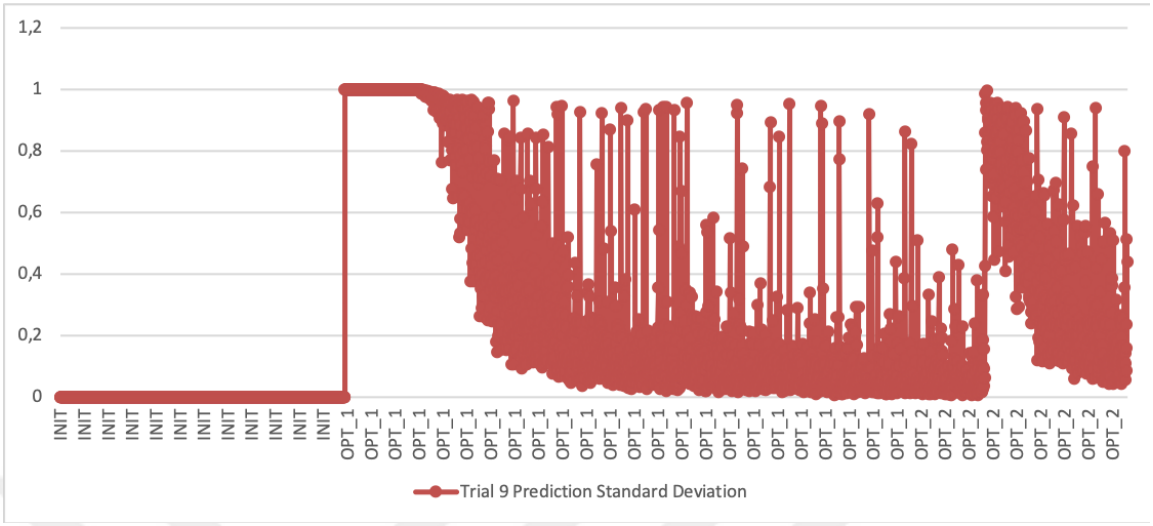

Figure 11: Bayesian Optimization Process Prediction Standard Deviation Distribution in 10 Period Setting

Even after 2500 rounds of optimization and a tighter reduced feature search space than in 3 period and 4 period games, the model still requires further optimization rounds. This proves that the complexity of the process increases as the number of periods increase.

In addition to that, time cost of the algorithm significantly increases as the number of periods increase as well. Table 9 below shows the time cost with related to the increased number of periods. In a 4 period game, the optimization process with 2000 optimization loops after an initial 1000 instance generation takes 20 minutes. In 10 period game, same process takes 90 minutes which is more than 4 times of the time it takes in a 4 period game. This demonstrates that the feasibility of this approach will be highly limited as the number of periods increase. In addition to the optimization process time costs, data generation process is also much more expensive in a 10 period game than a 4 period game.

Table 9: Time Complexity of Increased Period in Lot Sizing Cost Calculation Using Bayesian Optimization

Number of Periods	Number of Initialization Rounds	Number of Optimization Rounds	Time Took
4	1000	2000	20 minutes
10	1000	2000	90 minutes

3.2 Random Forest Model Results

We observe that the Bayesian optimization approach does not scale well as the number of period increases. For this reason, we decide to evaluate the random forest models to see if they can learn the relation between lot sizing costs and the target to better predict the y_true.

We develop random forest models for both 10 period games and 20 period games. Additionally, using two of our 10 period game models, we evaluate if we can better predict a 20 period game than a single 20 period game.

In this section, we share the results obtained in each method.

3.2.1 Results for 10 Period

Using the same process we use in the Bayesian optimization's data initialization step, we generate a data set of 4000 instances to train and test our model. We also generate an additional validation data of 8000 instances to evaluate using our model to better search the parameter space. In Section 2.1.2, we share the modelling methodology in more detail. We refer the reader to releated section for further information.

In our study, we generate 10 different data sets using 10 different seeds. In this sub section, we share the model performance in each of those 10 data sets. To evaluate if our model generates better predictions than the base pattern allocation method, we score the validation data of 8000 instances. By taking the instance with minimum

y_pred, we compare its *y_true* value with the *y_true* value of the instance generated using the base pattern allocation approach. If *y_true* of the base pattern allocation is lower, than for that data set, the base pattern allocation performed better. If not, the random forest approach performs better.

In our modelling approach, we first normalized the inputs and then, developed the random forest models using the following python dictionary of hyperparameters: 'max_depth': [4,5,10,15,20,21,22,23,25,None], 'max_features': [9,10], 'min_samples_leaf' : [1,2,5,10], 'n_estimators': [50,100,150,200]. The best model is selected using a 3 fold cross validation and the model with minimum mean absolute error in validation data sets are selected.

Table 10: 10 Period Random Forest Model Results

	Correlation Test	Correlation Validation	BPA Target	BPA Prediction	Model Best Target	Model Best Prediciton	Best Approach
Seed 1	0.958	0.951	0.108	0.129	0.066	0.053	RF Model
Seed 2	0.934	0.886	0.082	0.126	0.055	0.010	RF Model
Seed 3	0.927	0.872	0.149	0.258	0.133	0.144	RF Model
Seed 4	0.920	0.898	0.089	0.240	0.144	0.144	BPA
Seed 4 Train Size Increase	0.953	0.934	0.089	0.190	0.024	0.122	RF Model
Seed 5	0.941	0.927	0.144	0.104	0.017	0.025	RF Model
Seed 6	0.948	0.944	0.120	0.127	0.109	0.079	RF Model
Seed 7	0.933	0.903	0.170	0.388	0.104	0.095	RF Model
Seed 8	0.942	0.887	0.101	0.219	0.140	0.107	BPA
Seed 9	0.941	0.924	0.080	0.177	0.172	0.145	BPA
Seed 10	0.929	0.895	0.045	0.117	0.114	0.070	BPA

Table 10 shows the random forest model results in different data sets. For 7 out of 10 different data sets and lot sizing problem settings generated with different seeds, the random forest model performed better than the base pattern allocation approach. For the seed 4, the random forest model with the initial training size performed worse than the base pattern allocation approach. But after increasing the training set size to 2800, the model trained using random forest performed better than the base pattern allocation approach and the results are shown in the row with the index "Seed 4 Train Size Increase".

To compare the results of the random forest model and the base pattern allocation approach, we first calculate for each different problem, the best cost allocation scenario using base pattern allocation and calculate its actual target value using the exact methods in the CPLEX implementation of Python. Then, using the random forest model that is developed using the approach described in 2.1.2, we predict the maximum deviation from stability value of each cost allocation in the validation data set. Finally, we select the allocation with minimum predicted maximum deviation from stability value in the validation data set as the best cost allocation using random forest. By comparing the target values of the bbest cost allocation using random forest and the base pattern allocation, we determine if the random forest model is able to beat the base pattern allocation approach.

In addition to that, we also developed a single random forest model that aims to predict the cost allocations independently from data. To achieve that, in addition to the cost allocations in the training data set, we also include the problem setups so that model can learn the problem setup as well. This means that, in addition to the total cost of allocations in each period, we also included back order costs, capacities, demands, fixed costs, holding costs, and variable costs in each period in order to develop a data independent model. The results for this trial are presented in the Table 11 below. In order to develop such model, we combined the data sets that

are generated using seeds 1 to 7 into the training set. As the test set, we used the remaining 3 data sets and compared the base pattern allocation's target values with the target values of the allocations with minimum random forest model predictions. The random forest model predictions are represented in "RF Best's y_pred" colum in Table 11 and their target values are represented in "RF Best's y_true" column. The base pattern allocation's target values are presented in the "BPA's y_true" column.

The results in Table 11 show that although the model performs better in all the data sets in the training sample, it only performed better in 1 out of 3 data sets in the test datasets.

Table 11: Single Random Forest Model Using Data Sets with All 10 Seeds

Data Origin	**Seed**	**RF Best's y_true**	**RF Best's y_pred**	**BPA's y_true**	**Best Approach**
Train	1	0.032981	0.044316	0.108063	RF
Train	2	0.055807	0.148826	0.081568	RF
Train	3	0.117499	0.102263	0.148534	RF
Train	4	0.052013	0.093096	0.088967	RF
Train	5	0.018273	0.026718	0.143686	RF
Train	6	0.109286	0.094507	0.12018	RF
Train	7	0.103978	0.094847	0.169811	RF
Test	8	0.070037	0.129194	0.101405	RF
Test	9	0.110725	0.093971	0.079525	BPA
Test	10	0.140888	0.112888	0.045034	BPA

3.2.2 Results for 20 Period

Since the model results show potential in random forest, we also test the model performance in a wider time period of 20. We used the same approach to compare the random forest model's performance compared to the base pattern allocation approach

in 20 period game as we used in the 10 period game.

As we observed a performance increase in one setting of the 10 period game, we increase the number of instances in the data sets. For model development, we use a data set of 17000 instances and for the model testing, we use a data set of 3000 instances. To find the best cost allocations, we predict the maximum deviation from stability values of a data set that consists of 40000 instances. So, for 20 period game, a total of 60000 instances are used to test if the random forest model performs better than the base pattern allocation. Other than the sample sizes, we use the same approach as the 10 period game.

The results for the 20 period game are presented in Table 12. Contrary to the 10 period game, in all the cases, base pattern allocation performed better than the random forest model. Thus, we only developed random forest models for 6 different seeds. As we do not observe any case with better cost allocations using random forest model, we limit our model development approach to 6 different data sets.

Table 12: 20 Period Random Forest Model Results

	Correlation Test	**Correlation Validation**	**BPA Target**	**BPA Prediction**	**Model Best Target**	**Model Best Prediciton**	**Best Approach**
Seed 1	0.966	0.941	0.141	0.396	0.275	0.337	BPA
Seed 2	0.983	0.959	0.05	0.401	0.434	0.358	BPA
Seed 3	0.982	0.958	0.101	0.36	0.102	0.283	BPA
Seed 4	0.984	0.954	0.102	0.368	0.227	0.349	BPA
Seed 5	0.984	0.961	0.091	0.341	0.352	0.318	BPA
Seed 6	0.981	0.938	0.093	0.359	0.379	0.302	BPA

3.2.3 Results for 10+10 Period

As the base pattern allocation model performed better in the 20 period game, we also analyze the impact of the scenario setup. By dividing a 20 period game into 2 separate 10 period games and using 2 random forest models, we analyze the performance of the model.

To do that, we use the data sets generated using seed 1 and seed 2 in the 10 period game of the random forest model setup. By creating a new scenario of a 20 period game using 2 separate 10 period games, we are able to use the predictions in the 10 period games. To achieve that, we used the best cost allocations using random forest model in each of the game. In addition to the cost allocations, we also collect the game setup parameters, e.g. back order cost, capacity, demand etc. By concatenating the cost allocations of the 2 games and normalizing them using the combined game setup, we obtain a single cost allocation that is made of 2 different 10 period game. In addition to that cost allocation, we also calculate the best cost allocations of the new 20 period game using base pattern allocations and compare their maximum deviation from stability values.

Table 13 below shows the result of this approach. Also in this case, the base pattern allocation performs better than the random forest model approach.

Table 13: Random Forest 20 Period Approach Using 10+10

	BPA Target	**Model Best Target**	**Best Approach**
Seed 1 & Seed 2 10P	0.115	0.125	BPA

CHAPTER IV

CONCLUSIONS

In this paper, our objective is to find the optimal joint cost allocation in a capaciatated dynamic lot sizing problem with different time horizons using Bayesian optimization and machine learning.

Using Bayesian optimization, we observe that the complexity of the problem increases as the number of periods in the game increases due to combinatorial complexity. We show that in a 3 period game, the model conversion performance is sufficient while for the 4 period and 10 period games, the Bayesian optimization approach is not practical.

In addition to that, we observe that the model performance is highly impacted by the feature search space used in the Bayesian optimization algorithm. Especially when the number of period increases, the model conversion performance is dependant on the search space for each cost allocation being small for the algorithm to converge. This in turn increases the risk of converging to a local optimum.

Also, we observe that the increased time cost of the Bayesian optimization as the number of period increases makes the Bayesian optimization approach impractical under the current computational capacity.

For the random forest models, we observe promising results for the models on their respective data sets in a 10 period game setup. However, we show that for the 20 period setup, model performances reduce significantly. Even after splitting the 20 periods into two individual 10 periods, we observe no change in the model performance.

Lastly, we show that including problem setting in the model inputs and generating

higher number of samples within different data sets generated for different problems can provide larger use-cases. Due to time constraints and computational resource limitations, we have not been able to elaborate the increased number of different data sets in order to develop a model that can be applicable in any given problem setup. Further research can be done to develop a single model that is independent from the problem setup with a higher number of instances from high number of problem setups.

REFERENCES

[1] L. Chen, *Curse of Dimensionality*, pp. 545–546. Boston, MA: Springer US, 2009.

[2] H. M. Wagner and T. M. Whitin, "Dynamic version of the economic lot size model," *Management Science*, vol. 5, no. 1, pp. 89–96, 1958.

[3] N. Brahimi, S. Dauzere-Peres, N. M. Najid, and A. Nordli, "Single item lot sizing problems," *European Journal of Operational Research*, vol. 168, no. 1, pp. 1–16, 2006.

[4] N. Brahimi, N. Absi, S. Dauzère-Pérès, and A. Nordli, "Single-item dynamic lot-sizing problems: An updated survey," *European Journal of Operational Research*, vol. 263, no. 3, pp. 838–863, 2017.

[5] G. R. Bitran and H. H. Yanasse, "Computational complexity of the capacitated lot size problem," *Management Science*, vol. 28, no. 10, pp. 1174–1186, 1982.

[6] M. Florian and M. Klein, "Deterministic production planning with concave costs and capacity constraints," *Management Science*, vol. 18, no. 1, pp. 12–20, 1971.

[7] H.-D. Chen, D. W. Hearn, and C.-Y. Lee, "A new dynamic programming algorithm for the single item capacitated dynamic lot size model," *Journal of Global Optimization*, vol. 4, no. 3, pp. 285–300, 1994.

[8] S. Chen, Y. Feng, A. Kumar, and B. Lin, "An algorithm for single-item economic lot-sizing problem with general inventory cost, non-decreasing capacity, and non-increasing setup and production cost," *Operations Research Letters*, vol. 36, no. 3, pp. 300–302, 2008.

[9] D. X. Shaw and A. P. Wagelmans, "An algorithm for single-item capacitated economic lot sizing with piecewise linear production costs and general holding costs," *Management Science*, vol. 44, no. 6, pp. 831–838, 1998.

[10] W. Van den Heuvel and A. P. Wagelmans, "An efficient dynamic programming algorithm for a special case of the capacitated lot-sizing problem," *Computers & Operations Research*, vol. 33, no. 12, pp. 3583–3599, 2006.

[11] K. R. Baker, P. Dixon, M. J. Magazine, and E. A. Silver, "An algorithm for the dynamic lot-size problem with time-varying production capacity constraints," *Management Science*, vol. 24, no. 16, pp. 1710–1720, 1978.

[12] V. Lotfi and Y.-S. Yoon, "An algorithm for the single-item capacitated lot-sizing problem with concave production and holding costs," *Journal of the Operational Research Society*, vol. 45, no. 8, pp. 934–941, 1994.

[13] J. M. Leung, T. L. Magnanti, and R. Vachani, "Facets and algorithms for capacitated lot sizing," *Mathematical Programming*, vol. 45, no. 1, pp. 331–359, 1989.

[14] D. Gillies, "Solutions to general non-zero-sum games. contributions to the theory of games 5 (annals of mathematics studies, 40)," 1959.

[15] M. Maschler, B. Peleg, and L. S. Shapley, "Geometric properties of the kernel, nucleolus, and related solution concepts," *Mathematics of Operations Research*, vol. 4, no. 4, pp. 303–338, 1979.

[16] W. Van den Heuvel, P. Borm, and H. Hamers, "Economic lot-sizing games," *European Journal of Operational Research*, vol. 176, no. 2, pp. 1117–1130, 2007.

[17] D. Xu and R. Yang, "A cost-sharing method for an economic lot-sizing game," *Operations Research Letters*, vol. 37, no. 2, pp. 107–110, 2009.

[18] X. Chen and J. Zhang, "Duality approaches to economic lot-sizing games," *Production and Operations Management*, vol. 25, no. 7, pp. 1203–1215, 2016.

[19] J. Drechsel and A. Kimms, "Cooperative lot sizing with transshipments and scarce capacities: solutions and fair cost allocations," *International Journal of Production Research*, vol. 49, no. 9, pp. 2643–2668, 2011.

[20] H. J. Kushner, "A versatile stochastic model of a function of unknown and time varying form," *Journal of Mathematical Analysis and Applications*, vol. 5, no. 1, pp. 150–167, 1962.

[21] H. J. Kushner, "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise," *Journal of Basic Engineering*, vol. 86, pp. 97–106, 03 1964.

[22] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference*, pp. 400–404, Springer, 1975.

[23] A. Žilinskasi, "On statistical models for multimodal optimization," *Series Statistics*, vol. 9, no. 2, pp. 255–266, 1978.

[24] A. O'Hagan, "Curve fitting and optimal design for prediction," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 40, no. 1, pp. 1–24, 1978.

[25] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 1933.

[26] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[27] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[28] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[29] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, no. 7, pp. 1545–1588, 1997.

[30] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[31] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[32] K. Fawagreh, M. M. Gaber, and E. Elyan, "Random forests: from early developments to recent advancements," *Systems Science & Control Engineering*, vol. 2, no. 1, pp. 602–609, 2014.

[33] B. Altan, A. Ekici, and O. Özener, "Cost allocation mechanisms in a capacitated economic lot-sizing game." unpublished.

[34] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2017.

[35] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-theoretic regret bounds for gaussian process optimization in the bandit setting," *IEEE Transactions on Information Theory*, vol. 58, pp. 3250–3265, may 2012.

VITA

Before attending Özyeğin University, Ömer Berkay Sarıoğlu attended Galatasaray University where he earned Bachelor of Engineering in Industrial Engineering in 2017.

Since his graduation, he took on different roles as a data scientist in the financial sector. Currently, he is working as a data scientist in a private bank.