Learning to improve evolutionary computation for a Warehouse Design and Control Problem

Coppens, Remco H.M.

Award date:
2022

Department of Industrial Engineering and Innovation Sciences
Information System Group

Learning to improve evolutionary computation for a Warehouse Design and Control Problem

Remco Coppens (1446215)

Supervisors:
Dr. Y. (Yingqian) Zhang - TU/e
Dr. Ir. L. (Laurens) Bliek - TU/e
Ir. B. (Berend) Steenhuisen - Nobleo Manufacturing

In partial fulfillment of the requirements for the degree Master of science in Operations Management and Logistics

Eindhoven, March 2022

Abstract

Simultaneous optimization of warehouse layout design and control policies leads to superior results as opposed to independently optimizing its constituent sub-problems. The problem of simultaneous optimization is referred to as the Warehouse Design and Control Problem (WDCP). The proposed solution methods consists of a Multi Objective Evolutionary Algorithm (MOEA) with Adaptive Operator Selection (AOS), implemented through Deep Reinforcement Learning (DRL). The MOEA, more specifically the Non-dominated Sorting Genetic Algorithm III (NSGA-III), conducts a guided search through the space of available solutions. Through incremental improvements it will eventually converge to an approximation of the so-called Pareto frontier, which is a set containing all optimal solutions. Despite the high performance of NSGA-III, the downside of the algorithm is the high computational cost incurred. Especially for highly dimensional optimization problem with expensive evaluation of solutions, as is the WDCP central to this research, the problem quickly becomes intractable. The effect of expensive evaluations can be mitigated by minimizing the number of evaluations needed. This can be achieved by implementing AOS in balancing the exploration-exploitation dilemma. This dilemma concerns the trade-off between using known well-performing solutions to propose new solutions, i.e. exploitation, and looking into entirely new and unknown solutions, i.e. exploration. Too much exploitation and the algorithm will converge to local optima, where too much exploration will result in an unstable model that will converge either very slowly or never at all. Besides the benefit of dynamically setting the hyperparameters dictating this dilemma, implementing AOS also prevents the need to optimize the hyperparameters of the used algorithm beforehand. Hyperparameter optimization is an expensive process, especially for the type of problems this thesis focuses on. DRL shows a suitable method for AOS, due to it being a framework for sequential decision making. Implementing DRL as the AOS within NSGA-III increased performance significantly, both in terms of convergence speed and solution quality.

Executive summary

This summary will suffice as a brief overview of the executed research project. It will describe the research problem, the research approach, the proposed solution methods, the results, conclusions and recommendations.

Research problem
In collaboration with Nobleo Manufacturing, hereinafter referred to as Nobleo, research is conducted into the Warehouse Design and Control Problem (WDCP). The WDCP concerns simultaneous optimization of warehouse layout design and the underlying control policies. More specifically, the research focuses on a collective solution to the warehouse layout design problem and both the resource and product allocation problems. Throughout literature, and often in practice, these problems are optimized independently. Often leading to sub-optimal results when collectively implemented.

The environment in which this research intends to solve this problem concerns the warehouse of a large plastic manufacturer, which is a client of Nobleo. Their current operations suffer from inefficient storage utilization, resulting in the inability to store all of their products. Through introduction of a new type of storage, Nobleo was requested to redesign the warehouse. Using advanced simulation techniques Nobleo was able to replicate the current warehouse and propose a new and improved warehouse layout design. However, the process of creating this design was cumbersome, as it was done through trial and error with no guarantee of finding the optimal configuration. Nobleo desires a model that enables finding an optimal configuration using a low computational budget.

The performance of a configuration can be described using three objective values. The main objective concerns the tardiness of outbound trucks, which summarizes the performance of internal logistics. Also the incurred resource cost is measured, enabling the model to reach optimal performance while minimizing the cost incurred. Finally the number of unplaceable products is measured, focusing on the initial problem of the client of Nobleo.

Research approach
This thesis aims to create an intelligent model for solving the WDCP. The model should be generally applicable, for current and future warehouse-related optimization problems executed by Nobleo. Besides solution quality also convergence speed plays a considerable role in development of this model. As simulations often are relatively slow, testing a large amount of configurations becomes intractable fairly quickly. The WDCP consists of a collection of underlying sub-problems, which often have solutions using Evolutionary Algorithms. To minimize the computation time of the optimization model, a process called Adaptive Operator

Selection (AOS) will be applied. Literature shows a salient method to implement AOS is Deep Reinforcement Learning (DRL). These findings led to the following main research question, which is central to this master thesis:

How can a Multi-objective Evolutionary Algorithm and Deep Reinforcement Learning be combined to simultaneously optimize warehouse layout design and control policies using a low computational budget ?

This research question contains two algorithmic definitions, namely Multi-objective Evolutionary Algorithm (MOEA) and Deep Reinforcement Learning (DRL). The MOEA concerns a branch of so-called Pareto dominated optimization methods, based upon the phenomenon of evolution as found in nature. Through incrementally improving a set of solutions, this algorithm converges towards an approximation of the set of globally optimal solutions, i.e. the actual Pareto front, which is unknown. This approximated set of solutions is presented to the decision maker, in this case Nobleo, who eventually decides which solutions are preferred for further evaluation and/or implementation. Although powerful, a MOEA is often characterized as a computationally expensive algorithm due to its need to evaluate a considerable amount of potential solutions. To alleviate this cost DRL will be implemented. DRL is able to learn to adapt the behavior of the MOEA in such a way that it will need less evaluations to get to a good approximation of the Pareto front. Through implementing DRL into a MOEA an increase in convergence speed can be attained, without decreasing the quality of the found set of solutions.

Solution methods

As previously discussed the solution method consists of two parts, an optimization algorithm and a way to minimize the computational budget. All experimentation and solution design testing is conducted on a Python replication of the actual simulation of Nobleo. The reasons to replicate the simulation are two-fold, namely to reduce computation time and to increase robustness of the simulation through enabling the model to handle extreme cases. The replicated simulation model only contains necessary computations, uses smart lookup tables and allows for parallel computation. Evaluation of a warehouse configuration consists of 40 hours in simulation time, for which the computation time is decreased from 30 minutes to on average 4 seconds. However, this decrease in computation time comes at the cost of interpretability.

The implemented MOEA concerns the Non-dominated Sorting Genetic Algorithm III (NSGA-III). This algorithm starts by initializing a given number of configurations to be tested, a so-called population of individuals. Based on the evaluation and composition of the current population the algorithm proposes a new population, called the offspring. The offspring is created through applying two operators, namely crossover and mutation. Crossover uses two individuals of the current population, called parents, to create two new individuals. By combining the composition of both parents it is intended to create offspring that is superior in performance. The process of crossover is repeated on different parent combinations until a set of offspring is created equal in size to the current population. Thereafter the offspring is taken through mutation. With a given probability, mutation changes the values of these individuals (sub-)randomly. Finally both the population and offspring are taken through a selection operator, which results in the best performing individuals comprising the new pop-

ulation and removal of less-performing individuals.

To perform well, the NSGA-III needs to strike a fair balance between exploration and exploitation. Exploitation concerns utilization of the currently well-performing individuals and is often assigned to the crossover operator. Exploration, on the other hand, involves introducing completely new parts of an individual. The intended purpose of this is to minimize the probability of converging to sub-optimal solutions and thus missing potential optimal solutions in the process. This behavior is mostly related to the mutation operator. Too much emphasis on exploitation and the algorithm will converge to local optima, where too much emphasis on exploration will lead to prolonged time needed to converge or never converging at all. Using NSGA-III the exploration-exploitation trade-off is approached through hyperparameter optimization, executed before actually running the algorithm. However this approach has two main disadvantages, for one it requires a time-consuming process of finding optimal parameters which are not generalizable. Secondly, during the actual optimization process the parameters dictating the exploration-exploitation trade-off are not adjusted. By applying Adaptive Operator Selection (AOS) these parameters are adjusted throughout the generational optimization process, showing superior results throughout literature. Besides, AOS in itself can be generalizable if the algorithm used is designed to not take problem specific information into account.

The solution method used to implement AOS in the NSGA-III concerns Deep Reinforcement Learning (DRL). As a sequential decision making framework, DRL is able to adjust the operator settings based on the current progress of optimization. The DRL algorithm used concerns Deep Q-networks, which is beneficial due to its sample efficiency and robustness attained through implementation of experience replay and a frozen target network respectively. To remain generalizable, the state representation only contains information describing the generational progress, population performance and the performance of the approximated Pareto front. The action space concerns the values for the mutation operator. These values concern the the mutation distribution parameter, describing the magnitude of change incurred by mutation, and the independent mutation probability, which implies the probability of a single value to be mutated. For the reward used to train the agent different functions are tested.

All Python code written and used throughout this research can be found on Github, using the following link: https://github.com/RemcoCoppens/Master_Thesis_Code.

Results

The first three experiments are focused on the optimization model, being the NSGA-III. The first experiment shows the performance of the NSGA-III to be superior to its predecessor, the NSGA-II, and the Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) algorithm. The second experiment concerned Bayesian hyperparameter optimization, resulting in hyperparameter settings that increased performance for both the crossover and mutation operators. Besides these hyperparameter values, this method also returned promising regions for these hyperparameters, together with their feature importances. The feature importances showed the value set for the crossover parameter to be of little to no effect on the performance. This is, most probably, due to the non-convexity of the WDCP. On the other hand, the mutation parameters showed high importance, emphasizing a predisposition of the

NSGA-III on the WDCP towards exploration. Finally a sensitivity analysis of the optimized NSGA-III showed the algorithm to be robust in extreme scenarios, maintaining its ability to converge to an approximation of the Pareto front.

The final two experiments revolved around Adaptive Operator Selection (AOS) using Deep Reinforcement Learning (DRL). Experiment four concerned testing different reward functions for learning an agent and eventually learning a DQN agent using the best performing function. These experiments and learning are all executed on a benchmark problem named DTLZ2. The reason for this concerns computation time, as a lot of evaluations are needed to be executed for an agent to learn a well-performing behavioral policy. The best performing reward function concerns an episodic reward, calculated by an overall sum of all hypervolume indicator values obtained. This metric measures the area under the curve for the hypervolume indicator, which takes both convergence speed and solution quality into account. After completing a successful learning trajectory, the behavioral policy of the agent is analyzed. Showing a surprising predisposition to fairly radical exploration. Which again shows the tendency of the NSGA-III algorithm to explore.

Eventually the agent is taken out of the train environment and placed on the actual problem central to this research. On the WDCP the agent induced NSGA-III showed superior performance compared to the optimized NSGA-III, both on solution quality and convergence speed. The maximum value found for the hypervolume indicator was 0.92 for the agent and 0.89 for the optimized NSGA-III. With regards to convergence speed, the area under the curve increased from 174 for the optimized NSGA-III to 181 for the agent induced NSGA-III. Additionally, the learned policy is also tested against a NSGA-III implementation guided by a randomly initialized DRL agent. The performance of the learned agent is superior to the random agent, indicating the value of the learned policy.

Again the behavior of the agent is analyzed, showing a different behavioral pattern over consecutive generations as opposed to the DTLZ2 benchmark. This difference indicates the inability to replace the DRL agent with a simplistic set of rules and thus shows the need for sequential decision making. Thereafter the sensitivity analysis is replicated. In three of the four experiments executed the learned agent significantly outperformed the optimized NSGA-III. For one of the experiments the performance of the agent induced NSGA-III and the optimized NSGA-III was identical. A possible explanation for this can be the fact that the optimized NSGA-III already attained the prospected maximum performance possible. These analyses show the agent induced NSGA-III to be robust, as it is able to maintain increased performance despite facing extreme scenarios.

Conclusion and recommendations

Conclusively, by combining NSGA-III with a Deep Q-Networks (DQN) agent simultaneous optimization of warehouse layout design and control policies becomes possible using a decreased computational budget. The NSGA-III guided by the learned agent shows superior performance compared to the optimized NSGA-III. This superiority is not only seen in the speed of convergence, but it also enables the algorithm to find a better performing approximation of the actual Pareto optimal set. Comparing the behavior of the agent on different problems indicates the inability to replace the agent with a simplistic set of rules. Emphasizing the superiority of DRL over other AOS methods for this purpose. Besides this increased

performance, the entire proposed framework shows to be generalizable as well. This can be seen in the framework retaining its performance when it was taken from its training environment to the actual WDCP environment.

Future researchers, and potentially practitioners, is recommended to extend the research on different aspects. First, additional research into model volatility is required for optimal implementation. To ensure convergence, still a lot of evaluations are needed, partly due to the volatility in performance over several runs. Potential benefits with regards to volatility can be attained through optimization of the performance of the framework to be executed on either the optimization algorithm or the DRL agent or both. An example of potential improvements of the optimization algorithm include investigation into different population sizes. Secondly, additional research into model biases is required. The results showed a small inclination towards the second objective value, potentially as a result to the set upper and lower bounds. Mitigating the effect, or even getting rid of, the upper and lower bounds would lower the bias of the framework and thus improve results. Finally additional research into the found Pareto solutions is recommended for practitioners. This research showed a purely academical approach, focusing on model quality while disregarding the actual meaning of the solutions. Before implementation of the framework it is perceived valuable to look into the proposed solutions and potentially adjust the hyperparameters of the model to shift it to focus on what is desired.

Preface

This report is written as the partial fulfillment of the requirements to obtain the degree Master of Science in Operations Management and Logistics at Eindhoven University of Technology. It describes my research conducted over the past seven months on simultaneous optimization of warehouse layout design and control policies. Execution of the research is done in collaboration with Nobleo Manufacturing.

Through writing this I would like to express my deepest appreciation towards the people that guided and supported me throughout this master thesis project. In particular, I want to thank Yingqian Zhang. Despite it being an extraordinary period to be enrolled in a master study, due to the pandemic, I knew I could reach out to you and receive guidance if needed. Besides guidance, you let me free to shape my studies in the direction I desired them to go in, which I am deeply grateful for. I would also like to thank Laurens Bliek for the concise guidance and help with finding relevant literature. In addition I would like to thank Berend Steenhuisen, my company supervisor. Besides being very enjoyable to work with, our weekly meetings ensured that our vision about the direction of this research stayed aligned and the operational knowledge you shared helped me to solve all issues faced throughout the research.

I want to give a special thank you to Robbert Reijnen, which personally felt like an additional supervisor from time to time. Despite his own busy schedule he always made time to take my call and discuss potential solutions to my problems. Finally, I want to show my appreciation to my close friends, family and girlfriend for helping me through this often lonely process of working from home. Time spend with you allowed me to charge my personal battery, which enabled me to stay motivated and maintain a high level of performance when needed.

Remco Coppens
Eindhoven, March 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This paper will elaborate on the research conducted at Nobleo Manufacturing, hereinafter referred to as Nobleo. Nobleo was founded in Eindhoven in 2012 with the purpose of delivering specialized service on industrialization issues. Demand for their service rose quickly, mainly in the Automotive and High Tech Systems sectors. As a consultancy company, Nobleo focuses on industrialization and manufacturing processes. By helping to produce smarter, faster and better, the clients of Nobleo become and stay competitive in their market(s). By continuously delivering upon promised results, their clientele grew nationally and internationally [Nobleo (2021)].

All employees of Nobleo are member of one or more competence teams. A competence team is guided by a program lead, who is responsible for competence development within their team. Collaboration between these competence teams enables Nobleo to support companies in every phase of their industrialization transition. The thesis will be executed from within the Manufacturing Intelligence competence team, supervised by program lead B. Steenhuisen.

1.1 Problem context

The thesis project concerns the storage of plastic in a warehouse of a large chemical manufacturer. The warehouse has 24 truck docks for which there are 28 consolidation areas available. Through these areas loading and unloading of trucks occurs. In these areas all products are collected, after which inbound products are moved to storage locations in the warehouse and outbound products are moved to an outbound truck for transportation. Daily in- and outbound at the warehouse consists of on average 95 trucks. With 24 docks available, the 28 consolidation areas need to be turned-over 3-4 times a day. Currently insufficient storage space is available to store all of the incoming products in the warehouse. The main reason for this is inefficient storage utilization. Nobleo is asked to redesign the warehouse to increase storage space utilization and overall efficiency of warehouse operations.

The warehouse has three different types of storage, namely block, back-to-back (B2B) and shuttle storage. A visualization of these types of storage can be found in Figure 1.1, 1.2 and 1.3. In the current situation mostly block storage is used. This storage is characterized by having a lane only contain a single type of product, which can be stacked on top and in front of one another. For high volume products this method is most space efficient, as no

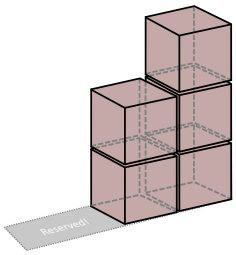


Figure 1.1: Block storage

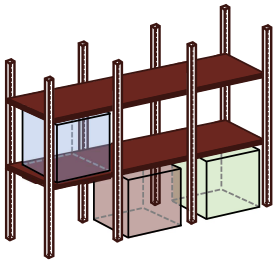


Figure 1.2: B2B storage

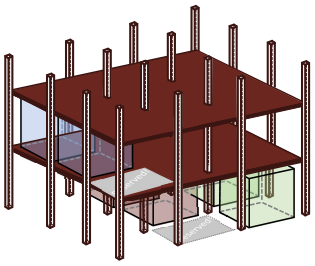


Figure 1.3: Shuttle storage

driving lanes have to be present in between the products. The complete opposite is B2B storage. In B2B storage a driving lane is present between every product, where racks allow for placement of different products on top of one another. The benefit of this method is that it ensures availability of storage locations, as it does not reserve storage spaces for identical products as block storage does. However, due to the need for driving lanes between every product, the downside is the loss of efficiency in storage space utilization. The third storage type, shuttle storage, combines both efficient storage space utilization with storage space availability. Using a mole, i.e. a small Automated Guided Vehicle (AGV), products can be placed in front of one another on the same level of a metallic shuttle frame. Leaving the possibility of storing different types of products on top of each other. Although promising, implementing shuttle storage involves incurring high investment and operating costs. The advantages and disadvantages of the three types of storage are summarized in Table 1.1.

	Advantages	Disadvantages
Block storage	High storage space utilization.	Low storage space availability.
Back-to-back storage	High storage space availability.	Low storage space utilization.
Shuttle storage	High storage space utilization and availability.	High investment/operating cost.

Table 1.1: Advantages and disadvantages of different storage functional areas

Besides storage type, also storage locations also differ in their storage dimensions. These dimensions vary in width and height. For block, back-to-back (B2B) and shuttle storage there are two, nine and eight different width-height combinations respectively. Products can be placed in storage locations equal or larger in size than the product dimensions. However, placing smaller products in larger storage areas results in inefficiency of storage space utilization. Decreasing the dimensions of storage locations will enable more products to be stored, but limits the number of available storage locations for larger products.

Internal transportation of products is currently executed using three different types of resources, namely: forklifts, reach trucks, and moles. Only forklifts are able to handle loading and unloading of out- and inbound trucks respectively. Transportation of products towards or from block storage can be executed by forklifts as well as reach trucks. However, transportation towards and from B2B and shuttle storage can only be executed by reach trucks. For shuttle storage a reach truck needs to have a mole at its disposal. Due to the high purchase cost of a mole, the number of moles available is limited and a specific mole is assigned to a single Reach Truck. The capabilities of different resources is summarized in Table 1.2.

	(De)loading trucks	Transportation to/from Block storage	Transportation to/from Back-to-back storage	Transportation to/from Shuttle storage
Forklift	X	X		
Reach truck		X	X	
Reach truck + Mole		X	X	X

Table 1.2: Capabilities of different resource types

Currently internal logistic of the warehouse is dictated by two algorithms, namely the Product Placement Algorithm (PPA) and the Truck Docking Algorithm (TDA). The PPA consists of a set of rules based upon a set of values. Through these rules it decides for every product if it should be stored in either block storage, back-to-back storage or shuttle storage. Placement in either one of these storage types is based on figures like historic outbound quantities, current inventory and the number of pallets that can be placed on top of one another. The TDA, on the other hand, decides for incoming trucks to which dock they get assigned. This algorithm concerns a so-called greedy algorithm. This concerns minimization of total travel distance, which for inbound trucks implies the travel distance to place all products in the warehouse and for outbound trucks to retrieve all products from the warehouse. The PPA en TDA dictate the internal logistics of the warehouse considered in this research. To further elaborate upon the internal processes, a flowchart visualizing the internal logistics is shown in Appendix A. This flowchart follows the process from a product perspective, showing all processes between product arrival through an inbound truck and product departure in an outbound truck.

However, the process has two important factors to note. Firstly, in most cases the outbound truck demand is known two hours before arrival of the actual truck. This enables internal logistics to start retrieval of the ordered products in the assigned consolidation area. An exception to this rule is made for rush orders which happen sporadically. For these trucks the demand is not known in advance and is revealed upon truck arrival. Secondly, not all requests for products concerns full-pallet orders. Occasionally a truck order contains a less-than full pallet quantity. When this happens a full pallet is picked and broken into a requested and remaining quantity. The requested quantity is transported to the consolidation area and the remaining quantity, called a broken pallet, is returned to inventory. Broken pallets are always stored in Back-to-Back storage, regardless of the PPA.

This research intends to create a procedure of simultaneously optimizing upon warehouse layout design and the underlying control policy. The decision variables of the optimization problem concern:

- The order of the rules comprising the PPA.
- The parametric values within the rules of the PPA.
- The amount of resources per resource type.
- The dimensions of the storage locations, i.e. the width in block storage and the width and height of the shelves in Back-to-Back and Shuttle storage.

Adaptation of these decision variables is supposed to lead to well-performing warehouse configuration. The performance of a warehouse configuration will be evaluated based on three metrics, making it multi-objective optimization. These three metrics will be discussed more thoroughly in section 4.1.3, but they concern:

- The tardiness of outbound trucks
- The resource cost
- The number of unplaceable products

An accelerated replication of the detailed simulation used within Nobleo will be the basis upon which an iterative optimization algorithm will be developed. Due to the high dimensionality and expensive computation of simulation results, the number of needed evaluations need to be kept to a minimum. To achieve this, the optimization algorithm must include some form of intelligent determination of potential solutions to be evaluated. It is desired to converge to a set of optimal warehouse configurations while minimizing the number of performance evaluations needed.

1.2 Research questions

The research problem is, throughout literature. described as the Warehouse Design and Control Problem (WDCP). The WDCP consists of a collection of underlying sub-problems, which often have solutions using Evolutionary Algorithms. To minimize the computation time of the optimization model, a process called Adaptive Operator Selection (AOS) will be applied. Literature shows a salient method to implement AOS concerns Deep Reinforcement Learning. These findings led to the following main research questions, which is central to this master thesis:

How can a Multi-objective Evolutionary Algorithm and Deep Reinforcement Learning be combined to simultaneously optimize warehouse layout design and control policies using a low computational budget ?

The main research question is subdivided into eight sub-questions, which together will result in an answer to the main question. These sub-questions are:

1. How does Nobleo currently find the optimal process configuration?
2. What are the current best practices to solving the Warehouse Design and Control Problem?
3. What are the state-of-art multi-objective bio-inspired metaheuristic algorithms, and what are their advantages and disadvantages?
4. How can a metaheuristic be applied to the Warehouse Design and Control Problem?
5. What are existing methods for Adaptive Operator Selection, and what are their advantages and disadvantages?
6. How can Deep Reinforcement Learning be applied as the Adaptive Operator Selection method?

7. What is the performance of the proposed optimization model?

8. How can the proposed model be made generic to enable application to different multi-objective optimization problems?

These questions can be categorized over different parts of the research. Question 2, 3 and 5 all concern questions regarding literature. Question 1 focuses on the current process, where Question 4 and 6 focus on the development of a framework to improve the current process. Finally, question 7 and 8 focus on the obtained results and implementation of the proposed solution framework. This division is visualized in Figure 1.4.

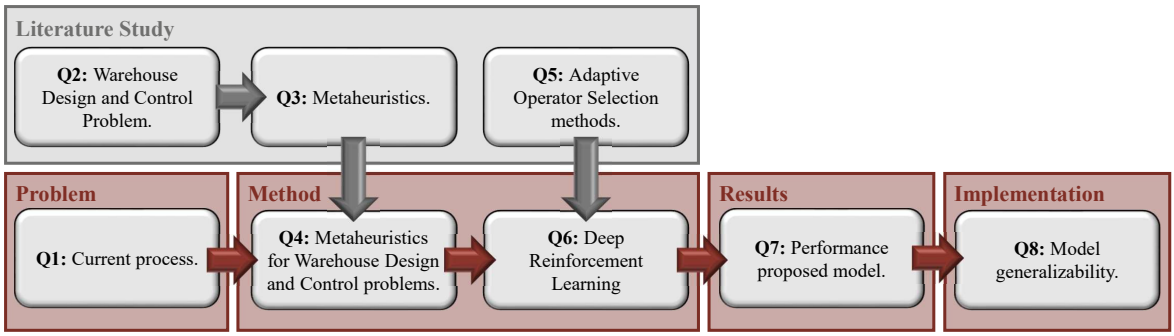


Figure 1.4: Subdivision Research Questions

1.2.1 Outline

The rest of this document is structured as follows. Chapter 3 will discuss a literature review. This review summarizes previous work and highlights potential solution methods to be used. Execution of this review intends to answer sub-questions 2, 3 an 5. The gathered knowledge from this review will aid the development of the proposed solution method, which is discussed in chapter 4. The design of the solution methods will answer sub-questions 4 and 6.

The eventual implementation of the solution method will be taken through a series of experiments, explained in chapter 5. These experiments revolve around performance comparisons, performance optimization and extensive analysis of the inner workings of the methods. The results of these experiments will be shown and discussed in chapter 6, which will answer sub-question 7 and 8. The obtained results and answers will be summarized in chapter 7, resulting in an answer to the main research question.

1.2.2 Contributions

This research increases both the knowledge and applicability of a solution method for simultaneous optimization of warehouse layout design and control policies. Through doing this, it provides insight into the following subjects:

1. This study concerns simultaneous optimization of warehouse layout design, resource allocation and product allocation. The amount and types of problems comprising this optimization problem, known as the Warehouse Design and Control Problem (WDCP),

is not found throughout literature. The closest related paper only solves the product allocation problem and the functional area size determination problem.

2. The proposed solution method allows for optimization of the set sub-problems without imposing any predefined preference on the objective values. Implying the method to be multi-objective in a way that it returns a set of Pareto-dominated solutions, instead of returning a single value using an apriori method as seen in literature. This set is presented to the decision maker, in this case Nobleo, who can choose which solutions to pursue.

3. The proposed solution method shows a novel implementation of combining the NSGA-III algorithm with Deep Q-Networks (DQN). Applying DQN as Adaptive Operator Selection (AOS) for the mutation operators of the NSGA-III, convergence performance with regard to both speed and quality is increased.

4. The proposed solution method is generic. All calculations exclude problem specific information, enabling it to be applied to a wide range of optimization problems. The method will be able to solve all problems that allow for a solution to be encoded in a single vector of values. Extending this research this could imply different compositions of the WDCP, consisting of any number of sub-problems. Because of this, it shows increased utility for practitioners.

5. The developed optimization framework is made compatible with the simulation software used within Nobleo. Besides, the framework is made to be fully operated through an Excel sheet. Within this Excel sheet the employees of Nobleo can change fundamental features of the optimization framework. These features concern the amount and type of decision variables, the amount and type of objective values, fundamental NSGA-III hyperparameters and enabling/disabling the learned Deep Reinforcement Learning agent.

To the knowledge of the author, through these contributions this research distinguishes itself from other research found in literature. It also intends to incentivize researchers to extend investigation into the simultaneous optimization of different warehousing principles.

Chapter 2

Problem formulation

This chapter will more thoroughly discuss the Warehouse Design and Control Problem (WDCP) central to this research. First the algorithms dictating internal logistics are described, which concern the Product Placement Algorithm (PPA) and the Truck Docking Algorithm (TDA). Following this, the decision variables will be discussed and defined mathematically. These mathematical notations will be used to create a formal definition of the problem.

2.1 Operational algorithms

As mentioned in the problem context, discussed in section 1.1, internal logistics is dictated by two algorithms. The first algorithm concerns the Product Placement Algorithm (PPA), which determines in which of the three storage types a given product will be stored. Psuedo-code, describing the PPA, can be found in Algorithm 1. The PPA consists of four rules, of which three actual rules and one fallback option in line 7, 9, 11 and 13 of the psuedo code respectively. These rules are handled in the order in which they are presented. Decisions made using these rules depend on five parameter values $(\alpha, \beta, \gamma, \delta, \upsilon)$. The order of the four rules and the five parameter values are decision variables. The PPA is currently already used within the simulation of Nobleo, in which the parameter values take the following values: $\alpha = 360$, $\beta = 12$, $\gamma = 3$, $\delta = 1.93$, $\upsilon = 2$. Also the order in which the rules are shown is the current rule order used within the simulation of Nobleo.

Algorithm 1: Product Placement Algorithm (PPA)

INPUT:
$p \leftarrow$ product to be placed;

OUTPUT:
$\mathcal{S}$: The storage type in which the product will be placed

1 **if** $p_{Yearly\_Outbound\_Shipments} < \alpha$ ***and*** $p_{Inventory} < \beta$ **then**
2 | Place product in Back-to-Back storage
3 **else if** $p_{Stack\_Level\_1} \geq \gamma$ **then**
4 | Place product in Block storage
5 **else if** $p_{Pallet\_Height} > \delta$ ***and*** $p_{Stack\_Level\_2} \geq \upsilon$ **then**
6 | Place product in Block storage
7 **else**
8 | Place product in Shuttle storage

Next to the PPA another algorithm is used within the internal logistics, namely the Truck Docking Algorithm (TDA). As discussed in section 1.1, the TDA concerns a greedy algorithm. This implies that a truck is docked in a way that minimizes the travel distance within the warehouse. For inbound trucks this is based on the products that are within the truck and their prospected storage locations, where for outbound trucks this is based on the requested products and their current locations in the warehouse. Pseudo-code describing the TDA can be found in Algorithm 2. The TDA does not contain any adjustable parameters, and is thus solemnly used to decide where to dock trucks in the simulation. The first part of the TDA (line 1 to 7) calculates the distance of all products in the truck or requested by the truck $[o_k]$ to all available docks. Thereafter, in the second part (line 8 to 16), the docks are sorted based on their total distance to travel. Looping through the docks, starting with the one resulting in the shortest travel distance, the first available dock with a consolidation lane available is used to dock the truck.

Algorithm 2: Truck Docking Algorithm (TDA)

INPUT:
$o_k \leftarrow$ list of items within or requested by the truck;
$\mathcal{D} \leftarrow$ the total amount of docks;

OUTPUT:
-

```
1  dist_dict ← {};
2  for dock ← 0 to 𝒟 do
3  |   total_dist ← 0;
4  |   for item ← 0 to o_k do
5  |   |   total_dist ← total_dist + 2 * get_dist(dock, item)
   |   end
6  |   dist_dict[dock] ← total_dist;
   end

7  sorted_dist_dict ← sort(dist_dict)
8  i ← 0;
9  while i < 𝒟 do
10 |   if sorted_dist_dict[d] = unoccupied and consolidation (lane) = available then
11 |   |   Assign truck to dock
12 |   |   Reserve consolidation area
13 |   |   End Algorithm
14 |   else
15 |   |   i ← i + 1;
   end
```

2.2 Decision variables

Briefly mentioned in section 1.1, there are four categories of decision variables. The first two categories concern the rule order and the parameter values comprising the PPA [Algorithm 1]. As there are three rules and a fallback option, the decision variables concern a list of four ordinal values ($\mathcal{O}$). Having four ordinal values allows for the fallback option to be shifted towards a higher position, eliminating all rules that follow this option. The parameter values,

on the other hand, consist of four integer values $(\alpha, \beta, \gamma, \upsilon)$ and one categorical value (δ).

The third category of decision variables concerns the amount of resources per resource type. There are three types of resources, namely forklifts $(\mathcal{Z}_1)$, reach trucks $(\mathcal{Z}_2)$ and reach trucks + mol $(\mathcal{Z}_3)$, i.e. a small Automated Guided Vehicle (AGV). The amount of resources per resource type can formally be described as $\mathcal{Z}_i,\ for\ i = \{1, 2, 3\}$.

The fourth category of decision variables concerns the dimensions of the storage locations. These dimensions concern combination of values taken from two sets containing different widths $(\mathcal{W})$ and heights $(\mathcal{H})$. The definition of this decision variable is split up over the five different halls $(\mathcal{X})$, which comprise the entire warehouse. Every hall contains different types of storage, where some halls only contain one type of storage while other halls can contain a combination of multiple storage types. The decision variables concern a list of percentages, describing per hall and storage type the amount of storage space being allocated to a given dimension. A formal definition of this decision variable is trivial, but an abstract representation of the integer amount of storage locations per storage dimension can be described as $d^x_{w,h} \forall x \in \mathcal{X}, \forall w \in \mathcal{W}, \forall h \in \mathcal{H}$. The decision variables described are summarized in Table 2.1.

Notation	Meaning	Datatype	Values
$\mathcal{O}$	Ordering of PPA rules	Ordinal	[1, 2, 3, 4]
α	(PPA) Bound on amount of Yearly Outbound Shipments	Integer	LB = 100 UB = 500
β	(PPA) Bound on Inventory	Integer	LB = 2 UB = 20
γ	(PPA) Bound on Stack Level 1	Integer	LB = 1 UB = 6
δ	(PPA) Bound on Pallet Height	Categorical	[1.13, 1.66, 1.93, 2.30]
υ	(PPA) Bound on Stack Level 2	Integer	LB = 1 UB = 6
$\mathcal{Z}$	Number of resources per type	Integer	LB = 1 UB = 20
$\mathcal{W}$	Different storage dimension widths	Categorical	[1.20, 1.40, 1.60]
$\mathcal{H}$	Different storage dimension heights	Categorical	[1.13, 1.66, 1.93, 2.30]
$\mathcal{X}$	Different halls comprising the warehouse	Categorical	[1, 2, 3, 4, 5]

Table 2.1: Formal definition of the decision variables

2.3 Problem definition

Based on the background information of the problem introduced in section 1.1, a formal problem definition can be defined. Consider a time horizon $\mathcal{T} = \{t_1, t_2, ..., t_T\}$ in which a set of inbound trucks $(\mathcal{K}_{inb})$ and outbound trucks $(\mathcal{K}_{outb})$ will arrive. During this time horizon a given number of resources per resource type is available. One of the objective values for optimization concerns the resource cost, which is denoted as $\sum_{i=1,2,3} c_i \cdot \mathcal{R}_i$, where "$c$" concerns an aggregate of both investment and operating costs of resources.

Upon arrival of an inbound truck $(\mathcal{K}_{inb})$, the TDA discussed in Algorithm 2 decides to which dock the truck is assigned based on all of the products its carrying (o_k). After docking,

all products are transported to the storage locations in the warehouse using the available resources ($\mathcal{Z}_i \forall i \in [1,2,3]$). The decision to which storage type a given product is transported is decided by the PPA (Algorithm 1). The dynamics of this algorithm depends on the decision variables concerning the rule order ($\mathcal{O}$) and the values ($\alpha, \beta, \gamma, \delta, \upsilon$). The decision in which storage type ($\mathcal{S}$) a given product (p) will be stored can be formally described as: $\mathcal{S} = PPA(p)$. After deciding in which storage type ($\mathcal{S}$) the product (p) will be stored, the availability of storage locations with suitable dimensions has to be checked. The number of storage locations of different dimensions depends on $d^x_{w,h} \forall x \mathcal{X}, \forall w \in \mathcal{W}, \forall h \in \mathcal{H}$. As products can only be stored in storage areas equal or larger in either one or both of the dimensions, the possible storage locations of a given storage type for a given product can be defined as $d_p^{\mathcal{S},x} \forall w \in \left[p_w, ..., \mathcal{W}^{\mathcal{S}}\right], \forall h \in \left[p_h, ..., \mathcal{H}^{\mathcal{S}}\right], \forall x \in \left[1, ..., \mathcal{X}^{\mathcal{S}}\right]$. To allow determination of the number of unplaceable products, being one of the objective values for optimization, an indicator value ($u_p^{\mathcal{S}}$) needs to be created. This variable gets value 1 if $d_p^{\mathcal{S},x} = 0 \forall x \in \mathcal{X}$ and 0 otherwise. The number of unplaceable product can then be formally defined as $\sum_{k \in \mathcal{K}_{inb}} \sum_{p \in o_k} u_p^{\mathcal{S}}$, where $\mathcal{S}$ is fixed as decided by the PPA.

Upon arrival of an outbound truck ($\mathcal{K}_{outb}$), the TDA (Algorithm 2) is used to decide to which dock the truck should be assigned. All products requested by the truck (o_k) need to be retrieved from the warehouse, transported using the available resources ($\mathcal{Z}_i \forall i \in [1,2,3]$). To evaluate the third and final objective value, tardiness of outbound trucks, the difference in truck arrival time (a_k) and its departure time (d_k) is needed. However, the departure time value dependents on the dynamics of the warehouse, which is stochastic. The truck handling time (y_k) is then formally expressed as: $y_k = d_k - a_k$. To calculate the tardiness of outbound trucks the truck handling time is multiplied with a constant (f_k). The value of this constant is determined based on an outbound truck performance classification metric supplied by Nobleo. It results in the constant (f_k) having a value of 0 if $y_k \leq 30$, 0.5 if $30 < y_k \leq 120$ and 1.0 if $120 < y_k$. The objective value, tardiness of outbound trucks, can then formally be described as $\sum_{k \in \mathcal{K}_{outb}} f_k \cdot y_k$.

To summarize, the objective values of the optimization problem is to minimize the resource cost $\sum_{i=1,2,3} c_i \cdot \mathcal{R}_i$, minimize the number of unplaceable products $\sum_{k \in \mathcal{K}_{inb}} \sum_{p \in o_k} u_p^{\mathcal{S}}$ and to minimize the tardiness of outbound trucks $\sum_{k \in \mathcal{K}_{outb}} f_k \cdot y_k$. The decision variables that can be adjusted during optimization are summarized in Table 2.1.

The above mentioned formal definitions show a simplification of the actual process, for which construction of such definitions is intractable. The definitions are solemnly used to increase problem understanding and will not be used throughout the remainder of this research.

Chapter 3

Literature review

The research to be conducted will be based upon a review of current-day literature. The reason this review is executed is two-fold. On one hand it will elaborate on the subject discussed and potential solution methods used. On the other hand, it is used to show the position of this research in literature. These subjects will be addressed throughout this review and will be discussed in the following sections.

3.1 Warehouse design and control problem

The problem discussed in chapter 1 can be described as a Warehouse Design and Control Problem (WDCP). The WDCP concerns simultaneous optimization of the warehouse lay-out design and the control policies, which dictate internal logistics. Following the paper of Rouwenhorst et al. (2000), the problem can be subdivided into three decision levels, namely: Strategic, Tactical and Operational. The strategic decisions concern long term decisions with a horizon of about five years, focusing on design of process flow and selection of (technical) systems. Tactical decisions concerns a shorter horizon of around two years. These decisions include the dimensions of the storage system(s), the lay-out design, the selection of equipment and the structural design of organizational policies. Finally the operational decisions focus on a short term of around one year, mainly focusing on fine-tuning of the designed organizational policies.

Decisions on different levels are interrelated hierarchically. Meaning that a decision on a higher decision level puts constraints and additional requirements on decisions on lower levels [Rouwenhorst et al. (2000)]. Besides the hierarchical relation, also various problems at the strategic level appear to be highly interrelated as well. To a lesser extent this holds for the tactical level, whereas decisions made at the operational level often can be considered independently [Rouwenhorst et al. (2000)]. A solution taking multiple decision levels into account would be able to utilize these (hierarchical) interrelations. Research into this field could lead to general design procedures and global optimization models [De Koster et al. (2007)].

Current-day literature shows little depth of academic research into the Warehouse Design and Control Problem (WDCP). For this reason the review of literature is conducted on the three sub-problems which constitute the collective problem of this research. These problems concern the warehouse layout design problem, resource allocation problem, and the

product allocation problem. The layout design and resource allocation problem both concern tactical decisions, whereas the product allocation problem concerns a decision on an operational level. Simultaneous optimization will enable utilization of the interrelations between the layout design and resource allocation problem and the hierarchical interrelation to the product allocation problem, as discussed in Rouwenhorst et al. (2000). Allowing for improved optimization results as opposed to optimizing all individual problems separately. Seprate optimizations would results in a number of independent local optima, which combined will lead to a sub-optimal overall solution.

3.1.1 Warehouse layout design problem

Solutions to the warehouse lay-out design problem show both exact and heuristic approaches. Earlier works by Rosenblatt and Roll (1984, 1988) and later work by Roodbergen and Vis (2006) show exact methods, where the former constructs a procedure of consecutive mathematical equations and the later solves a nonlinear programming problem. The focus of these studies is mainly on the relationship between the warehouse design and the total warehouse construction and/or material handling costs. The cost function used to measure the utility of the warehouse assumes that the warehouse is a continuum, which is an unrealistic assumption when looking at real-world applications. Work by Pandit and Palekar (1993) does not make this assumption and optimizes on warehouse response time using an iterative search procedure. Dropping this assumption makes exact methods intractable, in particular when problem size increases. Because of this, heuristic methods have to be applied in real-world problems, despite these solution methods having a significantly higher computation time compared to its exact counterparts.

3.1.2 Resource allocation problem

The solution methods used to solve the Resource Allocation Problem (RAP) consist of exact methods, heuristics and several metaheuristics. A lot of exact methods focus on an implementation of the Hungarian method [Younas et al. (2011); Xian-Ying (2012)], a combinatorial optimization algorithm especially designed for assignment problems. Despite the attained time efficiency of polynomial time [Kuhn (1955)], these methods do not extend further than single-objective, binary assignment problems having a static amount of resources and tasks with a predefined duration. The same is true for (Integer) Linear Programming [Daskalaki et al. (2004); Azimi et al. (2013)] and the Branch & Bound algorithm [Bretthauer and Shetty (1995); Miralles et al. (2008); Vila and Pereira (2014)]. The use of heuristic methods shows to be able to handle more complex problems [Bennour et al. (2005)], extending to multi-objective RAP. Compared to several metaheuristics, a heuristic shows lower computational cost but is unable to outperform the metaheuristics on solution quality upon convergence. For real-world implementations only metaheuristics show to be able to cope with the incommensurability of multiple objectives, although showing high computational cost [Mutlu et al. (2013); Fan et al. (2013); Thepphakorn et al. (2014); Odeniyi et al. (2015)].

3.1.3 Product allocation problem

The final solution methods analyzed are used to solve the Product Allocation Problem (PAP). The exact methods for the PAP are able to include necessary complexity to attain a realistic optimization process [Heragu et al. (2005); Guerriero et al. (2013)]. These methods consists

of an often simplistic objective function, subject to an extensive and detailed amount of constraints. The set of equations results in fast optimization for smaller instances, but quickly gets intractable as the size of the problem increases. Looking at real-world cases the PAP is often a high dimensional problem, which is shown to be more efficiently solved using a meta heuristic [Heragu et al. (2005); Sammons Jr et al. (2008); Li et al. (2009); Guerriero et al. (2013)].

3.1.4 Simultaneous Optimization

The papers by Heragu et al. (2005) and Roodbergen et al. (2015) are the only paper found looking into joint decision making, solving the product allocation problem (PAP) and warehouse layout related problems simultaneously. The environment used in the paper by Heragu et al. (2005) transformed stochastic variables into deterministic variables by imposing certain assumptions. Through a combination of a mathematical model and a heuristic, both the handling and storage cost were minimized. The different (meta)heuristics tested concern the Branch-and-Bound (B&B) algorithm, a custom-made heuristic and the Simulated Annealing (SA) algorithm, of which the latter is a (bio-inspired) metaheuristic. It is found that the B&B algorithm suffers from memory problems, where the SA method shows superior results for large problem instances. The paper by Roodbergen et al. (2015) concerns single objective optimization, focusing on average travel distance per order. Through defining problem specific mathematical rules, optimal settings are obtained through a proposed heuristic method.

For the case at Nobleo the three sub-problems comprising the WDCP are all high dimensional and need to be optimized simultaneously. Adding additional complexity due to internal and hierarchical dependencies between the problems. Concluding from the solution methods to the separate problems and the increased complexity of simultaneous optimization, a metaheuristic would be well suited. For this research bio-inspired metaheuristics will be further elaborated. The main reason for this being the large application domain of these methods found throughout literature. The disadvantage of using (bio-inspired) metaheuristics is high computational cost. In practice there is often a desire to test different optimization processes, allowing for different ranges of optimization. As this is the case, the high computation time should be mitigated as much as possible.

3.2 Bio-inspired metaheuristics

Real-world optimization problems are often challenging to solve, and many applications have to deal with NP-hard problems [Fister Jr et al. (2013)]. One way to solve these problems is through so-called Bio-inspired metaheuristics. Metaheuristics concern problem-independent techniques, of which Bio-inspired metaheuristics concern a category of models inspired by biological phenomena as seen in nature. The functionality of such heuristics resides on two different spaces, a decision space and an objective space. The decision space contains the values that describe a solution to be tested. The objective space contains the performance values that these solutions can attain.

To work on both of these spaces a metaheuristics consists of two parts, namely the optimization algorithm and an evaluation function. Upon initialization the algorithm proposes a set amount of randomly initialized solutions in the decision space, a so-called population

of individuals. The population will be taken through the evaluation function, resulting in a performance evaluation for all individuals in objective space. Based upon the composition and performance of the individuals comprising the current population the optimization algorithm will propose a new population, called the offspring. The offspring is proposed in decision space again, which completes a cycle between the two spaces. One cycle is called a generation, which is repeated until a predefined termination criterion is reached. Examples of termination criteria are execution of a desired amount of generations, no improvements for a predefined set of generations or a desired performances attained by the population. This process, together with both spaces, is visualized in Figure 3.1.

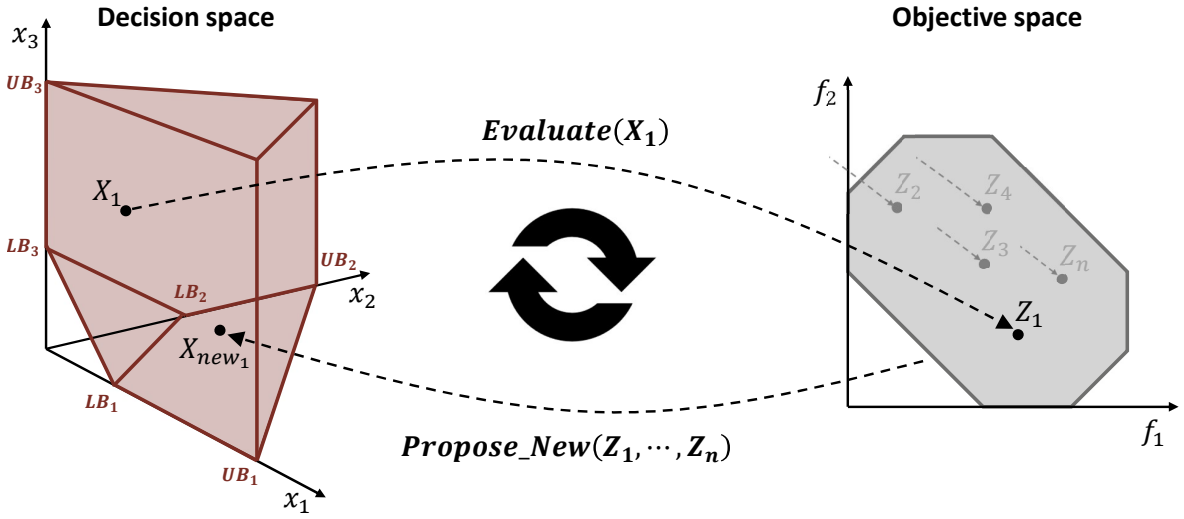


Figure 3.1: Decision and objective space (adjusted, original retrieved from: Deb (2014))

A key issue when solving optimization problems with complex fitness landscapes is to keep an appropriate balance between exploration and exploitation [LaTorre et al. (2020)]. Exploration concerns remaining or extending the diversity of the population. This is necessary to minimize the risk of converging to local optima and thus not discovering a potential global optimal solution. Exploitation, on the other hand, concerns searching close to the well-performing individuals found so far. Through combining the composition of two of these individuals, called parents, it is aspired to create new individuals that outperform their parents from which they originated. Too much emphasis on exploration and the algorithm becomes unstable and will never converge, while too much emphasis on exploitation will result in the algorithm converging to sub-optimal, local optima.

3.2.1 Multi-objective optimization

Multi-Objective Optimization (MOO) can be described as the process of optimizing systematically and simultaneously a collection of, often conflicting, objective functions [Marler and Arora (2004)]. Based on their way of handling the trade-off between the different objective values the MOO methods can be subdivided in four categories [Cui et al. (2017)], as visualized in Figure 3.2.

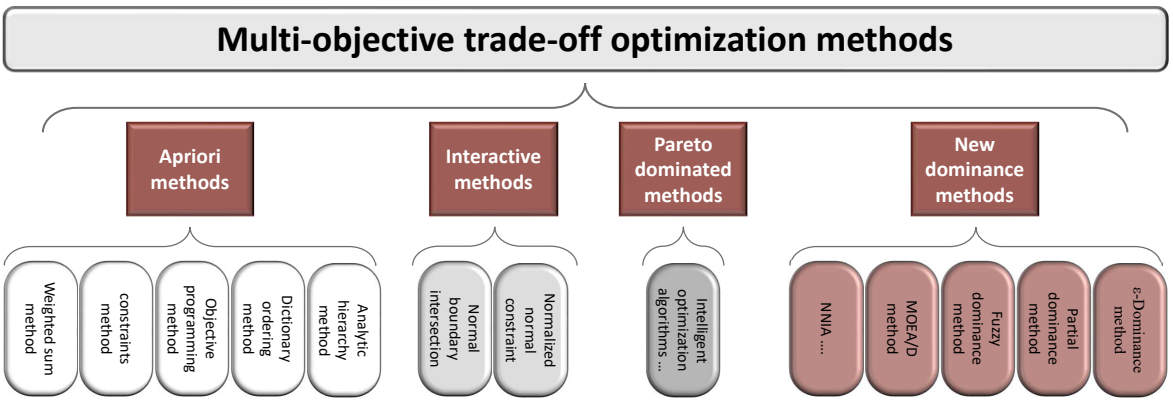


Figure 3.2: Methods multi-objective trade-off (retrieved from: Cui et al. (2017))

- **Apriori Methods** rely on predefined settings transforming the multiple objective functions into a single value. These settings concern weights for the Weighted Sum Method, constraints for the ε-Constraint Method or desired objective values for the Objective Programming Method. Setting these values requires a thorough understanding of the process, which is not fully present at Nobleo as it concerns a process of their client. Besides, these methods are often applied in fairly specific processes, for example in Ethylene cracking furnaces [Xia et al. (2013); Geng et al. (2012)] and hybrid energy systems [Ju et al. (2016)]. As the thesis project concerns a widespread, non-linear, non-convex optimization problem and the project is guided by a consultancy firm instead of the actual company, these specific values are difficult to attain. For this reason Apriori methods will not be used for this research.

- **Interactive methods** incorporate the preferences of the decision makers for each objective during the optimization process [Cui et al. (2017)]. These methods find optimal solutions through implementation of a so called Achievement Scalarization Function (ASF), based on the preferences of the decision maker. The intuitive methods tackle the multi-objective optimization problem from a geometrically intuitive viewpoint. The interactive methods also require extensive knowledge of the process. As with the Apriori methods, the position of this research is too far away from the process to facilitate this. For this reason it is decided to not use the interactive methods to trade-off the multiple objective values.

- **Pareto dominated methods** were introduced by Schaffer (1985), proposing an algorithm called the Non-dominated Sorting Genetic Algorithm (NSGA). These algorithms are able to solve non-convex optimization problems without coupling objectives, which implied maintaining individual features for all objectives [Cui et al. (2017)]. This characteristic sets it apart from the apriori and interactive methods. Although showing superior performance in convergence, diversity, robustness and flexibility, the computation process is often time-consuming and relatively low-efficient [Shukla and Deb (2007); Lu et al. (2012)]. Due to the high performance and the ability to solve non-convex optimization problems, it is decided to further investigate the applicability of these methods regarding the thesis project.

- **New dominance methods** are the newest algorithms, designed to solve the convergence and distribution problems of an optimization algorithm simultaneously [Cui et al. (2017)]. The proposed ε-dominance [Deb et al. (2002)] and later extended Pareto-adaptive ε-dominance [Hernández-Díaz et al. (2007)] mechanisms are both based on the idea of hyper grids related through the different optimization spaces. The performance of both of these methods show promising, except they experience drawbacks if the number of feasible objective vectors is small [Horoba and Neumann (2008)]. The New Dominance methods will, despite their performance, not be used in this thesis project. Due to the high level of mathematical complexity, these methods lack explainability and applicability towards the process at Nobleo.

3.2.2 Pareto dominated algorithms

Due their ability to solve non-convex optimization problems without coupling objectives, Pareto dominated algorithms are perceived valuable methods regarding the WDCP central to this research. As a result of maintaining individual features for all objectives, comparing individual solutions becomes arbitrary. For this reason the Pareto dominated algorithms use a so-called Pareto optimal set, which concerns an approximation of the set of globally optimal solutions. Optimality in this sense is decided through the concept of domination. A solution (x_1) dominates another solution (x_2) if and only if:

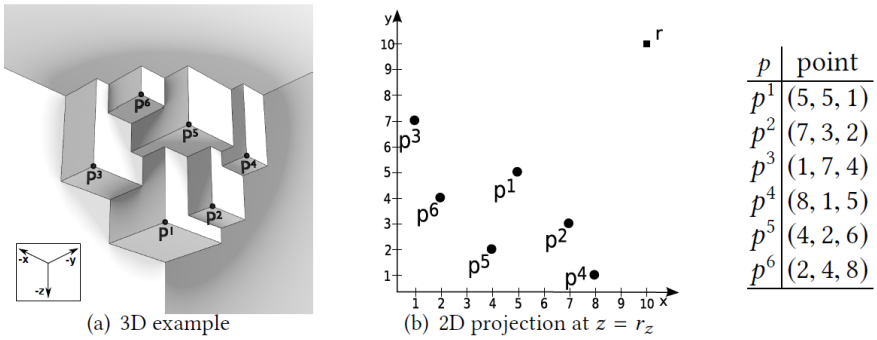
- x_1 is no worse than x_2 for all objectives.
- x_1 is strictly better than x_2 in at least one objective.

The list of solutions that are non-dominated form the Pareto optimal set, often called the Pareto front, which can be seen as the set of optimal solutions found by the algorithm.

Evaluation of the performance of such algorithms is essentially an evaluation of the properties of this set. The set obtained through the algorithm is an approximation of the actual Pareto front, as this set of solutions is unknown due to the unknown nature of the objective space, visualized in Figure 3.1. Besides an evaluation of this approximated set, also the computational resources needed to generate this set will be taken into account [Si et al. (2019)]. The evaluation of performance focuses on the objective space, rather than the decision space, as the intended purpose of multi-objective optimization algorithms is to observe the balance between conflicting objectives. Metrics used throughout literature concern the size, diversity and the convergence rate of the Pareto-optimal set, the proximity to the true or reference Pareto front and the quality of the single best solution [Si et al. (2019)]. The first three metrics show to be most robust, where the last two require educated guesses and domain knowledge.

The hypervolume indicator is first introduced by Zitzler and Künzli (2004). It concerns one of the most used quality indicators to asses the performance of a Pareto front found using a multi-objective optimization algorithm [Guerreiro et al. (2020)]. The hypervolume indicator concerns the region dominated by the Pareto set which is bounded above by a given reference point (r), as visualized in Figure 3.3. The percentage of the area that is dominated summarizes the performance of the found Pareto front. This concerns both the size and the diversity of the Pareto optimal set, as an increase in either one of these values would increase

the dominated area. By tracking the hypervolume indicator over consecutive generations, the convergence rate of the Pareto optimal set can be analyzed as well.



(a) 3D example (b) 2D projection at $z = r_z$

p	point
p^1	(5, 5, 1)
p^2	(7, 3, 2)
p^3	(1, 7, 4)
p^4	(8, 1, 5)
p^5	(4, 2, 6)
p^6	(2, 4, 8)

Figure 3.3: Example hypervolume indicator (retrieved from: Guerreiro et al. (2020))

The actual calculation of the hypervolume indicator can be defined as shown in Equation 3.1 [Zitzler and Künzli (2004)]. Within this equation $HV(f^{ref}, X)$ implies the hypervolume indicator value, representing the size of the space that is dominated by the solution comprising the Pareto frontier. $f^{ref} \in \mathbb{R}$ refers to the reference point chosen, and $\Lambda(.)$ refers to the Lebesgue measure. This measure is defined by first defining the Lebesgue outer measure, which is defined as the infimum of all values on the interval comprised by the values in the Pareto frontier. The values obtained through calculation of this infimum, show the dominated area and thus the hypervolume indicator value.

$$HV(f^{ref}, X) = \Lambda \left(\bigcup_{X_n \in X} \left[f_1(X_n), f_1^{ref} \right] \times ... \times \left[f_m(X_n), f_m^{ref} \right] \right) \tag{3.1}$$

For multi-objective optimization, literature shows two methods to be most widely used. These methods concern the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Deb et al. (2000)] and the Multi-Objective Particle Swarm Optimization (MOPSO) algorithm [Moore and Chapman (1999)]. Numerous comparative analyses between (variations of) these algorithms are conducted, stating dissimilar results. The papers by Hlal et al. (2019); Monsef et al. (2019) show NSGA-II to be more robust than MOPSO, although MOPSO being faster to achieve convergence. This convergence speed is again noted by Huang et al. (2012), which found both algorithms able to find a good approximation of the Pareto front. But MOPSO showing to be superior in rate of convergence. However other papers show the opposite to be true. The paper by Jain et al. (2018) shows MOPSO to be superior, where NSGA-II shows poor optimal solutions. Variations of the PSO-based algoritms show even better performance. The Optimized MOPSO (OMOPSO) algorithm outperforms NSGA-II on different ZDT benchmark functions [Parsopoulos and Vrahatis (2008)]. Also the Speed-constrained Multi-objective PSO (SMPSO) [Nebro et al. (2009)] shows to be the most salient technique in terms of quality of the approximation to the Pareto front found, and it also shows fast convergence towards this front. Mainly due to the complexity of the fitness landscape of the WDCP central to this research, it is decided to use the more robust NSGA methods for this research. More specifically the NSGA-III [Deb and Jain (2013)] will be further investigated,

which performance will be benchmarked against its predecessor NSGA-II and the SMPSO algorithm.

3.2.3 Adaptive Operator Selection

The above mentioned algorithms originally all abide to a predefined set of operators and their parametric values. According to the "no free lunch" theorem, no operator exists that outperforms all other operators on all optimization problems [Wolpert and Macready (1997)]. Every specific multi-objective optimization process requires a carefully selected set of operators and their hyperparameters, which can be attained through empirical comparison [Lindauer et al. (2015)]. However, for many real-word problems such a trial-and-error approach is computationally expensive and thus impractical [He et al. (2019)]. Through implementation of Adaptive Operator Selection (AOS), the best operator can be determined during optimization. This prevents the high computational burden of optimizing the parametric values of these operators beforehand.

Existing methods for AOS often relate to methods solving multi-armed bandit problems [Auer et al. (2002)]. These problems relate to the slot machines in casinos, where the objective is to maximize the obtained reward playing over several arms without knowing the probability distributions of these rewards. For operator selection both a credit assignment strategy, rewarding an operator according to the fitness improvement attained, and a selection strategy, selecting the next operator, should be designed [Tian et al. (2022)]. This process is visualized in Figure 3.4 below.

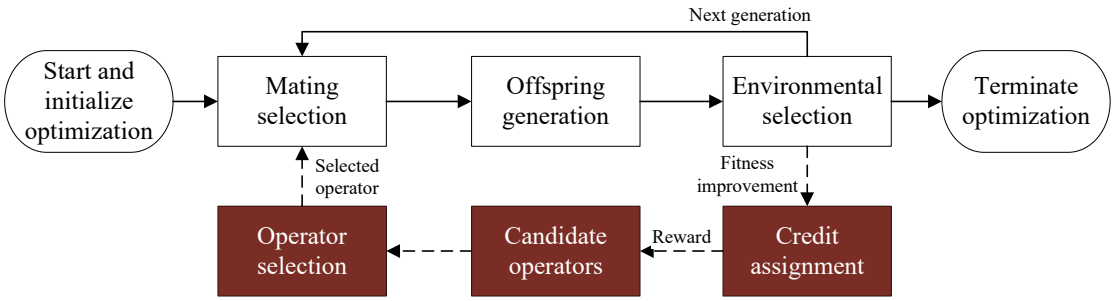


Figure 3.4: General procedure adaptive operator selection in evolutionary algorithm (retrieved from: Tian et al. (2022))

The original idea for operator selection applied to single-objective optimization [Davis (1991)]. The most common approach of credit assignment concerned the improvement of objective value brought by the created newly proposed offspring solutions [Lin et al. (2016)]. For multi-objective optimization the improvement of objective value is arbitrary, as the process often concerns optimization of conflicting objectives. Earlier credit assignment strategies for multi objective optimization problems rewarded an operator if the generated offspring solution dominates its parents [Huang et al. (2007)] or dominates other solutions from the previous generation [McClymont and Keedwell (2011)]. More recent work shows the hypervolume indicator of the whole population as the assigned reward [Huang et al. (2021)].

A problem arising with the implementation, as visualized in Figure 3.4, is that the operators are directly selected according to their recent rewards [Tian et al. (2022)]. Resulting in operators with higher rewards being selected more frequently, creating a positive feedback loop which further improves the reward for these operators. A solution to this bias came with the implementation of the Upper Confidence Bound (UCB) algorithm [Li et al. (2013)], which enables to strike a fair balance between exploration and exploitation in the multi-arm bandit problem. Still these methods are affected by the limited number of trials and the operator selection strategy choosing upon historical rewards. Selecting operators which performed well in previous generations might be ineffective in future generations. The paper of Tian et al. (2022) proposed the implementation of Deep Reinforcement Learning (DRL) to successfully alleviate these problems. Implementing DRL as a method for AOS makes it aware of the current state of optimization, allowing for different behavior to emerge based on the optimization progress achieved until the moment of selection. This alleviates both the bias in both the credit assignment and operator selection parts of AOS, as visualized in Figure 3.4.

3.3 Reinforcement learning

Reinforcement learning (RL) is a computational approach to automatic goal-directed learning and decision making [Sutton and Barto (2018)]. The emphasis of RL is on learning by an agent from direct interactions with its environment, which distinguishes it from other computational approaches. Learning occurs through trial-and-error, in which a RL agent interacts with its environment, observes the consequences of its actions and alters its own behavior in response to rewards received. This learning paradigm has its roots in behavioral psychology and is one of the main foundations of RL [Sutton and Barto (2018)]. The RL framework is visualized in Figure 3.5. Through multiple episodes of taking actions and receiving rewards, the agent is able to build an explicit map of the environment. Using this map, the RL agent is able to find an optimal policy through greedy execution of the best action in all states.

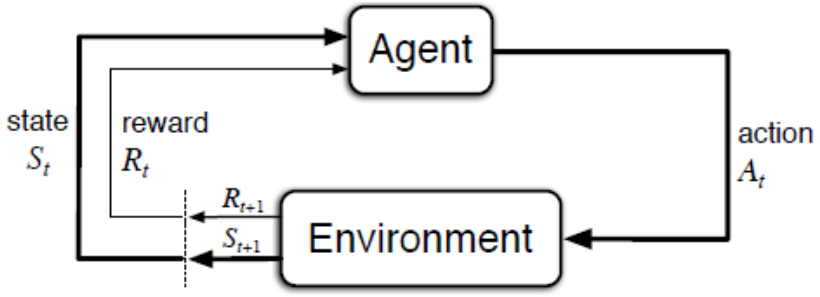


Figure 3.5: Reinforcement learning framework (retrieved from: Sutton and Barto (2018))

Reinforcement Learning can be described as a Markov Decision Process (MDP), which consists of:

1. A set of states $\mathcal{S}$;
2. A set of actions $\mathcal{A}$;
3. Transition dynamics $\mathcal{T}(s_{t+1}|s_t, a_t)$

4. An immediate reward function $\mathcal{R}(s_t, a_t, s_{t+1})$

5. A discount factor $\gamma \in [0, 1]$

The goal of an agent is to learn an optimal policy π^*, i.e. control strategy. This policy should maximize the expected return, i.e. cumulative, discounted reward. Mathematically formulated this can be summarized as:

$$\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1} \mid \pi\right] \tag{3.2}$$

A key concept underlying RL within the MDP framework is the Markov Property, which implies that the future is conditionally independent of the past given the present state [Arulkumaran et al. (2017)]. This property is held by the majority of RL algorithms, although requiring states to be fully observable which is somewhat unrealistic.

3.3.1 Deep learning

Deep Learning uses Artificial Neural Networks (ANNs), which in itself are bio-inspired algorithms inspired by the functioning of the (human) brain [Fan et al. (2020)]. Figure 3.6 shows a basic type of ANN called a fully-connected, feed forward neural network. Such networks, as all ANNs, consists of an input and output layer with one or more hidden layers in between. Each link between nodes of consecutive layers have an associated real-valued weight, which roughly corresponds to the efficacy of a synaptic connection in a real brain [Sutton and Barto (2018)]. Through iteratively feeding input-output combinations to the ANN, the mapping from input to output is learned by modification of these synaptic weights [Guresen and Kayakutlu (2011)]. After training, the network can be given new input values which it will transform into output values, resulting in the prediction of the trained network.

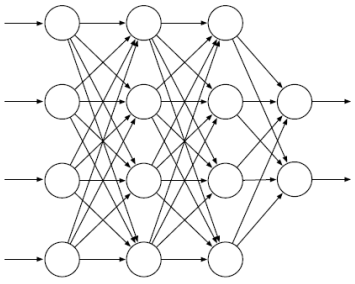


Figure 3.6: Feed forward neural network (retrieved from: Sutton and Barto (2018))

3.3.2 Deep reinforcement learning

Reinforcement learning (RL) concerns a framework for sequential decision making. Through learning the values of taking certain actions when in a given states, an agent is able to navigate its environment maximizing the expected future returns. Despite its success in the past, the original RL approaches lacked scalability and were inherently limited to fairly low-dimensional problems [Arulkumaran et al. (2017)]. This due to RL algorithms sharing the same complexity issues as other algorithms, namely: memory-, computational- and sample

complexity [Strehl et al. (2006)]. The tools to overcoming these problems developed through the implementation of deep learning, relying on powerful function approximation and representation learning properties of (deep) neural networks [Arulkumaran et al. (2017)]. The use of deep learning within RL algorithms defines the field of Deep Reinforcement Learning.

Within Deep Reinforcement Learning (DRL) the policy becomes a function that implicitly maps between state features and actions, as opposed to keeping an explicit map in the form of a lookup table [Sharma et al. (2019)]. This implicit mapping concerns a value approximation, instead of an exact value calculation. The key issue Deep Learning solves is that of generalization [Sutton and Barto (2018)]. As Neural Networks are able to handle unseen data, an approximation of the complete state space using only a limited subset of interactions can be created. DRL accelerated interest in Reinforcement Learning, as it enabled to handle high-dimensional state and action spaces that were previously intractable.

3.3.3 Applications in bio-inspired metaheuristics

As the field of Deep Reinforcement Learning (DRL) is relatively new, the amount of papers written on the implementation of DRL in Bio-inspired metaheuristics is limited. Most of the implementations found focus on adaptive selection of mutation strategies within Differential Evolution (DE) models. The paper of Sharma et al. (2019) proposes an agent based on Double Deep Q-Networks (DQN) [Van Hasselt et al. (2016)] to learn the selection of the optimal mutation strategy. The state representation used focuses on the locations and distribution of separate individuals in the solution space. Also three different reward functions are tested, of which the one focusing on improving upon the global best solution shows superior performance. The paper of Tan and Li (2021) also focuses on DE and shows an implementation of DQN. Instead of focusing the state representation on separate individuals, as in the paper of Sharma et al. (2019), the state representation focuses on the complexity of the fitness landscape.

Another approach of DRL in DE is described in the paper of Sun et al. (2021). They implemented a Policy Gradient method, which enables online learning of the agent. The agent is trained to propose optimal values for mutation, selection and crossover. These values are proposed through the use of a recurrent neural network, a variant of the Neural Network which includes memory and is thus able to learn temporal correlations. The variation of the recurrent neural network used is called Long Short-Term Memory [Hochreiter and Schmidhuber (1997)]. The experimental results show that through continuous training of the Long Short-Term Memory, a performance is attained that is competitive with current state-of-the art algorithms. Although reaching the desired performance, the need for computational resources make larger problems rather difficult to solve.

Where most papers focus on Differential Evolution, only one paper is found to focus on another bio-inspired algorithm. The paper of Durgut and Aydin (2021) focuses on the Artificial Bee Colony (ABC) Optimization algorithm. The authors found the current operator selections schemes to be independent of state, which is a problem which can be solved through the implementation of Reinforcement Learning (RL). By creating a reinforced-clustering algorithm, the optimal operator can be selected regarding the position and distribution of individual bees. The possible operators to be selected consist of the current state-of-art selec-

tion schemes [DaCosta et al. (2008)], namely: Probability Matching (PM), Adaptive Pursuit (AP) and Upper Confidence Bound (UCB). It is shown that the proposed algorithm outperforms the state-of-art algorithms on the Uncapacitated Facility Location (UFL) problem benchmark.

3.3.4 Deep Q-Network algorithms

Based upon discussed previous work, the algorithm to be used in this research concerns the offline, model-free Deep Q-Network (DQN) algorithm [Mnih et al. (2015)]. Being an offline algorithm implies the agent not learning immediately upon interactions but learning from an archive of previous interactions. Two main features characterizing a DQN agent are a frozen target network and experience replay. A frozen target network consists of the agent having two neural networks, one policy network used for action selection and one value network used to approximate state values. To prevent inconsistency in the policy being followed, the policy network is kept constant while the value network is updated during training. After a set number of iterations the learned weights of the value network are copied onto the policy network, adapting the behavioral policy of the agent. Using a frozen target network makes training more stable by preventing short-term oscillations from a moving target [Mnih et al. (2013)]. Experience replay implies that all state (s_t), action (a_t), reward (r_t) and next state (s_{t+1}) pairs are stored. Upon training this experience data is (randomly) sampled and used to train the value network, which makes DQN a so-called off-policy method. Using experience replay improves sample efficiency and tackles auto-correlations between consecutive agent interactions that would otherwise create a bias in its behavior [Mnih et al. (2013)].

3.4 Conclusion and position in literature

The executed literature review will, besides answering sub-question 2, 3 and 5, also form the basis for designing a solution method. In conclusion, the optimization algorithm used will consist of the NSGA-III. The main reason for this is the inherent robustness of the algorithm. Due to the non-convexity of the WDCP central to this research, this is perceived a desirable characteristic. To combat the high computational cost of the NSGA-III algorithm, AOS will be implemented. The implementation of AOS will be done using DRL, as this shows to be the most salient method to mitigate the effect of the limited number of trials and of choosing upon historical reward. The implementation of DRL will be done using the DQN algorithm. DQN shows beneficial due to its sample efficiency and robustness. These characteristics are attained through the use of experience replay and a frozen target network respectively.

Successful implementation of the proposed method, and thus execution of this research, will contribute to literature on the following levels:

3.4.1 Novel solution method to the warehouse design and control problem

As discussed in section 3.1, current-day literature does not show extensive research into the Warehouse Design and Control problem. Only two papers are found concerning simultaneous optimization of warehouse layout design and control policies. Both these papers [Heragu et al. (2005); Roodbergen et al. (2015)] focus on warehouse layout design and product allocation. Where the paper by Heragu et al. (2005) uses single objective Simulated Annealing (SA),

transforming the multiple objectives into a single objective through a cost function. The paper by Roodbergen et al. (2015) only concerns a single objective, namely the average travel distance per order, which is optimized using a proposed heuristic method.

This research extends this optimization process by including resource allocation. In comparison to the example of Heragu et al. (2005), optimization will be executed maintaining separate values for the multiple objective values. This is a novel approach shifting the way to trade-off multiple objectives from an apriori method, characterized by calculating a single objective through a predefined cost function, to a Pareto dominated method. The benefit of optimizing the objective values separately is that the importance of different objectives does not have to be predefined. this enables the decision maker to decide upon this after running the optimization algorithm, which circumvents potential biases of the algorithm. Also it makes it possible to construct a fully generalizable model. Allowing it to be applied to different optimization problems without requiring to define problem specific settings.

3.4.2 Application of Deep Q-Networks in NSGA-III

Despite the relative novelty of DRL, literature shows several examples of DRL implemented as AOS to guide meta heuristics. Most of these methods apply DQN to Differential Evolution (DE) [Sharma et al. (2019); Tan and Li (2021); Sun et al. (2021)].

This research extends the literature of AOS using DRL, by the novel implementation of DQN within NSGA-III [Deb and Jain (2013)]. This implementation allows for AOS while maintaining separate objective values, making it compatible with multi-objective optimization. Through adaptively setting the parametric values of the operators of the NSGA-III, it is aspired to improve the convergence properties of the optimization model. Besides its novelty, the improved convergence properties are mandatory to solve the WDCP central to this research.

Chapter 4

Solution methods

This chapter will elaborate upon the proposed solution methods used. These methods were applied to attain the overall goal of simultaneously optimizing upon layout design and control policies for the warehouse of the plastic manufacturer. First the environment will be discussed, which consists of a Discrete Event Simulation (DES) which is a created, accelerated replication of the simulation used internally by Nobleo. On top of this simulation an optimization model is created. A detailed functional explanation of the model of choice, the Non-dominated Sorted Genetic Algorithm III (NSGA-III), will be discussed. To increase the performance of the NSGA-III, Deep Reinforcement Learning (DRL) will be applied. The final section will explain the used DRL algorithm, for which a detailed discussion is presented regarding the implementation used.

4.1 Simulation environment

As discussed in section 1.1, the problem context, the simulation model used by Nobleo faces high computational cost. A single evaluation of a warehouse configuration with a duration of 40 hours takes around 30 minutes, lending the optimization problem infeasible. The reason for this high computation time mainly has to do with the simulation software Enterprise Dynamics (ED) used and the amount of detail used in the simulation. ED concerns software to run semi discrete event time simulations, being discrete event in their calculations but running continuous time on top of this to enable visualizations. The ED software puts great emphasis in comprehensibility through these detailed visualizations of the executed process. Besides, the ED software does not allow for multiple simulation runs to be executed in parallel.

4.1.1 Python replication

To alleviate the problem of high computational cost, the ED simulation is replicated in Python. This replication shows different to the ED simulation in the following ways:

- **Simulating discrete time**
  The first major distinction between the ED simulation and the Python replication concerns the perception of time. The Python replication simulates in so called discrete time, using the technique called Discrete Event Simulation (DES). As opposed to continuous time simulation, a DES only simulates occurrences, i.e. events. The main functionality

of a DES lies in the so-called Future Event Set (FES), which is a chronologically sorted list of events. Upon initialization the first event(s) are added to the FES, after which the first event with the earliest execution time is taken and executed. Execution consists of handling the event accordingly, after which a follow-up event is added if required. This process is schematically visualized in Figure 4.1.

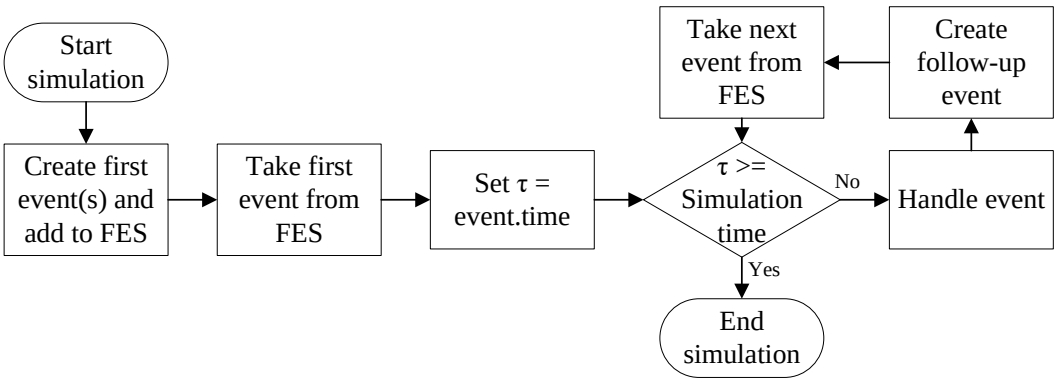


Figure 4.1: Flowchart of Discrete Event Simulation (DES) functionality

- **Pruning of excessive details**
  The level of detail in the ED simulation of Nobleo is high, as it is desired by their client to gain insights into all facets of the process. For the optimization model this level of details shows to be excessive and can be eliminated, as only the objective values have to be correctly extracted. Also the implementation of different tasks in the simulation can be aggregated. For example, loading a truck with 20 pallets can be dissected in 20 separate tasks. But these tasks have to be executed consecutively by the same resource, which allows for it to be seen as one large tasks which can be executed all in once. Doing this results in loading the truck to be a single operation, which is preferable in the DES implementation.

- **Implementing lookup instead of full-sweeps**
  Analysis of the ED simulation of Nobleo showed that a significant amount of computation time is used for execution of full-sweeps across the warehouse. These sweeps occur every time a product is requested from or to be placed in the warehouse. During these sweeps every location in the warehouse is reviewed upon their storage type, width, height and availability. In the Python replicated simulation these characteristics of storage locations are summarized in lookup tables. This enables simple lookup to get to a list of storage locations that fit the needed description, which is thereafter looped. As this process happens a lot, the difference in implementation has a significant effect on simulation time.

- **Allowing parallel evaluation**
  Besides the aforementioned adjustments, another benefit of using Python is that it enables for running multiple simulation runs in parallel. Paralellization allows the program to run multiple simulation simultaneously, bringing down the average simulation time even further. This process is implemented using the Multiprocessing module, which is a Python standard library.

- **Increased robustness simulation model**
  Next to improvements upon computation time, also an increased robustness needed to be attained. As the simulation model will be used within an optimization model, it has to be entirely capable of dealing with extreme cases. The main difference from the ED simulation concerns the inability to dock a truck upon arrival. The ED simulation cannot handle this, leading to an error terminating the process. The Python replication uses a prioritized truck queue, which allows to handle these scenarios. If a truck arrives either there is no truck dock available, no consolidation area available or both. Trucks in need for only a dock need to be handled first, thereafter dock and consolidation area and finally the trucks only in need of a consolidation area. This has to do with the potential risk for tardiness and internal space occupation.

Both the ED simulation and the created Python replication are stochastic. The stochasticity can be found in the arrival of trucks. Also the content of inbound trucks or the requested products by outbound trucks is stochastic. To validate the performance of the replicated simulation against the ED simulation, similar random seeds are tested on both models. The results only showed minor differences, which are considered insignificant and are thus neglected. The ED simulation takes around 20-30 minutes to evaluate a single configuration. The Python replicated simulation takes for a similar evaluation on average 2-3 seconds, if multiple simulations are computed in parallel over 10 cores. An extensive explanation of the functionality of the Python replicated simulation model can be found in Appendix B.

4.1.2 System configuration encoding

As discussed in section 1.1, the decision variables concern:

- ($\mathbb{O}$) The order of the rules comprising the PPA.
- ($\mathbb{V}$) The parametric values within the rules of the PPA.
- ($\mathbb{R}$) The amount of resources per type.
- ($\mathbb{D}$) The dimensions of the storage locations.

To allow for evaluation of different configurations of these variables, these values are encoded into a single list of values. This list describes the values of these parameters and thus dictates the internal logistics of the simulated warehouse. The encoding of a single configuration for the Warehouse Design and Control Problem (WDCP) central to this research consists of 96 values and is structured as follows:

$\mathbb{O}_1$	$\mathbb{O}_2$	$\mathbb{O}_3$	$\mathbb{O}_4$	$\mathbb{V}_1$	$\mathbb{V}_2$	$\mathbb{V}_3$	$\mathbb{V}_4$	$\mathbb{V}_5$	$\mathbb{R}_1$	$\mathbb{R}_2$	$\mathbb{R}_3$	$\mathbb{D}_1$	$\mathbb{D}_2$	$\mathbb{D}_3$	...	$\mathbb{D}_{82}$	$\mathbb{D}_{83}$	$\mathbb{D}_{84}$

Figure 4.2: Encoding format of a warehouse configuration

4.1.3 Objective values

The objective values describe what is desired to be achieved by the optimization model. These values need to describe the overall functionality of a certain system and should be set to the Key Performance Indicators used within the company under review. Three objective values are considered to evaluate the performance of a warehouse configuration, namely:

- **tardiness of outbound trucks**
  The main objective value concerns the outbound truck performance, classifying all outbound trucks ($\mathcal{T}$) in three categories. Category one concerns the so-called "Perfect trucks", which have at most 30 minutes between their arrival (a_t) and departure (d_t) times. Truck departures between 30 and 120 minutes after their arrival are considered "OK", where later trucks are considered "Too late". The following formula shows the calculation of the performance, adding a penalty constant (p_t) to lower the severity of "OK" trucks and emphasizing the need to minimize "Too late" trucks:

$$Obj_1 = \sum_{k=0}^{\mathcal{K}_{outb}} f_k \cdot (d_k - a_k) \qquad f_k = \begin{cases} 0.0 & if \quad (d_k - a_k) \leq 30\,min. \\ 0.5 & if \quad 30\,min. < (d_k - a_k) \leq 120\,min. \\ 1.0 & if \quad (d_k - a_k) > 120\,min. \end{cases} \tag{4.1}$$

  The reason it is decided to not minimize the total handling time of all trucks is to prevent the algorithm from minimizing the handling time of trucks that are considered "Perfect". This decision can be seen as some form of objective value clipping, where a truck handling time below 30 minutes is considered perfect and in no need for improvement. The handling times between 30 and 120 minutes are also considered "OK", but minimization is still desired to prevent it to be on the edge of becoming a "Too late" truck.

- **Resource cost**
  The second objective concerns the resource cost and is calculated multiplying the cost of a resource (c_i) times the number of resources used (r_i) per resource type ($\mathcal{Z}$). The resource cost is an aggregate of investment and estimated operating costs, which allows for a fair evaluation of the cost made in a given configuration. The calculation of the resource cost is mathematically formulated below:

$$Obj_2 = \sum_{z \in \mathcal{Z}} c_z \cdot r_z = c_1 \cdot r_1 + c_2 \cdot r_2 + c_3 \cdot r_3 \tag{4.2}$$

- **Unplaceable products**
  The final objective value is set to prevent the simulation of removing a lot of products. An unplaceable product concerns a product of a given height and width that cannot be placed in the storage type retrieved from the Product Placement Algorithm (PPA). Unplaceable implies that not only the product cannot be placed in storage locations equal to the product dimensions. It also implies unavailability of all scale up locations, which concern storage location either larger in height, width or both.

The tardiness of trucks was already a metric used within Nobleo, where the other two are constructed for this research. Resources are currently monitored by Nobleo, however the resource cost concerns a cost function. This enables to include a difference in severity of adding either a forklift or a reach truck. The unplaceable products objective is added after thorough evaluation of the simulation dynamics and the obtained results on different extreme scenarios. These results showed unrealistically optimistic performance, which later showed to be a result of sending a lot of products in the implemented fall-back which concerns a sink.

4.2 Optimization algorithm

Based upon literature, the optimization model to be used concerns the Non-dominated Sorting Genetic Algorithm III (NSGA-III) [Deb and Jain (2013)]. The main reason being its robustness, which is desired as the problem central to this research shows to be of high complexity due to the non-linearity and non-convexity. The NSGA-III concerns the optimization algorithm, where the replicated simulation will be used as the evaluation function. A potential solution, called an individual, is composed of 96 values and is encoded as discussed in section 4.1.2. The NSGA-III follows the general process of a Genetic Algorithm (GA), which is visualized in Figure 4.3.

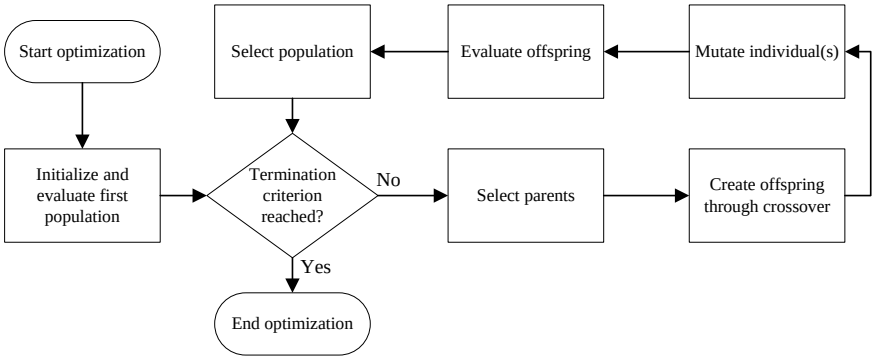


Figure 4.3: General process Genetic Algorithm (GA)

The optimization process is started by randomly initializing a set of individuals, called a population. The individuals comprising the population are taken through the simulation environment to evaluate their performance. Based on the performance and composition of the evaluated individuals, the NSGA-III will propose a new population called the offspring. This process is repeated until the set number of generations is reached. Upon which the algorithm will terminate and return the found set of optimal solutions. The NSGA-III is formally described in the pseudo code in Algorithm 3.

4.2.1 Crossover

The process of crossover uses two (well-performing) individuals of the population, called parents. These two parents are combined to create two new individuals, called childs. Through combining these parents, the algorithm intends to use the well-performing parts of both individuals to create a set of childs, called offspring, that outperforms the parents it originated from. As this process utilizes gathered knowledge about evaluated individuals, it mainly focuses on the exploitation part of the exploration-exploitation trade-off. Crossover is repeated on different combinations of parents until the offspring population is of identical size as the original population.

The NSGA-III applies crossover using Simulated Binary Crossover (SBX), as seen in line 8 of the psuedo-code in Algorithm 3. Motivated by the success of binary-coded GAs in discrete search spaces, Deb et al. (1995) developed a real-coded crossover operator which is the Simulated Binary Crossover (SBX). Concluded from a number of real valued tests, the performance shows equal or superior than binary-coded GAs using single-point crossover. SBX is found

Algorithm 3: Non-dominated Sorting Genetic Algorithm III (NSGA-III) [Deb and Jain (2013)]

INPUT:
$\mathcal{H} \leftarrow$ set of structured reference points
$Generations \leftarrow$ number of generations to execute
$\vec{LB} \leftarrow$ lower bounds of all values
$\vec{UB} \leftarrow$ upper bounds of all values
$\mathcal{N} \leftarrow$ the desired number of individuals comprising a population
$\mathcal{G} \leftarrow$ the desired number of generations to execute

OUTPUT:
F_1 final pareto frontier found

```
1  P_0 ← randomly initialized population of individuals;
2  for g ← 0 to 𝒢 do
3  |   Q_g ← ∅;
4  |   Parent_Combinations ← ∅;
   |
5  |   for i ← 0 to 𝒩 by 2 do
6  |   |   Parent_Combinations ⋃ (P_{g; i}, P_{g; i+1})
   |   end
   |
7  |   for each (parent_1, parent_2) in Parent_Combinations do
8  |   |   (child_1, child_2) ← Simulated_Binary_Crossover(parent_1, parent_2);
9  |   |   Q_g ← Q_g ⋃ Polynomial_Mutation(child_1);
10 |   |   Q_g ← Q_g ⋃ Polynomial_Mutation(child_2);
   |   end
   |
11 |   R_g ← P_g ⋃ Q_g
12 |   (F_1, F_2, ...) ← Non_Dominated_Sort(R_g);
13 |   P_{g+1} ← Reference_Based_Selection((F_1, F_2, ...));
   end
```

particularly useful in problems having multiple optimal solutions with a narrow global basin and in problems where the lower and upper bounds of the global optimum are not known apriori [Deb et al. (1995)].

Pseudo code for the SBX algorithm is shown in Algorithm 4. The SBX loops over all values comprising an individual and applies crossover with 50% probability. When crossover is applied two values, β and α, are calculated. Based on a new random value, these α and β values are recalculated into a β_q factor. The β_q factor concerns a value between 0 and 1, which is thereafter used to decide the difference between the value seen in the parents and the value to be used in the child. As this difference partly depends on the maximal and minimal value, as seen in line 13 and 19 in Algorithm 4, dissimilar parents will automatically result in higher dissimilarity in children than more similar parents. This behavior is desired, as it automatically adapts from exploration to exploitation as a population often starts dissimilar and gets more similar over time. Apart from this automatic balance, the crossover distribution parameter (η_{SBX}) can be used to determine the degree of difference between the parents and the offspring. A high value will create offspring resembling their parents, where a low value will create offspring that is more distinct.

Algorithm 4: Simulated Binary Crossover (SBX) [Deb et al. (1995)]

INPUT:

$\vec{Y} \leftarrow$ first individual; i.e. vector as shown in Figure 4.2
$\vec{Z} \leftarrow$ second individual; i.e. vector as shown in Figure 4.2
$\vec{LB} \leftarrow$ lower bounds of values comprising an individual;
$\vec{UB} \leftarrow$ upper bounds of values comprising an individual;
$\eta_{SBX} \leftarrow$ crossover distribution parameter value;

OUTPUT:

$\vec{V}$ first child, i.e. vector as shown in Figure 4.2
$\vec{W}$ second child, i.e. vector as shown in Figure 4.2

$\vec{V} \leftarrow \vec{Y}$;
$\vec{W} \leftarrow \vec{Z}$;

for $i \leftarrow 0$ **to** *length individual* **do**
  $rand_1 \leftarrow random[0,1]$;

  if $rand_1 \leq 0.5$ **then**
    $X_{min} \leftarrow min(Y_i, Z_i)$; $X_{max} \leftarrow max(Y_i, Z_i)$;
    $rand_2 \leftarrow random[0,1]$;

    $\beta \leftarrow 1 + \left(2 \cdot \frac{X_{min} - LB_i}{X_{max} - X_{min}}\right)$; $\alpha \leftarrow 2 - \beta^{-(\eta_{SBX}+1)}$;

    if $rand_2 \leq 1/\alpha$ **then**
      $\beta_q \leftarrow (rand_2 \cdot \alpha)^{1/(\eta_{SBX}+1)}$;
    else
      $\beta_q \leftarrow \left(\frac{1}{2 - rand_2 \cdot \alpha}\right)^{1/(\eta_{SBX}+1)}$;

    $c_1 \leftarrow 0.5 \cdot (X_{min} + X_{max} - \beta_q \cdot (X_{max} - X_{min}))$;

    $\beta \leftarrow 1 + \left(2 \cdot \frac{UB_i - X_{max}}{X_{max} - Xmin}\right)$; $\alpha \leftarrow 2 - \beta^{-(\eta_{SBX}+1)}$;

    if $rand_2 \leq 1/\alpha$ **then**
      $\beta_q \leftarrow (rand_2 \cdot \alpha)^{1/(\eta_{SBX}+1)}$;
    else
      $\beta_q \leftarrow \left(\frac{1}{2 - rand_2 \cdot \alpha}\right)^{1/(\eta_{SBX}+1)}$;

    $c_2 \leftarrow 0.5 \cdot (X_{min} + X_{max} + \beta_q \cdot (X_{max} - X_{min}))$;

    $rand_3 \leftarrow random[0,1]$;
    if $rand_3 \leq 0.5$ **then**
      $V_i \leftarrow c_2$, $W_i \leftarrow c_1$;
    else
      $V_i \leftarrow c_1$, $W_i \leftarrow c_2$;
end

4.2.2 Mutation

After crossover all created offspring individuals will be exposed to mutation. Where crossover focuses on exploiting well-performing individuals, mutation intends to diversify the population and thus induces exploration. Mutation is applied by looping over all values comprising an individual, as explained in section 4.1.2, which will be adjusted given a certain probability (p_m). The way these values are adjusted depends on the mutation algorithm used. For the NSGA-III Polynomial Mutation [Deb et al. (1995)] is applied, as seen in line 9 and 10 of the pseudo-code shown in Algorithm 3.

More specifically, the highly disruptive Polynomial mutation [Hamdan (2010)] is used, which allows for larger jumps around the search space. Polynomial Mutation (PLM) loops over all values comprising an individual. With the independent mutation probability (p_m) a value is mutated, which implies changing the value sub-randomly. If mutation will be applied, two β values are calculated. Both values sum to one, where β_1 depicts the fraction of space between the value of the individual and the Lower Bound (LB) and β_2 towards the Upper Bound (UB). Thereafter another random value is sampled, which decides in which direction the value will be changed. If this random value is lower or equal to 0.5 the mutated value will be sampled from the β_1 range, implying the range between the current value and the Lower Bound (LB). If the random value is higher than 0.5, the opposite is true and the value will be sampled from the range between the current value and the Upper Bound (UB). As with the SBX algorithm in Algorithm 4, the mutation distribution parameter (η_{PLM}) dictates the magnitude of difference between the current value and the sampled value. Psuedo-code for the PLM algorithm is shown in Algorithm 5.

4.2.3 Reference-based Selection

After the creation and mutation of offspring, the created individuals are evaluated using the replicated simulation model. After evaluation there are both an evaluated parent population and an evaluated offspring population. Both populations are equal in size, resulting in a set of individuals twice the size of the original population ($\mathcal{N}$). Selection concerns the process of reducing this pool of individuals to the predetermined population size ($\mathcal{N}$).

As opposed to its predecessor, the NSGA-II, NSGA-III maintains diversity among its solutions through the use of an external guidance mechanism [Seada and Deb (2015)]. NSGA-III uses so-called Reference-based Selection (RBS), as shown in line 13 of Algorithm 3. This mechanism consists of a predefined, uniformly distributed set of reference directions ($\mathbb{Z}_r$). Each individual is attempted to adhere to one of these reference directions in a process called niching [Seada and Deb (2015)]. The improvement upon NSGA-II lies in the assurance of a globally diversified population. The crowding distance of NSGA-II only ensures diversity on the found set of individuals, where NSGA-III through reference directions looks at the entire search space. To emphasize the difference, both diversification mechanisms are visualized in Figure 4.4 and 4.5. The black dots symbolize selected individuals, whereas the white dots concern discarded individuals. As can be seen, NSGA-II selects different individuals than NSGA-III, resulting in a lower global diversity of the population.

Algorithm 5: Polynomial Mutation (PLM) [Liagkouras and Metaxiotis (2013)]

INPUT:

$\vec{X} \leftarrow$ individual; i.e. vector as shown in Figure 4.2
$\vec{LB} \leftarrow$ lower bounds of values comprising an individual;
$\vec{UB} \leftarrow$ upper bounds of values comprising an individual;
$p_m \leftarrow$ independent mutation probability (indpb);
$\eta_{PLM} \leftarrow$ mutation distribution parameter value;

OUTPUT:

$\vec{Z}$ mutated individual, i.e. vector as shown in Figure 4.2

$\vec{Z} \leftarrow \vec{X}$;
for $i \leftarrow 0$ **to** *length individual* **do**
  $rand \leftarrow random[0, 1]$;
  if $rand \leq p_m$ **then**
    $\delta_1 \leftarrow \frac{X_i - LB_i}{UB_i - LB_i}, \qquad \delta_2 \leftarrow \frac{UB_i - X_i}{UB_i - LB_i}$;
    $r \leftarrow random[0, 1]$;
    if $r \leq 0.5$ **then**
      $\delta_q \leftarrow \left[2r + (1 - 2r)(1 - \delta_1)^{\eta_{PLM}+1}\right]^{\frac{1}{\eta_{PLM}+1}} - 1$;
    else
      $\delta_q \leftarrow 1 - \left[2(1 - r) + 2(r - 0.5)(1 - \delta_2)^{\eta_{PLM}+1}\right]^{\frac{1}{\eta_{PLM}+1}}$;
    $Z_i \leftarrow X_i + \delta_q (UB_i - LB_i)$;
    $Z_i \leftarrow min(max(Z_i, LB_i), UB_i)$;
end

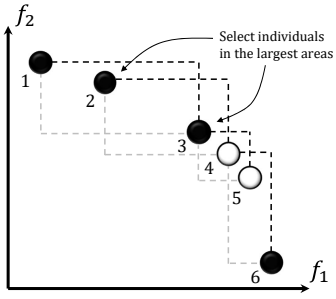


Figure 4.4: NSGA-II: Crowding distance selection

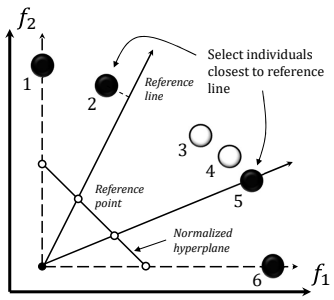


Figure 4.5: NSGA-III: Reference-based selection

To further explain the functionality of the Reference-Based Selection (RBS) operator of the NSGA-III, pseudo-code is shown in Algorithm 6. What can be seen is that the creation of a new population starts with the non-dominated solutions (F_1), thereafter the once-dominated (F_2) and so on. This happens until the next tier of dominated solutions does not entirely fit inside the population no more. The remaining space in the population is filled using a process called Niching, in which all individuals are associated to a reference direction. Based upon the niche count of every reference direction (ρ_j), additional individuals are selected to be added to the next population (P_{g+1}).

Algorithm 6: Reference-Based-Selection (RBS) [Mkaouer et al. (2015)]

INPUT:
$(F_1, F_2, ...) \leftarrow$ list of different tiers of Pareto fronts;
$\mathcal{N} \leftarrow$ the desired number of individuals comprising a population;
$\mathbb{Z}_r \leftarrow$ set of reference directions

OUTPUT:
P_{g+1} Next (selected) population of individuals

$S_t \leftarrow \emptyset$;
$i \leftarrow 1$;

do
  $S_t \leftarrow S_t \bigcup F_i$; $i \leftarrow i + 1$;
while $|S_t| \leq \mathcal{N}$
$F_l \leftarrow F_i$;

if $|S_t = \mathcal{N}$ **then**
  $P_{g+1} \leftarrow S_t$;
else
  $P_{g+1} \leftarrow \bigcup_{j=1}^{l-1} F_j$;
  $K \leftarrow \mathcal{N} - |P_{t+1}|$;
  $[\pi(s), d(s)] \leftarrow Associate(S_t, \mathbb{Z}_r)$;
  $\rho_{j \in \mathbb{Z}_r} \leftarrow 0$;
  for $s \in S_t / F_l$ **do**
    if $\pi(s) = j$ **then**
      $\rho_{j \in \mathbb{Z}_r} \leftarrow \rho_{j \in \mathbb{Z}_r} + 1$
  end
  $P_{g+1} \leftarrow Niching\left(K, \rho_{j \in \mathbb{Z}_r}, \pi(s), d(s), \mathbb{Z}_r, F_l, P_{g+1}\right)$;

4.3 Deep Reinforcement Learning agent

To mitigate the expensive computation cost of using the NSGA-III algorithm, Adaptive Operator Selection (AOS) will be applied. As a sequential decision making framework, Deep Reinforcement Learning (DRL) shows well-suited for this task [Tian et al. (2022)]. The sequential decision making needed in the implementation of DRL in AOS concerns the setting the operator values dynamically. Loosely based on the "No free lunch" theorem [Wolpert and Macready (1997)], no specific operator outperforms all other operators on all optimization problems, no set of operators shows the be superior to all other in all stages of optimization. Based on previous interactions, consisting of a state, action, reward and next state, a DRL agent can learn to make sequential decisions to optimize the reward obtained. After learning, an agent will be able to look at the current state of optimization and choose the most beneficial set of operators for the desired outcome.

Compared to other methods of implementing AOS, DRL is able to alleviate the problems that arise due to the limited number of trials and the operator selection strategy choosing upon historical rewards. These problems result in the expectation that operators that performed well in the past generations will perform well in future generations, which does not have to be the case in NSGA-III. The use of DRL impedes this problem, as it allows for different

behavior to emerge based on the optimization progress achieved until the moment of selection.

The implementation of DRL will be done using the offline, model-free Deep Q-Networks (DQN). DQN, being a DRL algorithm, intends to learn predicting the values of being in a certain state. If it is able to get a correct understanding of these values, the agent will learn to understand its environment. This understanding lets the DQN agent in every state select the action leading to the next state with the highest value. Taking these most "profitable" actions will result in the creation of a so-called optimal behavioral policy, which tells the agent to take which action in which state. As evaluation time is the main concern of this research, training needs to happen using as little evaluations as possible. In this case, DQN has some beneficial characteristics. These characteristics concern sample efficiency and robustness, which are attained through implementation of experience replay and a frozen target network respectively. Experience replay concerns a buffer of interactions, frequently sampled to train the weights of the underlying neural network. The frozen target network, on the other hand, implies the agent using two neural networks. One network is used to dictate the behavior of the agent, where the other is used to predict the value of being in a certain state. The weights of the value network are repeatedly updated, using samples taken from the experience replay buffer. Only periodically, the learned weights will be copied onto the behavioral network, which dictates the behavior of the agent. This way the behavior of the agent remains stable, preventing it to fall into a positive feedback loop chasing an actions returning a desired return.

The variation of the DQN used is called the Dueling DQN. The additional characteristic concerns a so-called Advantage function [Sewak (2019)]. This function returns an advantage value for all possible actions the agent can take, when in a certain state. These advantage values tell how much better it would be to take action a_k in state s over all other possible actions $a \in \mathcal{A}$ in state s. The advantage values together with the state values are used to calculate the Q-value(s), using the following equation:

$$Q(s; a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha))$$

The main benefit of using the advantage function besides the state values is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm [Wang et al. (2016)]. Dueling DQN is beneficial if there could be cases where two actions have identical value, which is likely to happen given the environmental design the agent is deployed in.

4.3.1 State representation

The state representation concerns a description of the current state the agent is in. This representation will be taken through the behavioral network of the DQN agent, which will return an action to be executed. The design of this state representation should abide to two factors. First, it should sufficiently describe the environment to enable the agent to learn a policy that maximized the obtained rewards. Secondly, all variables comprising the representation need to be made generic, as the purpose of the proposed solution method is to be generalizable. The constructed representation focuses on three parts, namely:

1. The progress of the evolutionary process
2. The performance and spread of the individuals comprising the population
3. The performance and size of the found Pareto optimal set

First the evolutionary progress is measured using two variables, the current generation ($\mathcal{G}$) and the stagnation counter ($\mathcal{S}$). The current generations ($\mathcal{G}$) gives the agent insight into how much time remains to optimize, where the stagnation counter ($\mathcal{S}$) shows the agent how many generations it did not produce any improvement. The stagnation counter is clipped, meaning that a maximal value is set. The stagnation counter value cannot exceed 10, which is a boundary that is only reached in extreme cases concluded from empirical evaluations. Through clipping a certain range of values is enforced, ensuring that the value shown to the DQN agent remains its value by preventing it from becoming too large.

The population performance and spread is summarized in three values. The first value takes the average of all normalized objective values (O_{mean}), giving the agent some indication of the overall performance of the population. The second value concerns the average of all normalized minimal values per objective value (O_{min}), focusing on the best obtained values found so far. Where the third and final value describes the average normalized standard deviations (σ) of all objectives. The reason the values are aggregated into a single value is to allow for generalization. By using aggregates the agent can be applied to any optimization problem, independent of the number of objective values considered.

The last part of the state representation describes the performance of the current approximation of the Pareto front. This performance is summarized in two metric, the hypervolume indicator ($\mathcal{H}$) and the size of the Pareto set (PS). The hypervolume indicator ($\mathcal{H}$) is the most used set-quality indicators for the assessment of multi-objective optimizers where the actual Pareto front is unknown [Guerreiro et al. (2020)]. Calculation of this metric uses Equation 3.1, discussed in section 3.2.2. The Pareto size (PS) is included to show the agent to what extend the maximum size of the front is reached. If the pareto size has not reached its maximum, there is a possibility to extend the Pareto optimal set without the need to discard another solution.

4.3.2 Action space

The action space of the agent concerns the decisions it is able to make. With regards to the NSGA-III, three values are considered to be included in this space. These values are summarized in Table 4.1 below, also showing the current values used in the initial implementation of the NSGA-III [Deb and Jain (2013)] and describing the effect of high and low values.

The values described in Table 4.1 are solemnly considered. Actual construction of the action space relies on additional information gained from the process of Bayesian hyperparameter optimization, discussed later in section 5.2. This information will create insight into feature importance and promising parametric ranges. The importance of the features will decide on inclusion or exclusion of certain variables in the action space. The actual action space will be made discrete, having their values uniformly taken from the parametric ranges gathered from the hyperparameter optimization. The main reason the action space is made

	η_{SBX}	η_{PLM}	$indpb_{PLM}$
Operator	Crossover	Mutation	Mutation
Description	Crossover distribution parameter	Mutation distribution parameter	(Independent) mutation probability
NSGA-III value	30	20	0.01
Effect of high value	Produce children resembling their parents	Produce mutation resembling its origin	Higher chance for values to be mutated
Effect of low value	Produce children dissimilar to their parents	Produce mutation dissimilar to its origin	Lower chance for values to be mutated

Table 4.1: Potential action space Deep Reinforcement Learning (DRL) agent

discrete is that small changes in the considered values do not impact behavior significantly. By setting discrete value pairs the selection of different action can be made significantly different, which is desirable for the implementation of the DQN agent.

4.3.3 Reward function

The intended purpose of implementing DRL as AOS in the NSGA-III is to improve its convergence properties with regard to speed while maintaining solution quality. The implementation of the DQN agent is not immediately on the problem, instead its environment consists of the optimization algorithm, more specifically NSGA-III. For this reason the objective of the agent is different than the actual objectives of the Warehouse Design and Control Problem (WDCP) central to this research. The actual objectives of the WDCP concern the optimization objective values, discussed in section 4.1.3, whereas the DRL objective concerns fast convergence to an Pareto front while maintaining the quality of the found set of solutions. For this reason, choosing a reward function shows to be nontrivial. According to the paper by Karafotias et al. (2015), scale sensitivity is of high importance when constructing a reward function for this purpose. Within the NSGA-III, this implies the relative difference in attainable improvement comparing the first generations to the final generations.

The reward functions tested are partly based upon the ones proposed in Karafotias et al. (2015). Extending this list with a reward function focused on the number of dominations [Huang et al. (2007)] and an episodic reward, calculated by the sum of all hypervolume indicator values (calculated using Equation 3.1) over an entire optimization run [Huang et al. (2021)]. In total five reward functions are tested, which are briefly introduced in Table 4.2. The mathematical formulations use $\mathcal{G}$ to represent the desired number of generations to execute until termination of the NSGA-III.

Three of the five proposed reward functions show to be scale insensitive, which concerns reward function A, D and E. It is expected that one of these three functions will perform best. However, the scale sensitive rewards B and C are included for testing, as the paper by Karafotias et al. (2015) does not preclude these reward functions from having any potential to perform well.

4.3.4 Neural architecture

As previously mentioned, DRL is implemented through the Dueling DQN algorithm. The difference compared to the regular DQN algorithm is the additional advantage function, which

01. Binary improvement of hypervolume indicator	
Type	Intermediate reward
Description	A value of 1 if the hypervolume increases, 0 otherwise
Source	Retrieved from: [Karafotias et al. (2015)]
Mathematical formulation	$\text{Reward}_{gen} = \begin{cases} 1 & if \quad HV_{gen} > HV_{gen-1} \\ 0 & otherwise \end{cases} \quad \forall \quad gen \quad \in \quad [1, ..., \mathcal{G}]$
02. Real-valued improvement of hypervolume indicator	
Type	Intermediate reward
Description	Returning the difference in hypervolume indicator compared to the previous generation
Source	Retrieved from: [Karafotias et al. (2015)]
Mathematical formulation	$\text{Reward}_{gen} = HV_{gen} - HV_{gen-1} \quad \forall \quad gen \quad \in \quad [1, ..., \mathcal{G}]$
03. Raw hypervolume indicator	
Type	Intermediate reward
Description	Returning the obtained value for the hypervolume indicator
Source	Retrieved from: [Karafotias et al. (2015)]
Mathematical formulation	$\text{Reward}_{gen} = HV_{gen} \quad \forall \quad gen \quad \in \quad [1, ..., \mathcal{G}]$
04. Number of child-parent dominations	
Type	Intermediate reward
Description	Returns the amount of parents dominated by their childs
Source	Retrieved from: [Huang et al. (2007)]
Mathematical formulation	$\text{Reward}_{gen} = \sum_{c=1}^{Offspring} D_{c,p_1} + D_{c,p_2} \quad D_{c,p} = \begin{cases} 1 & if \quad c \prec p \\ 0 & otherwise \end{cases} \quad \forall gen \quad \in [1, ..., \mathcal{G}]$
05. Sum of hypervolume indicator over entire generational process	
Type	Episodic reward
Description	Rewards all actions with the sum of all hypervolume indicator values obtained
Source	Inspired by: [Huang et al. (2021)]
Mathematical formulation	$\text{Reward}_{gen} = \sum_{g=0}^{[1,...,\mathcal{G}]} HV_g \quad \forall \quad gen \quad \in \quad [1, ..., \mathcal{G}]$

Table 4.2: Potential reward functions Deep Reinforcement Learning (DRL) agent

benefits the performance by improving the algorithmic stability during training. The Dueling DQN agent uses two neural networks, namely the value network and the target network. The value network is used to predict the value of being in a given state and the target network is considered to be the behavioral policy, dictating the actions taken by the agent. Training of the neural network is done on the value network, for which the weights are updated using batched samples of the replay buffer of past interactions. During training, the target network is kept constant. By only periodically replacing the target network weights with the value network weights, after every so-called replacement period, stability during training is attained. As the agent acts upon its target network, it remains a steady course of action for the time duration of one target network replacement period. This prevents the agent to immediately following previously obtained high rewards and thus falling into a positive feedback loop.

Both the value and target network have an identical architecture, as the weights of the former are periodically copied onto the latter. The architecture consists of two feed-forward, fully-connected layers. After these layers the network is split into a state value layer and advantage layer. These layers are taken through an aggregation layer, which results in a

prediction of the state-action values (Q-values). The split in state values and advantage values is created to simplify action selection. For action selection only the advantage values are needed, as they indicate how good a certain action is compared to all other possible actions. Simply choosing the action with the highest advantage value will result in the optimal action selected with regards to the current behavioral policy of the DQN agent. The neural architecture is visualized in Figure 4.6.

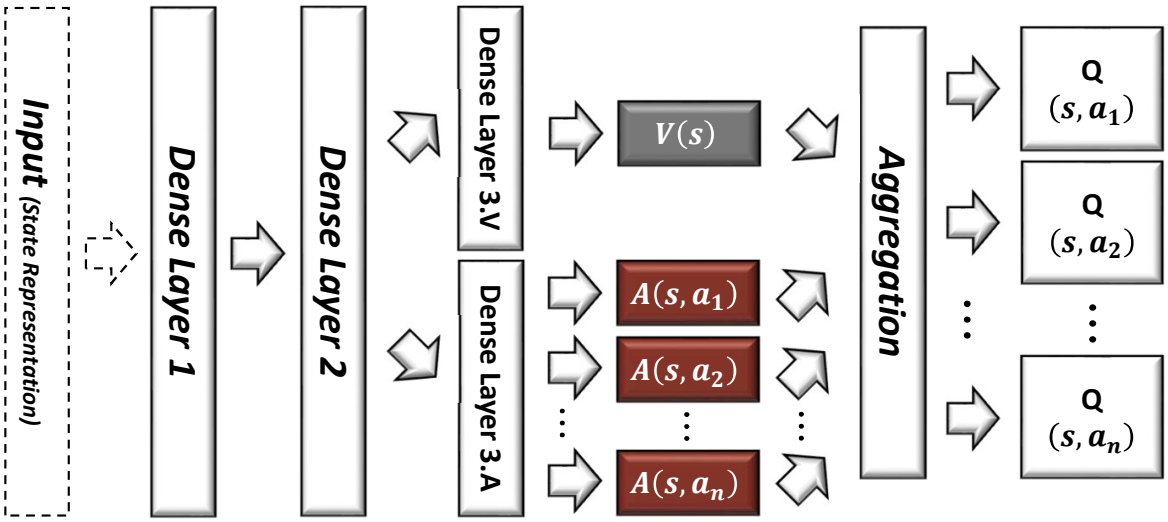


Figure 4.6: Neural architecture Dueling DQN agent

Displayed in Figure 4.6 the separation in state values and advantage values can be seen in the third hidden layer. The state value consists of a single value, as it predict the value of being in state s. The advantage layer, on the other hand, consists of as many values as there are actions to be selected. These values represent their relative utility, implying that for example advantage value $A(s, a_1)$ depicts how beneficial it is to select action a_1 in state s compared to all other actions $a_2 ... a_n$. The aggregation layer combines the value of the state and the advantage values into Q values, also called state-action values. For example, the Q-value $Q(s, a_1)$ describes the value of being in state s, then taking action a_1 and thereafter following the current behavioral policy of the agent.

4.4 Solution framework

The above mentioned solution methods are combined into a solution framework, which shows the relations between these methods. A visual representation of this framework can be found in Figure 4.7.

As can be seen in Figure 4.7, the framework consists of three main parts after initialization. The Discrete Event Simulation (DES) is used to evaluate the first population of randomly initialized individuals. The evaluated individuals are outputted towards the select parents step of the NSGA-III, which starts the actual optimization algorithm. The parents are used to generate a set of offspring individuals, which are then again taken through the DES for evaluation. The performances of the offspring population are thereafter, together

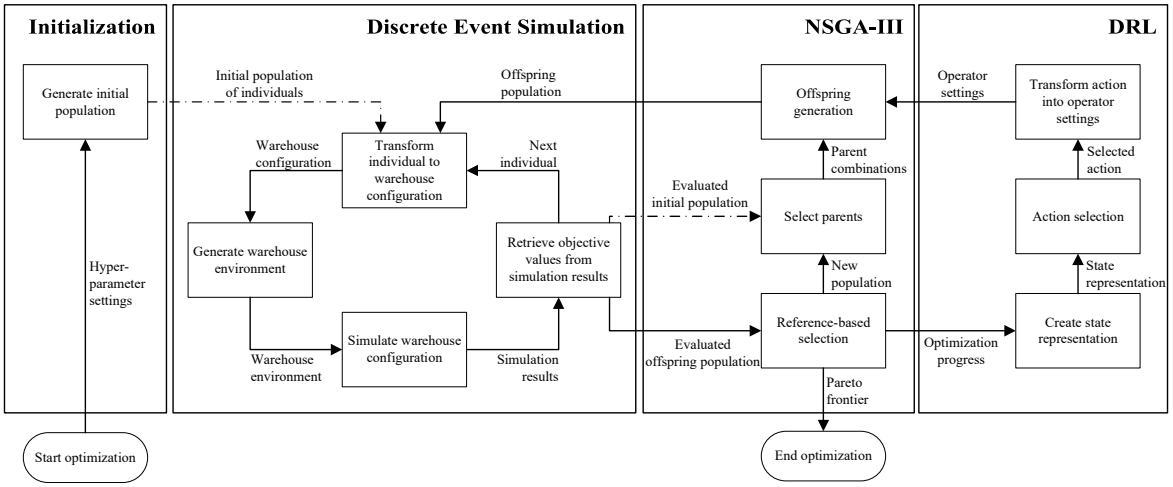


Figure 4.7: Solution framework

with the initial population, taken through the reference-based selection operator. The selection operator reduces the number of individuals again to the initial population size, creating a new population. After creation of this population, the optimization performance is send to the DRL agent. The agent creates a state representation and selects an action based in the interpretation of its environment. The selected action concerns operator settings for the offspring generation process. This process repeats until the termination criterion is reached, after which the final Pareto frontier is returned and the optimization process is ended.

Chapter 5

Experimental setup

To test the proposed solution methods, five consecutive experiments will be executed. The order in which these experiments are executed lends itself to utilize results obtained from previous experiments into the following experiments. These interrelations, together with a small summary of the experiments, is described in Table 5.1.

Exp.	Description	Algorithmic focus	Reason for execution	Uses results of experiment						
				1a	2a	2b	3a	4a	4b	4c
1	Benchmark NSGA-III	Optimization algorithm	**1a.** Analyze NSGA-III performance against NSGA-II and SMPSO.							
2	Hyperparameter optimization	Optimization algorithm	**2a.** Optimize the performance of NSGA-III	X						
			2b. Focus action space DRL agent on important parameters and ranges.	X						
3	Sensitivity analysis	Optimization algorithm	**3a.** Analyze robustness of the optimized NSGA-III algorithm.	X	X					
4	Learning DRL agent on DTLZ2 problem	Deep Reinforcement Learning	**4a.** Analyze performance of different reward functions.			X				
			4b. Evaluate learning capability of DRL agent.			X		X		
			4c. Evaluate learned policy of DRL agent.					X	X	
5	Evaluation of DRL on Warehouse Design and Control Problem	Deep Reinforcement Learning	**5a.** Benchmark DRL agent against optimized NSGA-III.	X	X	X		X	X	
			5b. Analyze robustness of DRL agent against optimized NSGA-III.	X	X	X	X	X	X	

Table 5.1: Interrelations experimental setup

The first three experiments all focus on the optimization model. First the decision to work with the NSGA-III is validated, by benchmarking its performance compared to the NSGA-II and the SMPSO algorithm. Then experiment 2 optimizes the hyperparameters of

the NSGA-III, after which the robustness of the optimized algorithm is tested in experiment 3.

The final two experiments focus on Deep Reinforcement Learning (DRL). First experiment 4 tests the different reward functions and analyzes the learning capability of the proposed implementation on the DTLZ2 benchmark problem. Thereafter the learned model will be placed on the problem central to this research, namely the Warehouse Design and Control problem. Besides benchmarking the performance against the optimized NSGA-III from experiment 2, also the robustness of the implementation is tested. By replicating the sensitivity analysis executed in experiment 3 the ability of the agent to maintain its performance in extreme scenarios is tested against the peformance of the optimized NSGA-III.

All Python code written and used can be found on Github, using the following link: https://github.com/RemcoCoppens/Master_Thesis_Code.

5.1 Experiment 1: Benchmark performance NSGA-III

First the performance of the Non-dominated Sorted Genetic Algorithm III (NSGA-III) is compared to the state-of-art algorithms mentioned alongside NSGA-III in literature. These algorithms concern its predecessor NSGA-II and Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO), which concerns a swarm intelligence algorithm. Literature shows inconclusive about which algorithm performs best, which falls in line with the "No free lunch" theorem [Wolpert and Macready (1997)]. This theorem roughly states that there exists no operator/algorithm that outperforms all other algorithms on all optimization problems.

These three algorithms will be run on the Python replication of the simulation model, as discussed in section 4.1.1. To compare performances the hypervolume indicator, calculated according to Equation 3.1, will be used. As explained in section 3.2.2, this metric summarizes the performance of the entire Pareto optimal set. The higher this metric, the more space is dominated by the front, the better the set of solutions. Tracking this value over consecutive episodes will enable evaluation of the Pareto front in terms of size, diversity and convergence rate.

Evaluation of all three algorithms will be done over five independent optimization runs, showing both the average performance and the standard deviation over these runs. The reason five independent runs are used has to do with the computational cost of executing a single optimization trajectory. Five runs are perceived minimally sufficient to draw conclusions on this stochastic process. Due to the memory overhead needed to run SMPSO, the population size used for optimization cannot consist of more than 20 individuals. Besides, due to the high high computation cost it is not possible to optimize the hyperparameters of all three algorithms. For this reason, to enable fair evaluation, all algorithm specific hyperparameters are left to the values mentioned in their original papers. All optimization runs consist of 200 generations, using 20 individuals this accumulates to 4000 warehouse configurations taking around 4.5 hours to be evaluated. Empirical analysis showed that 200 generations is sufficient to converge to some form of an approximation of the actual Pareto front. The hyperparameter settings of this experiment are summarized in Table 5.2.

	SMPSO	NSGA-II	NSGA-III
Number of generations	2000	2000	2000
Population size	20	20	20
Crossover probability	N/A	0.9	1.0
Crossover η	N/A	20	30
Mutation probability	N/A	1.0	1.0
Mutation η	N/A	20	20
Mutation indpb	N/A	0.01	0.01
Intertia weight	1	N/A	N/A
Local archive size	3	N/A	N/A

Table 5.2: Hyperparameter settings experiment 1

5.2 Experiment 2: Optimization of Hyperparameter

After proving superiority of the NSGA-III using the benchmark in experiment 1, its hyperparameter settings will be optimized. For optimization of the hyperparameters the framework of Optuna [Akiba et al. (2019)] will be utilized. The Optuna framework uses Bayesian Optimization, which builds a probability model of the objective function(s) and uses it to select the most promising hyperparameters to evaluate next. This is beneficial as the next hyperparameters are chosen based on previously obtained results, which sets it apart from classical methods like grid and random search.

The objective value used to evaluate different hyperparameter settings is focuses on the convergence performance of the algorithm, taking both convergence speed and solution quality into account. The designed metric consists of the overall sum of hypervolume indicator [Equation 3.1] attained at every generation. This way both fast convergence and higher quality solutions lead to the performance value being higher. To mitigate the effect of high computation time on the possible amount of algorithmic evaluations, it is decided to not run the entire 200 generations. Dependent on the NSGA-III performance of experiment 1 the number of generations is set to a minimum number, still allowing it to converge properly in most cases.

The Optuna algorithm got three values which it could adjust, namely the distribution parameter of both the crossover (η_{SBX}) and mutation (η_{PLM}) operator and the independent mutation probability ($indpb_{PLM}$). The allowed ranges for these parameters are summarized in Table 5.3. These ranges are set to allow for a wide enough range to optimize, without loosing the value of the underlying operator. To remain valid parameter settings for the distribution parameters (η), the upper bounds are set through empirically investigating different settings of the operators in an isolated setting on single valued individuals.

Hyperparameter	Type	Lower bound	Upper bound
Crossover distribution parameter (η_{SBX})	Int	0	100
Mutation distribution parameter (η_{PLM})	Int	0	100
Independent Mutation Probability ($indpb_{PLM}$)	Float	0.0	1.0

Table 5.3: Optuna hyperparameter ranges

Besides optimal hyperparameter settings for the NSGA-III operators, also the hyperparameter objective space will be extensively studied. Information regarding promising hyperparameter regions and the importance of features will guide the design of the action space for the Deep Reinforcement Learning agent.

5.3 Experiment 3: Sensitivity analysis

After hyperparameter optimization, the optimized model will be tested against different scenarios. These scenarios are designed to test the robustness of the optimization algorithm, showing the extend to which the algorithm remains its performance despite increased complexity of the underlying problem. Three scenarios are tested, which all focus on a potential increase in complexity for either one or more of the objective values used throughout this research.

1. **Increased amount and size of trucks arriving**
   Increasing the amount of truck arrivals and their size will imply increasing the overall workload for internal logistics. The arrival of these trucks is uniformly distributed over different time slots over a day, showing no concentration of excessive workload on a specific part of the day. This increase will occur on both inbound and outbound trucks, as an equilibrium is needed to keep the warehouse inventory stable. This analysis focuses on overall workload, having an effect on all aspects of the optimization problem. These aspects concern the Product Placement Algorithm, the number of resources and the layout design of the warehouse. An expected result of this increased workload is congestion around the docks and consolidation areas, due to the increased loading and deloading time. It is expected that all objective values will increase, with the most significant effect on *tardiness of outbound trucks*.

2. **Inconsistency in truck arrivals**
   As opposed to an overall increase of pressure on internal logistics, this analysis looks into an irregular distribution of the daily workload. The overall workload will not be adapted and will stay identical to the original implementation. The distribution of this workload will, however, be concentrated on the first half of the day. Creating a scenario in which 3/4 of the trucks arrive in the first half of the day and the remaining 1/4 arrives in the second half, as opposed to the uniform distribution of workload in the original scenario. The complexity of irregular workload will result in presumably high resource cost, despite a high level of resources staying idle in the second half of the day. The model will have to decide to either place too much resources or too little resources, with no possibility to strike a balance between the two. It is expected that this trade-off will increase volatility of the performance of the algorithm. This scenario increases the complexity of the underlying Resource Allocation Problem (RAP), looking into the sensitivity towards the second objective value *Resource cost*.

3. **Increased product portfolio**
   The final analysis looks into increasing the difficulty of the underlying Product Allocation Problem (PAP) by increasing the diversity of the product portfolio. The number of possible products is doubled, increasing the product portfolio size from 1981 to 3962 distinct products. By duplicating the occurrence probability of these product, the diversity of products both within inbound trucks and requested by outbound trucks is

doubled. The increased complexity of this scenario focuses on the sensitivity towards the third and final objective, namely the amount of *unplaceable products*.

5.4 Experiment 4: Learning DRL agent on DTLZ2 problem

Despite the performance gain obtained through replication of the simulation in Python, still the simulation model is too slow to learn a DRL agent in a considerable amount of time. Additionally, different agent configuration with regard to reward functions and hyperparameter settings need to be tested, which asks for a faster alternative. For this reason an agent will be trained on another problem, which is more simple and faster to evaluate individuals. This problem concerns the problem suite designed by Deb, Thiele, Laumanns and Zitzler, named DTLZ2. DTLZ2 is a mathematical, multi-objective optimization problem. The reason this problem is chosen is that it allows for adaptation of the number of objectives and decision variables. Minimizing the difference between the DTLZ2 problem and the Warehouse Design and Control Problem central to this research.

As can be seen in Table 5.1, the reasons to execute this experiment is three fold. First the proposed reward functions, described in Table 4.2, are tested. To evaluate the learning capabilities of differently configured agents, the volatility of the learning trajectory needs to be taken into account. For this reason all reward functions are tested over five independent runs, showing their average score and standard deviation. The performance score of an agent is calculated using a discounted sum over the performance of the generational processes. The performance of a single process is identically calculated as in hyperparameter optimization, namely by the sum of all hypervolume indicator values [Equation 3.1] obtained during all generations of the optimization process. A mathematical formulation of this performance score can be described as follows:

$$Performance\ agent = \sum_{e=0}^{Episodes} \gamma^e \cdot \sum_{g=0}^{Generations} HV_{g,e} \tag{5.1}$$

By discounting these values over all training episodes, both the needed training time and the quality of the learned policy are evaluated. During reward function evaluation the agent is allowed to learn for 2000 episodes, in which every episode represents an entire run of the optimization algorithm for 200 generations. The hyperparameter settings for this analysis are described in Table 5.4.

The other two reasons for execution of this experiment concern evaluation of the learning capabilities of the agent and an analysis of the learned optimal policy. First a new agent is trained using the best performing reward function, showing the highest performance score calculated using Equation 5.1. For the learning trajectory the number of episodes is doubled, from 2000 to 4000 episodes of entire optimization processes of 200 generations. The intended purpose of this is to increase the change of the agent converging to its optimal policy. The increased number of episodes also decreases the epsilon decay factor to 0.99825, extending the period in which the agent explores the environment using random actions. The performance of the trained agent on the DTLZ2 training environment is compared to a randomly initialized

Hyperparameter	Value
NN layer size 1	32
NN layer size 2	64
Number of episodes	2000
Learning rate	1e-4
Gamma	0.99
Batch size	32
Epsilon start	1.0
Epsilon decay	Exponential
Epsilon decay factor	0.9975
Epsilon end	0.1
Replacement period	20000 (Every 100 optimization runs)

Table 5.4: Hyperparameter settings DRL Agent

agent and the optimized NSGA-III algorithm without Adaptive Operator Selection (AOS).

Finally, the learned optimal policy is analyzed. This analysis is conducted on a data set gathered by running 500 independent optimization trajectories, in which the agent acts greedily upon its learned policy. First the fraction of actions selected in different generations will be analyzed, to gain insight into policy differences in consecutive stages of the evolutionary process. To further understand the decisions made by the agent a decision tree will be fitted using the gathered data. A decision tree is able to visualize a high-level overview of the decisions made by the agent, indicating values determining the selection of certain actions.

5.5 Experiment 5: Evaluating agent performance on the Warehouse Design and Control Problem

The fifth and final experiment concerns the application of the Deep Reinforcement Learning (DRL) agent on the actual Warehouse Design and Control Problem (WDCP). The DRL agent is not applied on the problem directly but instead on the optimization algorithm, more specifically the NSGA-III. As the NSGA-III handles the optimization problem as a black box, it is expected that the agent can simply be copied from one problem to the next. The main difference between the DTLZ2 benchmark problem and the WDCP is the complexity of the objective space. DTLZ2 concerns a relatively simple, convex objective space, where the WDCP has a complex, non-convex objective space.

First the performance of the agent will be tested on the WDCP central to this research. This test will compare its performance with both a random agent and the optimized NSGA-III algorithm, obtained in experiment 2. The random agent is included to validate that the performance difference can be assigned to the learning policy, and it not being a byproduct of random Adaptive Operator Selection (AOS). To include variability, all model performances show and average and their standard deviation over five independent optimization runs.

Finally the robustness of the agent is tested, for which the sensitivity analyses of experiment 3 are repeated. By comparing the performance of the agent and the optimized

NSGA-III algorithm, potential differences in model robustness can be measured. For this analysis no random agent will be included. The main reason for this is time, as running these optimization trajectories has a high computational cost. As variability is present, every model performance is an aggregate over five independent runs, showing the average values and their standard deviations.

Chapter 6

Results

This chapter will elaborate on the results obtained from execution of the experiments discussed in chapter 5. The results are discussed in the order of execution of the experiments, where in most cases the results of the previous experiments are used into the consecutive experiments. First the NSGA-III is benchmarked against the NSGA-II and the SMPSO algorithm in section 6.1, after which its hyperparameters are optimized in section 6.2. The optimized NSGA-III model is taken through several sensitivity analyses in section 6.3. After which Deep Reinforcement Learning is trained in section 6.4 and evaluated on the problem of this thesis in section 6.5.

6.1 Experiment 1: Benchmark performance NSGA-III

First the validity of the decision to work with the NSGA-III is tested, benchmarking its performance against other well performing multi-objective optimization models discussed in section 3.2.2. These models concern its predecessor, the NSGA-II, and the Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) algorithm.

As a result of the high computation time of a single optimization run, the evaluation is restricted to use solemnly the original hyperparameter settings as mentioned in their original papers [Deb and Jain (2013); Deb et al. (2000); Nebro et al. (2009)]. This computation time restriction also makes a thorough evaluation of the variability of these algorithms impracticable. Still these models are evaluated using five independent optimization runs, sharing an identical random seed for the underlying warehouse simulation. Execution of multiple optimization runs allows for some indication regarding the variability of the optimization models, visualized using error bars, shown in the plotted performances in Figure 6.1. The thick line represents the average performance over all five independent runs, where the areas display the volatility between the runs.

Looking at the model performances in Figure 6.1 it can be seen that both the NSGA-II and the SMPSO algorithm have a fairly slow optimization trajectory. For the NSGA-II, the main reason for this slow optimization is its loss of population diversity. This loss happens around generation 27 and is most probably an effect of the small population size used. After this loss of diversity the algorithm is unable to improve its performance any further. Crossover does not yield severe enough difference in their offspring to include enough new genetic

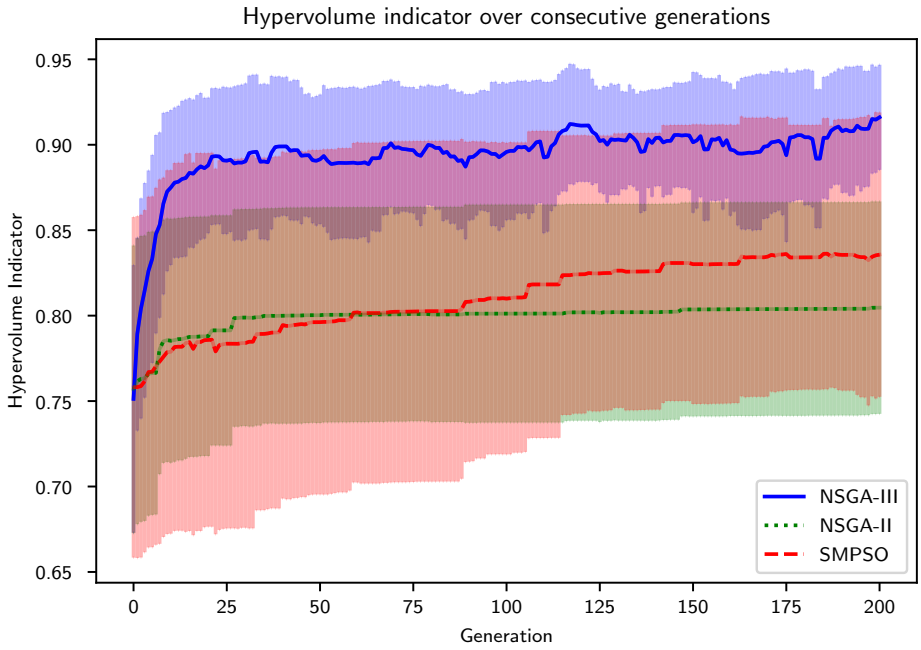


Figure 6.1: Benchmark NSGA-III against NSGA-II and SMPSO on the WDCP

material to break out of this converged population.

SMPSO, on the other hand, keeps a steady pace of incremental improvement. Due to its inherent design of gradually moving through the search space, the SMPSO algorithm is less prone to loss of population diversity. The slow improvement of SMPSO can have multiple explanations, of which local archive size is most probably affecting the algorithm the most. The local archive holds the best found solutions for all individual particles and is set to hold a maximum of three Pareto-optimal solutions. Holding less solutions, these archives contain less well-defined local Pareto fronts. Decreasing the accuracy of local attraction, being one of the forces deciding the direction and velocity of a particle flying through the decision space.

Conclusively it is clear that NSGA-III outperforms both the NSGA-II and the SMPSO algorithm. The highest hypervolume indicator attained by NSGA-III is 0.92, which is significantly higher than the 0.80 and 0.84 obtained by the NSGA-II and the SMPSO algorithm. This difference shows that the NSGA-III is able to obtain a better solution quality. On the other hand, looking at convergence speed, the NSGA-III is also able to outperform both other algorithms. The convergence speed is calculated looking at the area under the curve, which shows a significant difference. The total area under the curve is calculated by taking the sum over all obtained hypervolume indicator values. This metric accumulates to 180 for the NSGA-III, 161 for the NSGA-II and 163 for the SMPSO algorithm. Besides performance, both the NSGA-II and NSGA-III show faster computation than the SMPSO algorithm. All computation of a single generation for NSGA-II and NSGA-III take around 56 and 55 seconds respectively, as opposed to the 64 seconds for the SMPSO algorithm. The difference in com-

putation time lies in the different algorithmic design, in which SMPSO needs a lot more computations to get from one population to the next. These finding validates the decision to use NSGA-III as the algorithm of choice for further execution of this research project.

6.2 Experiment 2: Hyperparameter optimization

After the decision to use NSGA-III as the algorithm of choice for this research project, its hyperparameters need to be tuned. Using the Python library Optuna [Akiba et al. (2019)], Bayesian optimization is applied on the previously discussed parameters and value ranges. The main purpose of hyperparameter optimization is to gain convergence speed, which only requests insight into the beginning of the optimization process. For this reason, concluding from the performance of the NSGA-III shown in Figure 6.1, it is perceived sufficient to only run 50 generations per hyperparameter settings. In total 30 trials consisting of 50 generations using 20 individuals are executed, surmounting to a total computation time of around 33 hours. These trials are summarized in a contour plot, visualized in Figure 6.2.

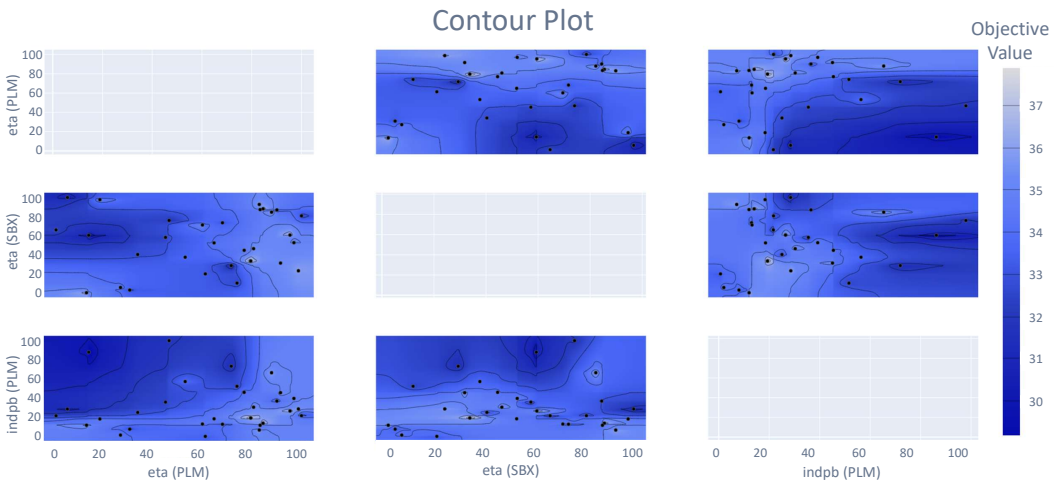


Figure 6.2: NSGA-III hyperparameter optimization - Contour plot

The contour plot shows all combinations of the three hyperparameters, namely the distribution parameter for crossover (*eta* (SBX)) and mutation (*eta* (PLM)) and the independent mutation probability (*indpb* (PLM)). The black dots represent trials, proposed by the Bayesian optimization algorithm and thereafter evalauted for their performance. The lighter areas show higher objective values, indicating higher quality and/or speed of generational convergence.

The performance of different values for the independent mutation probability (*indpb*) show a clear range of values, between 0(%) and 20(%), returning higher objective values. The distribution parameter for mutation (*eta* (PLM)) and crossover (*eta* (SBX)), however, show a less well-defined range of optimal values. The plot showing both these distribution parameters indicates that the mutation has a more significant effect on performance, where crossover shows a more uniform distribution of values returning near-equal performance. This uniform spread of values might indicate a lower feature importance, as opposed to the other values

which clearly show different results for different values. Further evaluation of this feature importance is executed by extracting the observed importance of all parameters by the Bayesian optimization algorithm, which are visualized in Figure 6.3.

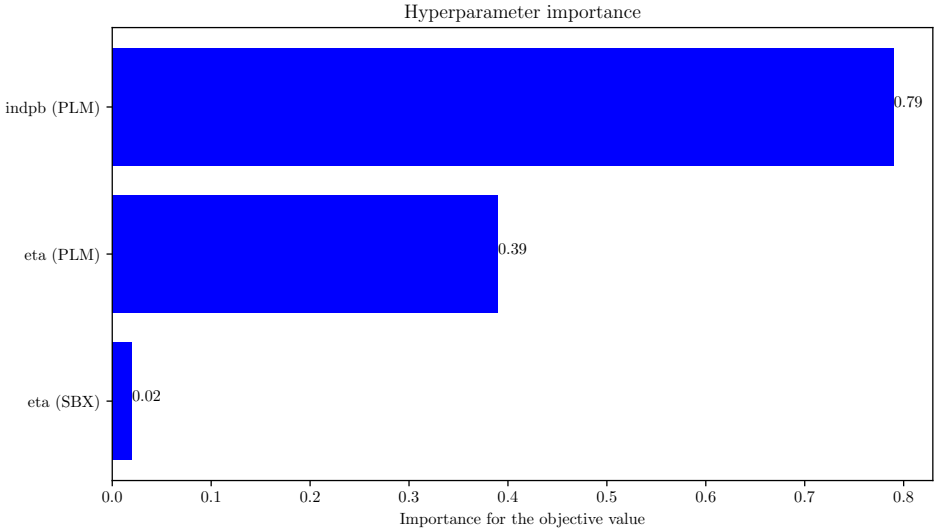


Figure 6.3: NSGA-III hyperparameter optimization - Feature importance

Concluding from Figure 6.3 it can clearly be seen that the hypothesis concerning the crossover distribution parameter being of less importance than the other parameters is valid. A potential reason for this might be due to the non-convexity of the WDCP problem. As the local neighborhood of well-performing individuals does not guarantee good performance, exploitation becomes less beneficial. For this reason it is decided to remove the crossover operator from the action space in the implementation of Deep Reinforcement Learning (DRL). Leaving the agent to only adjust the mutation operator. The allowed ranges are deduced from the contour plot, resulting in the independent probability to be varied between 0.0 and 0.2 and the distribution parameter on the full range between 0 and 100. The crossover distribution parameter (eta (SBX)) is excluded from the action space of the DRL agent and is set to 34, which is the optimal value found by Optuna. This value is remarkably close to the value found in the original implementation of NSGA-III, being 30 [Deb and Jain (2013)].

6.3 Experiment 3: Sensitivity Analysis

Using the optimized hyperparameter settings obtained in the previous experiment, a sensitivity analysis is executed to test the robustness of the optimized NSGA-III. This analysis consists of three sub-analyses, all focusing on putting tension specifically on one of the three objective values. Increasing the amount and size of truck arrivals puts tension on truck tardiness [section 6.3.1], inconsistent truck arrivals on resource cost [section 6.3.2] and finally an increased product portfolio on the number of unplaceable products [section 6.3.3].

All performances are shown in two plots, one showing the hypervolume indicator over consecutive generations and one showing the average Pareto front performance on all three objective values separately. These values are obtained from five independent optimization runs, showing both the average and standard devation for the hypervolume indicator plots. For the average Pareto front performance only the average performance is shown, as inclusion of standard deviations would obscure the understandability of the plots.

6.3.1 Increased amount and size of trucks

Increasing both the size and amount of incoming trucks increases the complexity for the model to balance between less resources while maintaining a low outbound truck tardiness. The decision to remove a single resource from the available fleet, independent of resource type, will result in a more severe reaction on the tardiness of outbound trucks. This effect can clearly be seen in the hypervolume indicator, visualized in Figure 6.4.

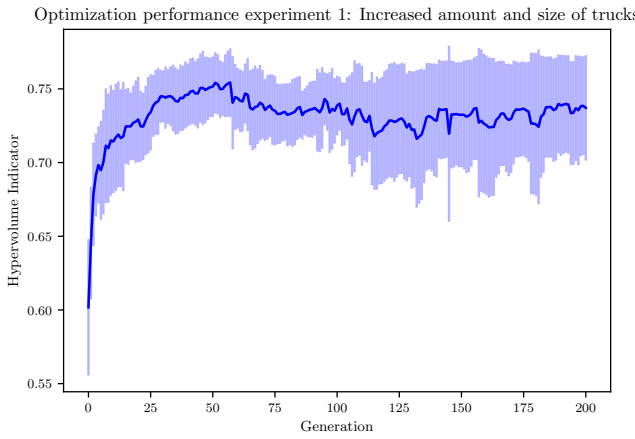


Figure 6.4: Increased amount and size of trucks - Hypervolume indicator

The model still shows the ability to converge to a set of Pareto-optimal solutions, albeit on a relatively low hypervolume indicator. The reason for this can be explained by the way this metric is calculated. More specifically with the normalization applied to enable calculation of the hypervolume indicator. The objective values are (Min-Max) normalized towards their hypothesized best attainable values, ensuring the value to fall within the allowed ranges through value clipping. Due to the previously discussed increased effect of decreasing resources, values near the optimal truck tardiness will become dominated quickly. This will remove them from the Pareto set, excluding them from the hypervolume indicator calculation. This effect can also be seen in the average population performance, as visualized in Figure 6.5.

The plot showing the average tardiness of outbound trucks shows an increase in the value over consecutive generations. This clearly indicates the effect of domination of solutions performing fairly well on this metric. Explaining the relatively poor performance of the hypervolume indicator in Figure 6.4.

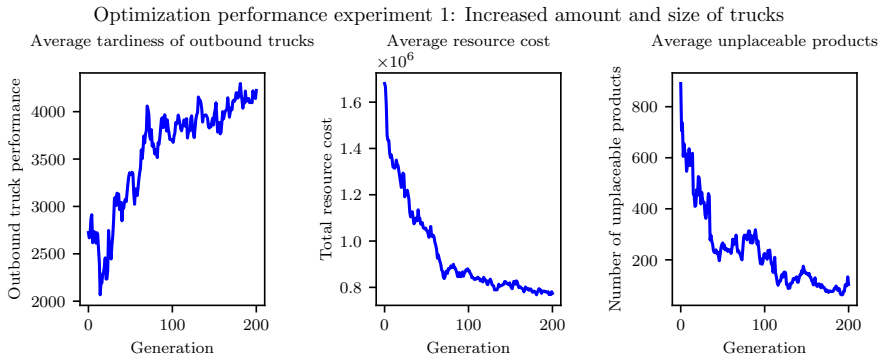


Figure 6.5: Increased amount and size of trucks - Average Pareto set performance

6.3.2 Inconsistent arrival of trucks

Removing the consistency with which trucks arrive will complicate the Resource Allocation Problem (RAP), discussed in section 3.1.2. The RAP being a part of the Warehouse Design and Control Problem (WDCP) central to this research. In the tested scenario 3/4 of the amount of trucks arrive in the first half of the day, where the remaining 1/4 of trucks arrive in the second half. Doing this remains the overall workload of a single day, setting this analysis apart from the first scenario tested. The performance of this scenario is visualized in Figure 6.6.

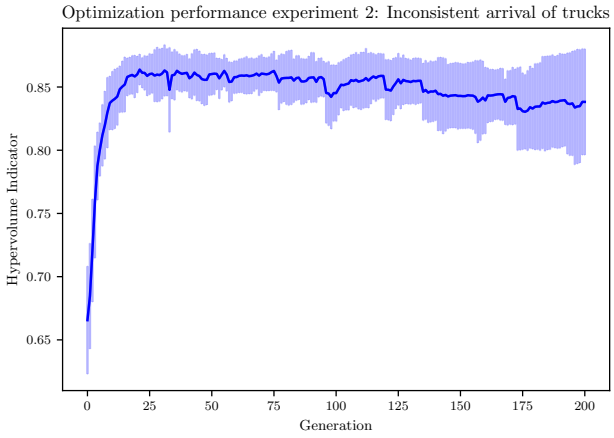


Figure 6.6: Inconsistent truck arrivals - Hypervolume indicator

Looking at the hypervolume indicator over consecutive generations in Figure 6.6, it can be seen that the algorithm is able to converge to a decent approximation of the actual Pareto frontier. Interesting to note is that the variability between the five distinct optimization runs shows to increase over consecutive generations. A possible explanation for this may be the different ways the algorithm can handle the trade-off between resource cost and their idle time. Higher amounts of resources will allow for better performance during higher workload,

but will result in resource cost incremented for resources staying idle. The opposite is also true, lower amounts of resources will lower the resource cost and idle time, but will result in poorer performance during higher workload.

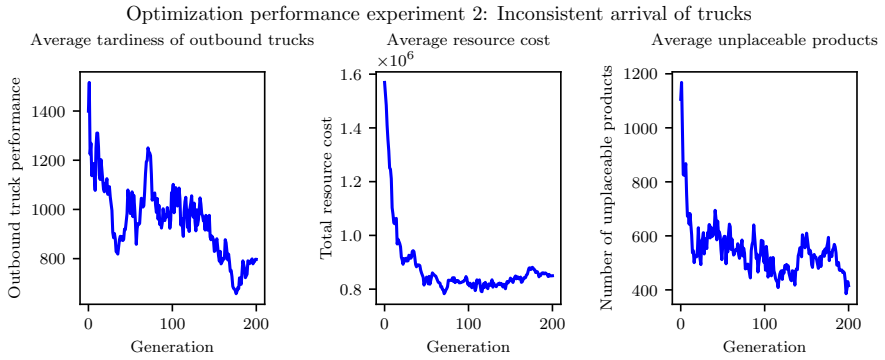


Figure 6.7: Inconsistent truck arrivals - Average Pareto set performance

This effect can also be seen in the average population performance, visualized in Figure 6.6. The average tardiness of outbound trucks shows a high volatility, all within a relatively high range of values. The spikes in this graph indicate the domination of well-performing individuals on this metric, which are removed from the population. An increased population size, instead of the currently used size of 20, might mitigate this effect. Increasing the population size will increase the allowed size of the Pareto-optimal set, leaving room for a wider variety of non-dominated solutions after the selection operator of the NSGA-III. However, increasing the population size will significantly increase computation time. As almost all computation time is dedicated to the evaluations of individuals it is decided to not increase the size of the population.

6.3.3 Increased product portfolio

The final analysis looks into the effect of doubling the product portfolio. Increasing product diversity complicates the Product Allocation Problem (PAP), discussed in section 3.1.3. The PAP is also part of the problems underlying the WDCP central to this research. The creation of in- and outbound truck orders is based on a set of probabilities, defining for all individual products the chance of belonging to a given truck order. The probability assignment over these products is replicated, recomputing the probabilities to again represent a fair set of probabilities. This ensures that the diversity of products is doubled, following identical distributions as the original set of products. The convergence performance of the NSGA-III algorithm in this scenario is shown in Figure 6.8.

The hypervolume indicator over consecutive generations, shown in Figure 6.8, indicates that the performance is relatively unaffected by the environmental adjustment. It does, however, show high volatility until the 75 generation. A possible explanation for this could be found in the procedure of selecting product based on a set of probability distributions. Resulting in some simulation runs showing a higher amount of different products as opposed to order consisting of a lot of similar products. To further evaluate the sensitivity of the model

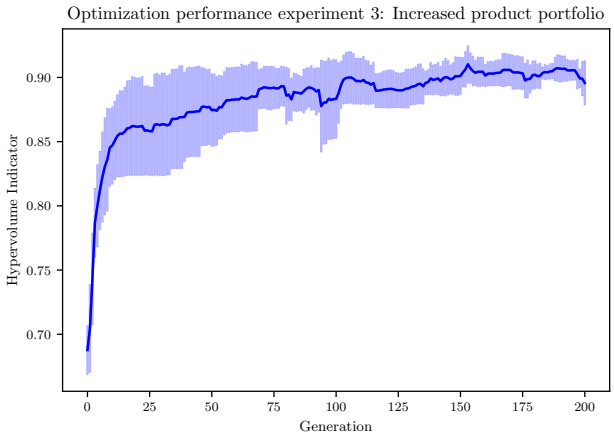


Figure 6.8: Increased product portfolio - Hypervolume indicator

towards this environmental adjustment, also the average population performances are plotted in Figure 6.9.

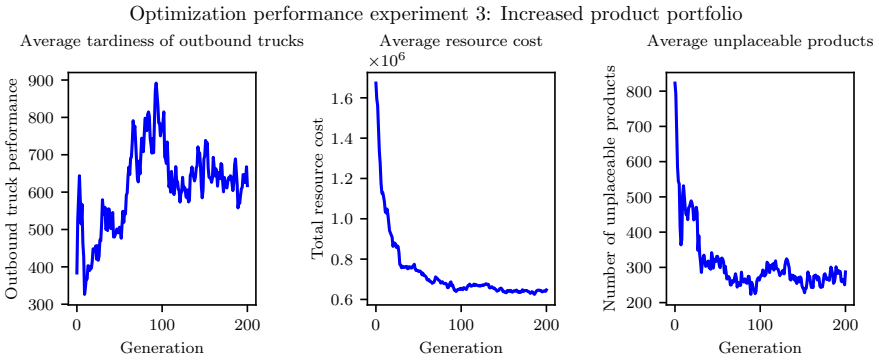


Figure 6.9: Increased product portfolio - Average Pareto set performance

Increasing the size of the product portfolio was expected to have a significant effect on unplaceable products. This expectation shows to be false, as indicated by the stable minimization of this metric throughout the population over consecutive generations, as seen in Figure 6.9. The performance attained, despite the larger product portfolio, might indicate the strength of the Product Placement Algorithm, as described in algorithm 1. The increased variety of different products results in the inventory of individual products to be lower. As the rule for placement in back-to-back storage includes the current inventory being lower than a set value, the probability of placement in back-to-back increases. This is beneficial in this scenario, as back-to-back storage places all products in their own storage location. The model remaining its performance is a clear indication that the design of the PPA is robust, withstanding even extreme conditions.

The increasing trajectory of the average truck tardiness is also remarkable, as this scenario was not expected to have any effect on this metric. Quite possibly this might be due to products being stored more spread around the warehouse, increasing the difficulty of correctly docking a truck through the Truck Docking Algorithm (TDA), as explained in algorithm 2. This will increase the needed travel time to retrieve products from storage to the consolidation area, which will affect the tardiness of outbound trucks.

6.4 Experiment 4: Deep Reinforcement Learning on test problem

Due to the time complexity of evaluating a single warehouse configuration, the Deep Reinforcement Learning agent will not be learned on the Warehouse Design and Control Problem (WDCP) central to this research. The learning problem concerns the DTLZ2 benchmark, as implemented in the Pymoo Python package, which concerns a multi-objective optimization problem. DTLZ2 allows for adaptation of the amount of decision variables and objective values, which enables increasing the similarity compared to the WDCP. Doing this minimize the differences in problem structure to impede the agent from learning a successful policy for the WDCP.

6.4.1 Testing reward functions

As mentioned in section 4.3.3, the selection of a reward function is nontrivial. Mainly as the objective of the DRL agent is different than the objective of the optimization model on the WDCP. For this reason five different reward functions are tested, all using similar hyperparameter settings as discussed in section 5.4. The performance is evaluated using the formula:

$$Performance\ agent = \sum_{e=0}^{Episodes} \gamma^e * \sum_{g=0}^{Generations} HV_{g,e} \tag{6.1}$$

For each reward function five independent learning trajectories are run, of which the averages and standard deviations are shown in Table 6.1. The discount factor (γ) used is set to 0.99.

Reward index	Reward name	Reward type	Mean performance	Standard deviation performance
01.	Binary HV	Intermediate	12.537	18.26
02.	Continuous HV	Intermediate	12.469	54.24
03.	Raw HV	Intermediate	12.556	24.53
04.	Number of child-parent dominations	Intermediate	12.560	67.11
05.	Sum of HV over all generations	Episodic	12.584	49.63

Table 6.1: Performance different reward functions

Concluding from Table 6.1 the episodic reward shows the best average performance. The differences are marginal, which might be an effect of the decaying epsilon value throughout training. Due to discounting of the obtained hypervolume indicator, there is a larger emphasis on earlier episodes. These earlier episodes are also affected by a higher number of random actions, due to the exponentially decaying epsilon. As this is identical for all trajectories

independent of reward function used, and all evaluations are run five independent times, the comparison is still perceived valid.

Regarding standard deviation there are significant differences between the different reward functions. Binary hypervolume indicator shows the lowest standard deviation, as it concerns a metric measured over the entire population. The opposite is true for the number of child parent dominations. Calculation of this metric concerns all offspring individuals separately, allowing for greater difference between different generations and thus resulting in the highest standard deviation. The standard deviation of the episodic reward, however, also shows relatively high. A possible explanation might be that episodic reward requires more episodes to distinguish good from bad actions. As all actions within an episode are given an identical reward, some bad actions will be praised and some good actions will be punished. By running more episodes this effect will decrease, as probabilistically good actions are more apparent in episodes obtaining a high reward and bad actions more apparent in low reward episodes. Extending the learning time from 2000 to 4000 episodes will most probably result in a lower standard deviation and thus a clear preference for the episodic reward. For this reason the actual DRL agent will be learned with an episodic reward function, mathematically formulated in Equation 6.2.

$$Reward_{gen} = \sum_{g=0}^{Generations} HV_g \qquad \forall \quad gen \in Generations \tag{6.2}$$

6.4.2 Performance comparison

Concluding from the comparison of reward functions, the episodic sum of hypervolume indicators shows superior performance. To gain insight in the actual performance of the agent, it is compared against a random policy and the optimized NSGA-III without AOS. The average performances of these different models are visualized in Figure 6.10 below. It is decided to show only the first half of the 200 generations executed. These generations emphasize the difference in convergence speed and quality, as later generations show a fairly linear line to convergence.

Concluding from Figure 6.10 the learned agent outperforms both other models, indicating that the agent is able to successfully learn a policy. The difference between the optimized NSGA-III and the agent induced NSGA-III is significant, showing the added value of AOS. However, the difference between the trained and random agent is of a smaller magnitude, showing the average performance of the former to perform on the upside of the error bars of the latter. A possible explanation can be that this is the effect of working with a small population and the inherent design of the NSGA-III algorithm. Under the expectation that the random agent will decide uniformly which action to take, 50% of its actions show an inclination towards more radical mutation. This unguided probability for mutation might help the algorithm the make significant leaps through the objective space, which could explain the good performance of the random agent. Still, the guided actions of the trained agent show superior convergence speed. Taking contextual information into account allows the agent to decide on severity of mutation more effectively. The benefit of the learned agent over the

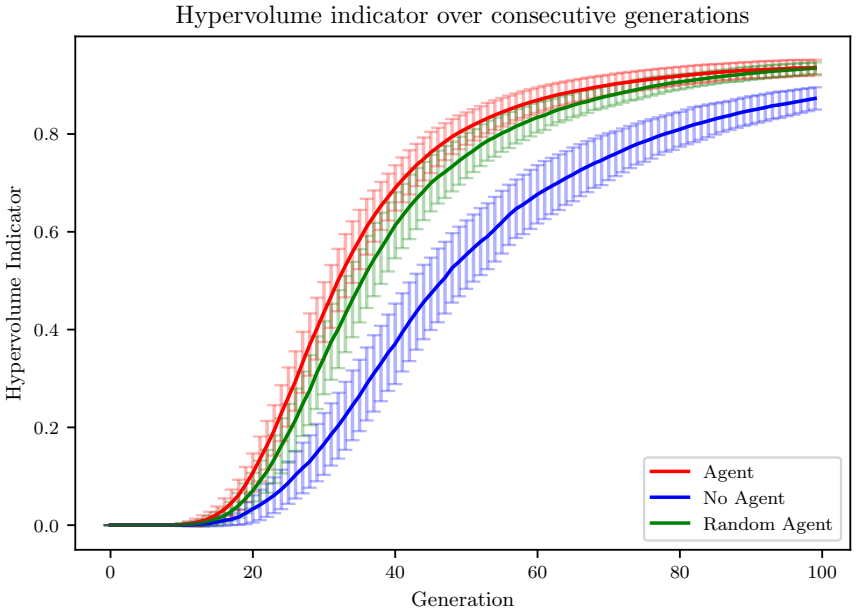


Figure 6.10: Performance benchmark DRL agent on DTLZ2

random agent concerns convergence speed, in which the former needs less evaluations to reach a well-performing Pareto front than the latter.

6.4.3 Policy evaluation

To both gain insight into the behavioral policy of the agent and to validate the radical mutation hypothesis for the performance of the random agent, a policy evaluation is executed. The evaluation consists of monitoring the behavior of the agent over 500 independent optimization runs, deciding upon the parametric values of the mutation operator based on the learned policy. The values allowed for the mutation operator are taken from the ranges retrieved from the Bayesian hyperparameter optimization, as discussed in section 6.2. During these runs the agent is only used to request actions, meaning that no adjustments are made to the underlying weights of the neural network. By taking the fractions of the amount of times a single action is chosen in a given generation, insight is gained into the behavioral policy of the agent over time. A visualization of these fractions over consecutive generation is shown in Figure 6.11.

Interesting to see is that the behavioral policy of the learned agent shows a higher preference towards explorative behavior in later generations. This finding contradicts the intuitive sense that the agent first diverges and then converges. An explanation can be found in the structural design of the NSGA-III algorithm. As the DRL induced algorithm starts to converge around generation 50 it gets more difficult to create offspring which dominates its parents. Especially in the neighborhood of the solution comprising the Pareto optimal set. Explorative behavior is beneficial in this scenario, as it allows to explore new regions of the

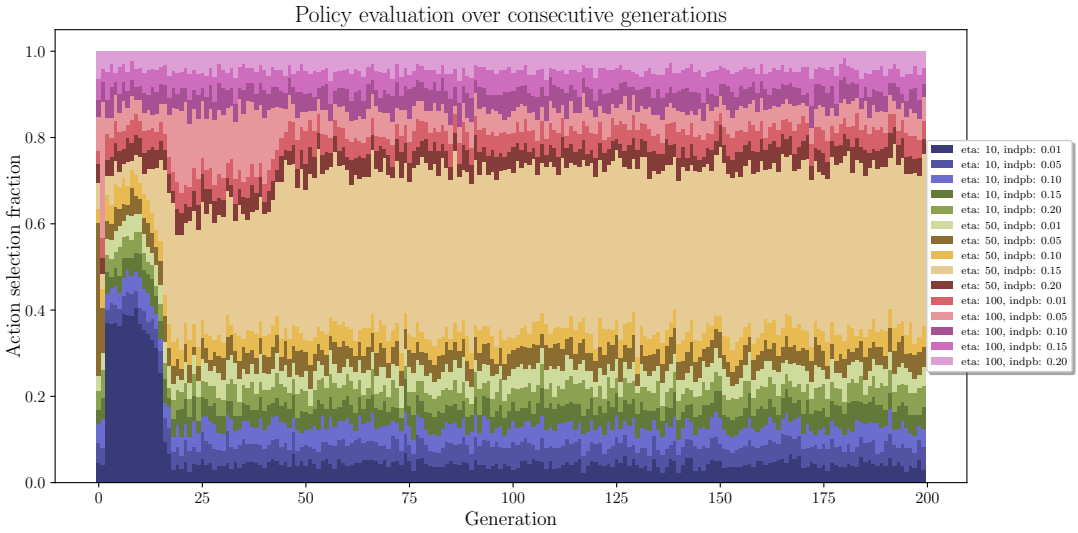


Figure 6.11: Policy visualization for the DTLZ2 benchmark problem over consecutive generations

objective space. Through the selection operator of the NSGA-III, less performing individuals will be eliminated. This prevents the algorithm from losing its performance, and thus mitigates the downside of more severe exploration. The dynamics of the selection operator of the NSGA-III might also explain the performance of the random agent, as it has a 50% chance of choosing for behavior inclined to more radical mutation. To gain more insight into the behavior of the agent, the average state representation values of the interactions underlying Figure 6.11 are plotted in Figure 6.12.

Apparent is the spike in the stagnation counter during the first 20 generations. This spike is directly related to the region inclined to high exploitation, as seen in Figure 6.11. A possible explanation for this spike can be due to normalization for the hypervolume indicator calculation. During normalization the values are clipped, implying values above an upper bound or below a lower bound are set to the upper and lower bound values respectively. For the DTLZ2 benchmark these values are set relatively strict, resulting in often setting values to the upper bound. If this happens for all objective values this results in a hypervolume indicator value of 0, which can also be seen in the hypervolume indicator trajectory shown in Figure 6.10. During the generations it takes the algorithm to minimize the values below the upper bound no improvements are made, resulting in an increased stagnation counter.

Further evaluation of the behavioral policy can be conducted through fitting a decision tree on the set of interactions. The data set used concerns five million interactions of the learned agent with the DTLZ2 environment. To allow for interpretability of the tree, the action space of the agent is categorized in either exploration or exploitation. This categorization is summarized in Table 6.2.

Before training the decision tree on the data set, preprocessing is applied. In this process

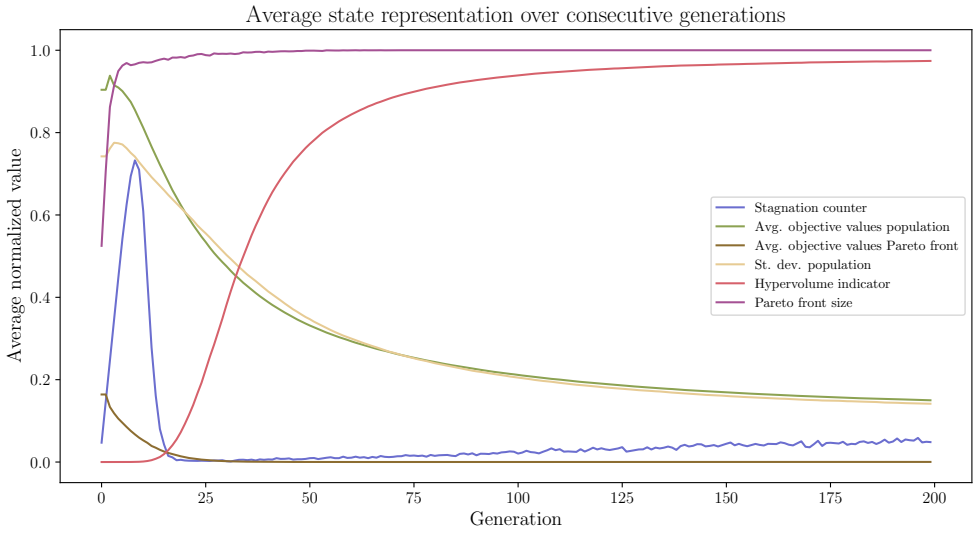


Figure 6.12: Visualization of state representation values for the DTLZ2 over consecutive generations

Exploit		**Explore**	
eta (PLM)	**indpb (PLM)**	**eta (PLM)**	**indpb (PLM)**
10	0.01	50	0.10
10	0.05	50	0.15
10	0.10	50	0.20
10	0.15	100	0.05
10	0.20	100	0.10
50	0.01	100	0.15
50	0.05	100	0.20
100	0.01		

Table 6.2: Categorization action space DRL agent

the data set is balanced and normalized, which steps are configured in a data pipeline to prevent data leakage and thus enable fair evaluation of the performance. The performance is measured through 10-fold, stratified cross-validation and is evaluated for trees varying from a depth of 2 to a depth of 10. Eventually, it is decided to work with a tree of depth 4, as to maintain a sense of interpretability. This tree attained a result of 65% accuracy and is visualized in Figure 6.13.

Analyzing this tree it can be seen that most predictions follow the light blue trajectory, resulting in exploratory behavior. Interesting to see is that a low hypervolume indicator often leads to exploitation, whereas a higher current performance leads to exploration. This finding shows again to be counter intuitive, as you expect a desire to exploit well performing individuals and explore when the current population is not performing well. Concluding from the decision tree the agent does not pay much attention towards the current generation, which is only present on the far left of both second-level sub trees. The hypervolume indicator and Pareto size are, however, most often used. This difference in perceived importance can be

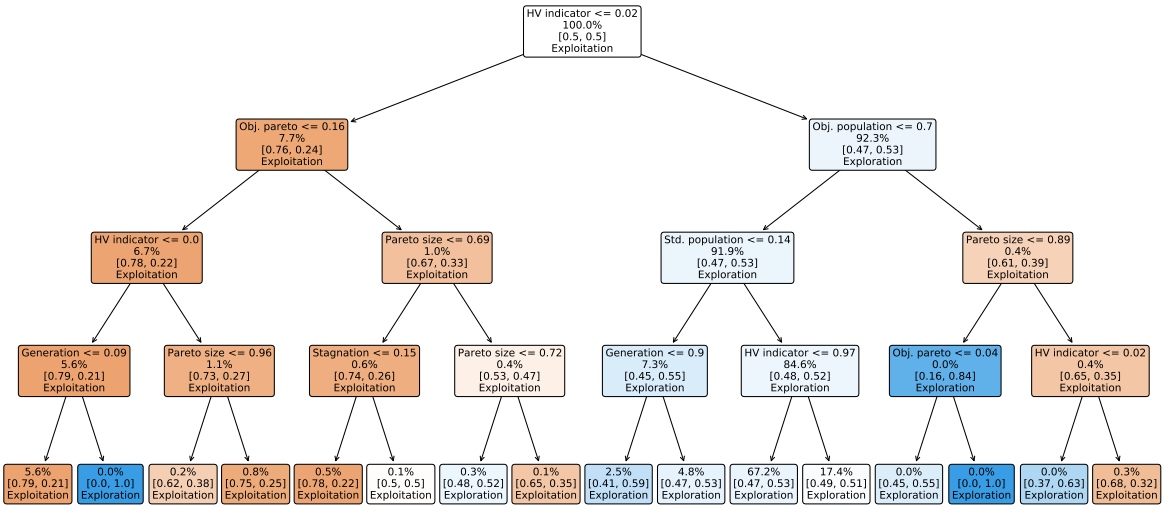


Figure 6.13: Decision Tree showing high level explanation of behavioral policy

explained by the correlation between the former and the latter. As, in almost all instances, later generations have a higher hypervolume indicator and larger Pareto size.

6.5 Experiment 5: Evaluating agent performance on actual problem

The final experiment concerns an application of the learned agent on the actual Warehouse Design and Control Problem (WDCP) central to this research. First the agent will be benchmarked against a random agent and the optimized NSGA-III algorithm without the use of AOS. Thereafter the evaluation is extended by executing an identical sensitivity analysis as in experiment 3. Replicating the sensitivity analysis is intended to show the ability of the agent to remain its performance, despite facing more challenging environmental features. All evaluations shown are aggregates over five independent optimization runs, showing the obtained average performance and the corresponding standard deviations.

6.5.1 The learned agent on the Warehouse Design and Control Problem

To evaluate the performance gained through implementation of Deep Reinforcement Learning (DRL), a comparison is made between the learned agent, a random action selecting agent and the optimized NSGA-III described described as no agent. The performance evaluation looks at both speed and quality of convergence, which are visualized in the hypervolume indicator over consecutive generations. The attained performances and their standard deviations over five independent runs for all three NSGA-III implementations are plotted in Figure 6.14.

Concluding from Figure 6.14, it can be seen that the learned agent outperforms both the random agent and the optimized NSGA-III (described as “No Agent”). With regards to solution quality, the learned agent attained a maximum hypervolume indicator of 0.92, which

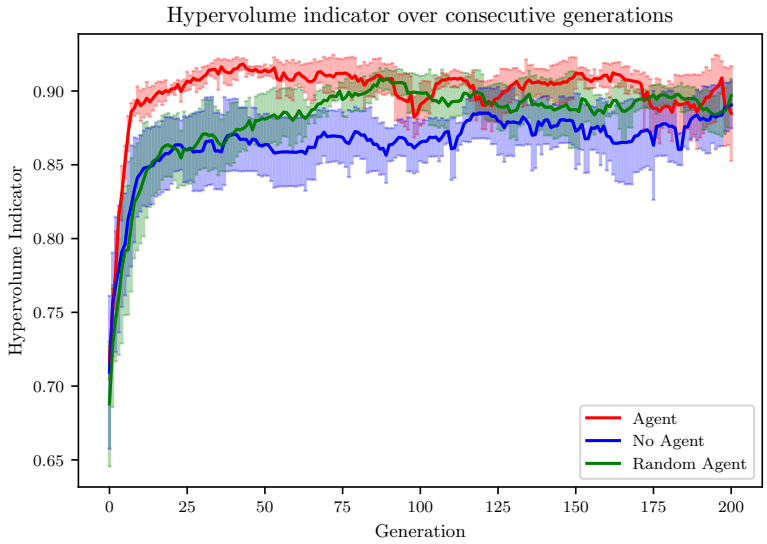


Figure 6.14: Performance benchmark DRL agent on the WDCP

is higher than the 0.91 and 0.89 achieved by the random agent and the optimized NSGA-III respectively. The fact that the random agent outperforms the optimized NSGA-III indicates the value of Adaptive Operator Selection (AOS). Implementing AOS increases the likelihood of breaking out of a converged Pareto optimal set, as there is a dynamic trade-off between exploration and exploitation. The hypervolume indicator value all three implementation of NSGA-III converge to is suspected to be the best attainable Pareto optimal set. Upon reaching this Pareto optimal set only fluctuations through domination can occur, which allows for all three lines to cross multiple times. This behavior can, in particular, be seen between the random and learned agent, starting around generation 100. Due to their ability to mutate more radically, they reach the suspected optimal hypervolume indicator value more rapidly, resulting in earlier fluctuations around one another.

Looking at the convergence speed of the algorithms, also the learned agent outperforms both the random agent and the optimized NSGA-III. The area under the curve, calculating by taking the overall sum of attained hypervolume indicator values, shows to be significantly higher for the agent. This metric results in a value of 181 as opposed to 177 and 174 for the random agent and optimized NSGA-III respectively. Besides faster convergence, implementation of an agent lowered the needed computation time of a single generation as well. The optimized NSGA-III takes around 55 seconds to execute all computation of a single configuration, whereas the agent induced algorithms take 47 seconds. This decrease in computation time can be dedicated to the removal of computations needed for Polynomial Mutation (PLM) [Algorithm 5], which are replaced with the DQN agent.

Aside from the improvement, also the standard deviation between the individual runs is decreased comparing the learned agent to both the random agent and optimized NSGA-III. Implying a better performing and more stable learning trajectory. This improvement is em-

phasized when looking at only the first 50 generations, as visualized in Figure 6.15.

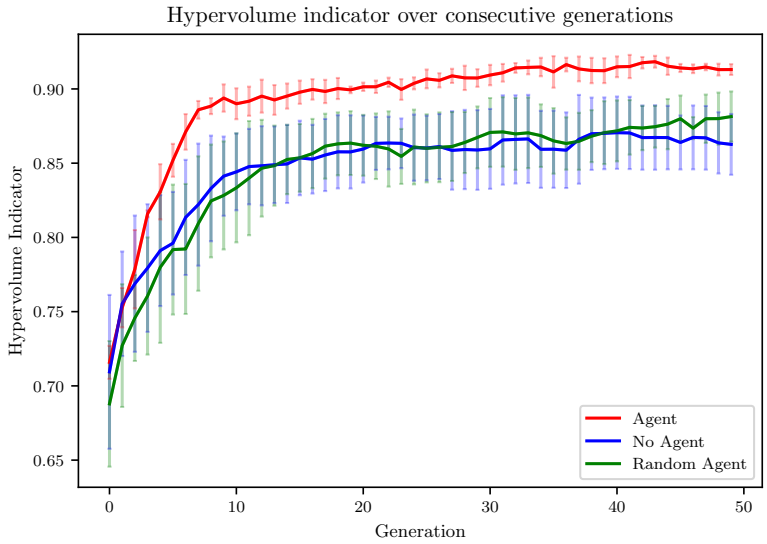


Figure 6.15: Close up of the first 50 generations of DRL agent performance on the WDCP

Reviewing Figure 6.15 it can clearly be seen that the agent suffers from lower volatility and attains a higher level of performance in a shorter window of time. These findings show the benefit of implementing a learned agent using Deep Reinforcement Learning as Adaptive Operator Selection within the NSGA-III for the WDCP central to this research.

6.5.2 Policy evaluation on the Warehouse Design and Control Problem

By comparing the behavior of the learned agent between the DTLZ2 benchmark and the WDCP environment, the utility of DRL instead of a simple heuristic can be proven. If the behavior of the agent shows similar for both problems, it could indicate that a simple set of rules would be sufficient to dynamically set the mutation operator settings. However if the behavior differs significantly, it shows the added value of the DRL agent in the need for sequential decision making. The behavior of the agent is visualized in Figure 6.16.

Concluding from Figure 6.16, it can be seen that the behavior significantly differs from the behavior on the DTLZ2 benchmark as shown in Figure 6.11. Identically to the DTLZ2 benchmark, the agent starts with behavior inclined towards exploitation. After reaching convergence to the highest hypervolume indicator the agent increases the severity of mutation. Eventually leading to radical mutation towards the final 50 generations. The inclination towards severe mutation in this period is larger than seen on the DTLZ2 benchmark. To gain insight into the reason for this difference in behavior, the average state representations are analyzed. Using the gathered experiences underlying the visualizations of the behavior of the agent, the average values for all parts of the state representation are calculated. These average values are plotted in Figure 6.17, excluding the generation number, as this value follows an identical trajectory for all problems and optimization runs.

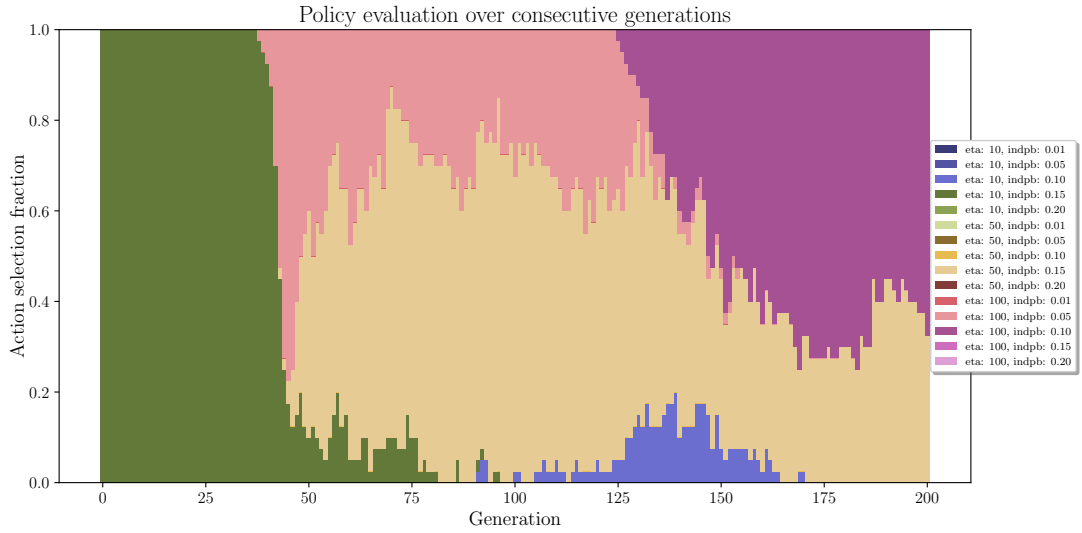


Figure 6.16: Policy visualization for the WDCP over consecutive generations)

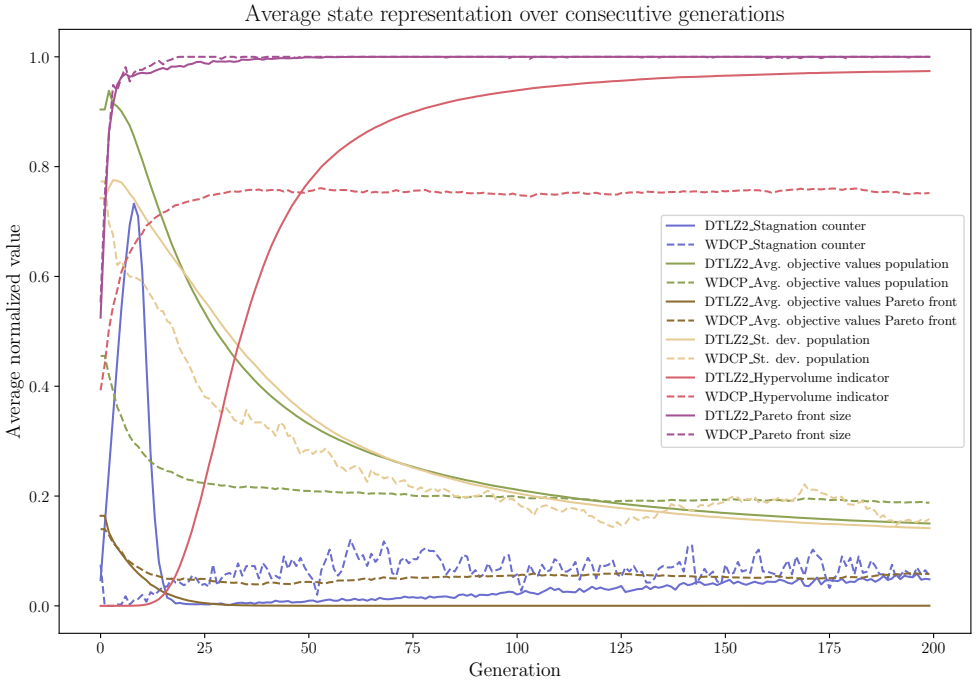


Figure 6.17: Visualization of state representation values over consecutive generations

Concluding from Figure 6.17, it can be seen that some values differ significantly over consecutive generations. Besides the spike in the stagnation counter, most apparent are the differences in the average objective values and standard deviation of the population and the hypervolume indicator. It is difficult to assign the effect to these values to the differences in behavior, although it is expected that these values combined have the largest contribution to the change in the behavior of the agent.

6.5.3 Sensitivity of the learned agent

To fully evaluate the performance of the learned agent an identical sensitivity analysis will be conducted as in Experiment 3, for which the results are shown in section 6.3. The results will show a comparison between the learned agent and the previously obtained results of the optimized NSGA-III without AOS. Visually indicating differences in performance between the two models.

First the amount and size of trucks is increased, leading to a higher overall workload and higher pressure on internal logistics. What can be seen in Figure 6.18 and 6.19 is that the performance of the agent stays above the performance of the optimized NSGA-III model, before converging to a nearly identical performance around generation 200. The highest solution quality is attained by the learned agent, with a hypervolume indicator value of 0.79 as opposed to 0.77 by the optimized NSGA-III. Also with regards to convergence speed, the performance of the agent is superior with an attained area under the hypervolume indicator curve surmounting to 154 compared to 150 for the optimized NSGA-III. Besides increased performance also the variability between the five independent runs is lowered throughout the process, eventually showing identical volatility around the final generations. Looking at the average Pareto performance, visualized in Figure 6.19, it can be seen that the agent handles the trade-off between truck tardiness differently. Showing decreased truck tardiness, albeit at an increase resource cost towards generation 200.

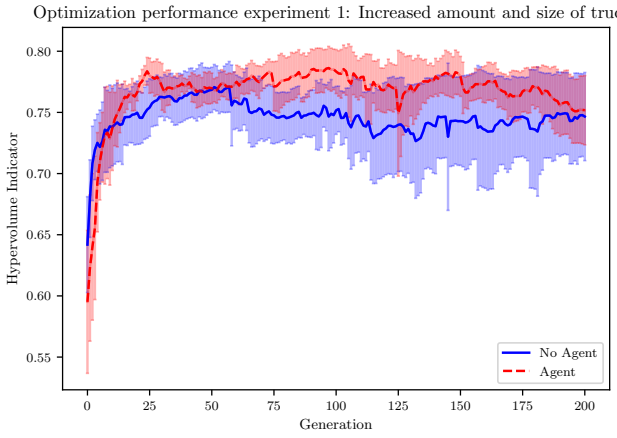


Figure 6.18: Increased amount and size of trucks - Hypervolume indicator

The second analysis concerns dropping the consistency of truck arrivals. This scenario

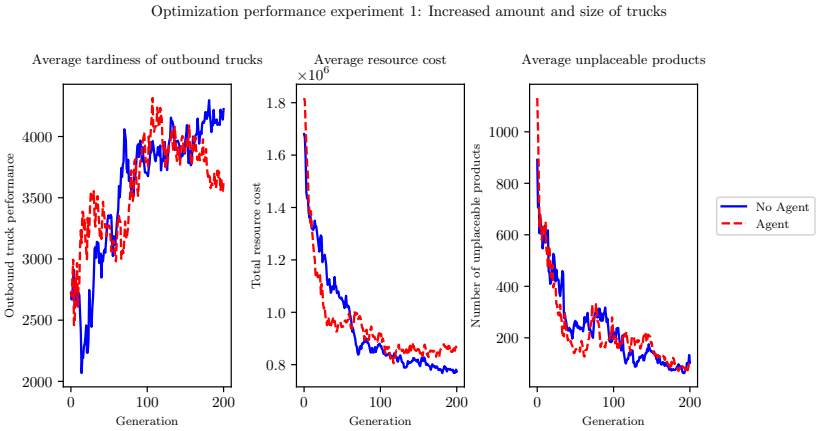


Figure 6.19: Increased amount and size of trucks - Average Pareto set performance

shifts 3/4 of the trucks to the first half of the day and the remaining 1/4 to the second half. Increasing the difficulty of the underlying Resource Allocation Problem (RAP). Comparing the results shown in Figure 6.20 and 6.21, no clear difference in performance can be noted. Both in terms of solution quality and convergence speed the learned agent and the optimized NSGA-III attain nearly identical results of 0.86 and 170 respectively. The same can be said for the average Pareto performance, which shows some differences but converges to a nearly identical score for all objectives after 200 generations. The reason for this might be due to the maximum attainable hypervolume indicator value, which is expected to be achieved by both algorithms. The standard deviation in the first half of the optimization trajectory is decreased however, showing some utility of applying the DQN agent in this scenario.

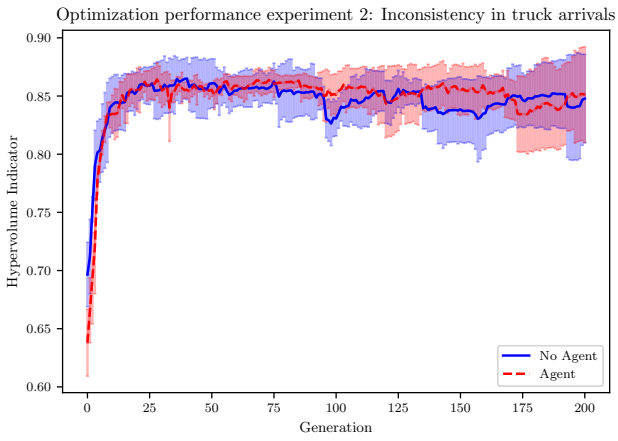


Figure 6.20: Inconsistent truck arrivals - Hypervolume indicator

The final experiment concerns an increase in product diversity. By doubling the product portfolio, from 1981 different products to 3962, the difficulty of the underlying Product Al-

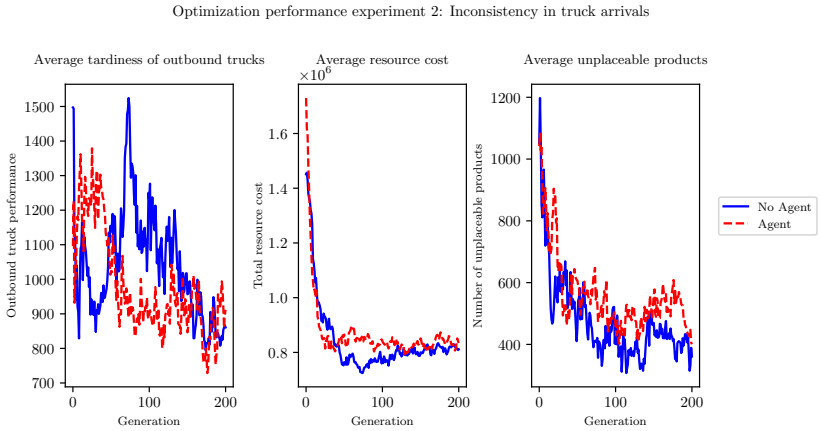


Figure 6.21: Inconsistent truck arrivals - Average Pareto set performance

location Problem (PAP) is increased. Previous analysis of the optimized NSGA-III did not show that much effect, besides a high level of volatility in the first 75 generations. Looking at the hypervolume indicator, shown in Figure 6.22, it can clearly be seen that both the average performance is increased and the volatility in the first 75 generations is decreased. Looking at the solution quality the learned agent, with a maximum hypervolume indicator value of 0.92, shows significant superiority towards the optimized NSGA-III which only attained a value of 0.90. Besides, also the convergence speed is increased with the area under the curve for the learned agent surmounting to 182 as opposed to the 175 attained by the optimized NSGA-III. However, looking at Figure 6.23, the average Pareto performance shows some interesting behavior. It can be seen that the agent finds and maintains a Pareto set with lower values for resource cost, resulting in a significantly higher average tardiness of outbound trucks. As the hypervolume indicator still remains above the optimized NSGA-III algorithm, it is expected that the lower resource cost is more beneficial to the dominated area than a lower average tardiness of outbound trucks. Which might be an indication of a bias introduced in setting the upper and lower bounds of these objectives. The result of this bias is that the algorithm will prefer a certain improvement over another.

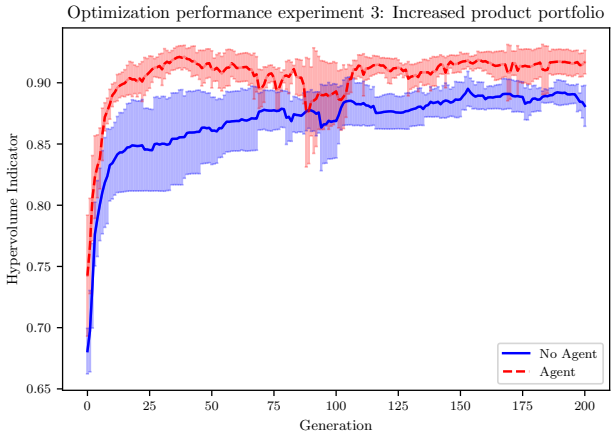


Figure 6.22: Increased product portfolio - Hypervolume indicator

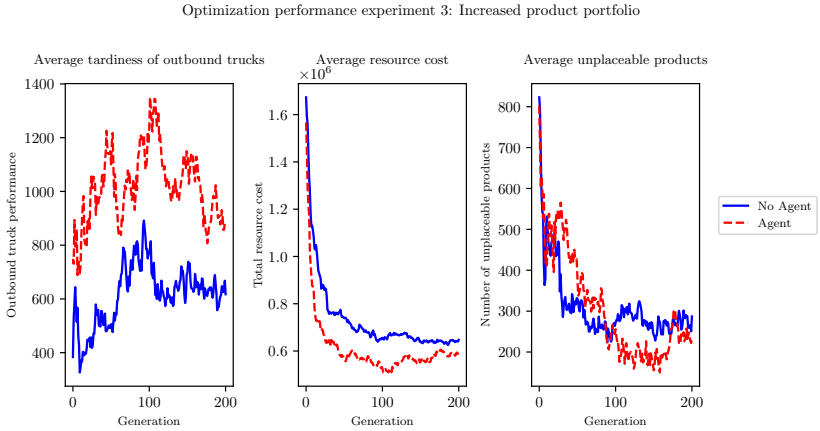


Figure 6.23: Increased product portfolio - Average Pareto set performance

Chapter 7

Conclusions and recommendations

This research project concerns a study focused on simultaneous optimization of warehouse layout design and control policies, using a low computational budget. Throughout literature this problem is referred to as the Warehouse Design and Control Problem (WDCP), which concerns collective optimization of a set of smaller warehouse-related problems. For this research the collection of sub-problems consists of the Warehouse layout design problem, the Resource Allocation Problem (RAP) and the Product Allocation Problem (PAP). A literature review is executed to investigate different approaches to the WDCP, showing little research on the subject. However, for the sub-problems a lot of approaches were found, which led to a design for the solution method used.

The research is commissioned by the consultancy company Nobleo Manufacturing, relating to a client of theirs. Complexity arises as both the solution quality and convergence speed are considered. Implying that computation time needs to be kept to a minimum, while remaining the quality of the found set of solutions. As Nobleo Manufacturing desired to apply the created framework to different warehouses it also needs to be generalizable, limiting the possibilities of framework designs. In this chapter the conclusion, and subsequently the recommendations, are formulated.

7.1 Conclusion

Throughout this research all evaluations are executed not on the actual (ED) simulation within Nobleo Manufacturing, but on a replication of this simulation in Python. This replication runs in discrete time, only contains necessary computations, uses smart lookup tables, runs in parallel and is more robust against extreme scenarios. Evaluation of a warehouse configuration consists of 40 hours in simulation time, for which the computation time is decreased from 30 minutes to on average 2-3 seconds. Making execution of this research tractable, which was not possible on the original ED simulation model.

The solution method used consists of two parts, namely an optimization algorithm and a Deep Reinforcement Learning (DRL) agent. The optimization algorithm used concerns the Non-dominated Sorting Genetic Algorithm III (NSGA-III). NSGA-III outperforms both its predecessor NSGA-II and the SMPSO algorithm on the Warehouse Design and Control Problem (WDCP) central to this research, as shown in experiment 1 for which the results can

be found in section 6.1. The hyperparameters of the NSGA-III are optimized towards a high computation speed and solution quality. Both of these convergence properties are taken into account by maximization of the total sum of attained hypervolume indicator values, representing the area under the curve of this value. Thereafter the optimized NSGA-III is shown to be robust against extreme scenarios. The algorithm is able to maintain its optimization performance, converging to a Pareto front despite being faced with challenging circumstances. The optimized hyperparameter settings resulted in some increase in convergence speed, but this method is not generalizable. The parameters are tailored to this specific problem, having no guarantee to be optimal on different problems. Interestingly, the hyperparameter optimization of the NSGA-III showed little importance for adapting the crossover parameter. This effect can be explained by the non-convexity of the objective space. As the local neighborhood of well-performing individuals does not necessarily have to lead to better performance, exploitation becomes less beneficial. For this reason the DRL agent will only focus on the mutation operators, as these are perceived important by the hyperparameter feature importance analysis.

The desired generic increase in convergence speed can be attained using Deep Reinforcement Learning (DRL). DRL is applied in the Adaptive Operator Selection (AOS) of the NSGA-III. What this means is that a DRL agent will be trained to dynamically set the hyperparameter values of the operators within the NSGA-III based on the current state of optimization. The algorithm used concerns the Deep Q-Networks (DQN) algorithm. DQN is known to be sample efficient and stable, due to the use of experience replay and a frozen target network respectively. The used design of the agent consists of a state representation of seven values, focusing on generational progress, population performance and the performance of the approximated Pareto front. The action space consist of the parametric values of the Polynomial Mutation (PLM) operator used within the NSGA-III. During training an episodic reward is used, which consists of the total sum of hypervolume indicator values. This metric is identical to the metric used in hyperparameter optimization, as it takes both convergence speed and solution quality into account.

Comparing the performance of the learned agent induced NSGA-III model with the optimized NSGA-III model, it can be concluded that the model shows equal or improved performance on all tests. Showing an increase in performance on 3/4 of the tested scenarios. The scenario in which no improvement is made reached identical performance. The most probable reason for the agent not improving in this scenario is that the performance attained by the optimized NSGA-III is already the maximum attainable performance. Interestingly the obtained improvements reach further than the initial purpose of speeding up convergence. Implementation of DRL also improved, in most cases, the solution quality of the found Pareto front. Due to the dynamic trade-off between exploration and exploitation the algorithm is able to escape from local optima, increasing the chance of reaching points belonging to the set of globally optimal solutions. The need to use DRL for AOS is emphasized through evaluation of the behavior of the DQN agent. The difference in behavioral patterns over consecutive generations between the two problems, being the DTLZ2 benchmark and the WDCP, indicates the inability to replace DRL with a set of simplistic rules. As a static set of rules will not suffice, sequential decision making is needed.

Throughout the design of the solution methods close attention is paid to the generalizab-

ility of the framework. Both the optimization algorithm as the Adaptive Operator Selection (AOS) using DRL are entirely generalizable. The optimization algorithm, the NSGA-III, does not use problem specific information. Instead it works with the problem as a black box, only used to retrieve the performance of proposed configurations. This characteristic enables it to be used for different problems, in which it should be able to return a good approximation of the actual Pareto front. The used design of the DRL agent also allows for it to be generalizable, as it only uses generic information about the optimization process. To allow extension of the number of objective values beyond the three objective values used in this research, the performance metrics of the population are implemented as aggregates. More specifically, all objective values are normalized using Min-Max normalization on a given upper and lower bound. Thereafter the average is taken over all these normalized objective values. This results in a value between 0 and 1, summarizing the average performance of the population. As we take the average over all normalized objectives, the DRL agent is able to calculate these values without imposing any constraint on the allowed number of objective values used.

7.2 Limitations and recommendations

The executed research showed a method to solve the Warehouse Design and Control Problem (WDCP). Using an evolutionary algorithm a solution to the WDCP is successfully developed, showing improved performance on the algorithm by implementing Deep Reinforcement Learning (DRL) as AOS for the NSGA-III. Although successful, this research is not complete. Due to the limited time frame for the master thesis, some aspects and challenges are excluded from this research. Future researchers and practitioners are therefore advised to:

1. **Increase research into volatility**
   Due to the remaining high computation time it was intractable to do sufficient measurements to show statistical significance in the results. Some indication of volatility is included by using five independent runs for all results shown, which is expected to be sufficient to prevent faulty statements. However, to be more secure about the actual performance and prove statistical significance, this aspect of the research should be executed more thoroughly.

2. **Optimize model performance**
   The proposed solution method performs well, although the development was approached to create a proof of concept. Extension of hyperparameter optimization for the unaffected parameters within the NSGA-III algorithm and the parametric values guiding the Double DQN agent, could result in improved performance. Some optimization is done, for the NSGA-III algorithm using Optuna and for the Double DQN agent on the DTLZ2 benchmark, however there is still room for improvement in testing different algorithmic combinations or underlying hyperparameter settings.

3. **Investigate different population sizes**
   Throughout this research a population size of 20 individuals is maintained. Initially this figure was set in the benchmarking of the NSGA-III against the NSGA-II and SMPSO algorithm, due to the computational overhead. It is concluded that increasing the population size will increase computation time significantly. However, no further investigation is executed in the optimal trade-off between computation time and performance as a result of the size of the population. As discussed in section 6.3.1, the

population size of 20 might be lowering the robustness of the algorithm. Increasing the size will allow for a larger number of individuals comprising the Pareto front, which presumably will also result in less volatility in the performance evaluation using the hypervolume indicator.

4. **Analyze model biases**
   The results of the third sensitivity analysis using the learned agent, showed some unexpected behavior. As discussed, this potentially might indicate a bias introduced in setting the upper and lower bound of the objective values. Future work might look into improving these values or even getting rid of these values altogether. Doing this would improve the robustness of the model as the found potential bias would be removed.

5. **Improve understanding of Pareto solutions**
   Throughout this research the quality of proposed warehouse configurations is not considered. The reason for this is that the purpose of multi-objective optimization is to return a set of optimal solutions. Thereafter it is the task of the decision maker to choose which solution is desired. Before implementation of the model, it is advised to also include analysis of the returned solutions. Enabling adjustment of model parameters if the selection of results does not satisfy the end user.

Bibliography

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 43, 50

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38. 20, 21

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256. 19

Azimi, M., Beheshti, R., Imanzadeh, M., and Nazari, Z. (2013). Optimal allocation of human resources by using linear programming in the beverage company. *Universal Journal of Management and Social Sciences*, 3(5):48–54. 13

Bennour, M., Crestani, D., Crespo, O., and Prunet, F. (2005). Computer-aided decision for human task allocation with mono-and multi-performance evaluation. *International Journal of Production Research*, 43(21):4559–4588. 13

Bretthauer, K. M. and Shetty, B. (1995). The nonlinear resource allocation problem. *Operations research*, 43(4):670–683. 13

Cui, Y., Geng, Z., Zhu, Q., and Han, Y. (2017). Multi-objective optimization methods and application in energy saving. *Energy*, 125:681–704. 15, 16

DaCosta, L., Fialho, A., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 913–920.

Daskalaki, S., Birbas, T., and Housos, E. (2004). An integer programming formulation for a case study in university timetabling. *European journal of operational research*, 153(1):117–135.

Davis, L. (1991). Handbook of genetic algorithms.

De Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European journal of operational research*, 182(2):481–501.

Deb, K. (2014). Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer.

Deb, K., Agrawal, R. B., et al. (1995). Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International conference on parallel problem solving from nature*, pages 849–858. Springer.

Deb, K. and Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002). Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 825–830. IEEE.

Durgut, R. and Aydin, M. E. (2021). Reinforcement learning-based adaptive operator selection. In *International Conference on Optimization and Learning*, pages 29–41. Springer.

Fan, K., You, W., and Li, Y. (2013). An effective modified binary particle swarm optimization (mbpso) algorithm for multi-objective resource allocation problem (morap). *Applied Mathematics and Computation*, 221:257–267.

Fan, X., Sayers, W., Zhang, S., Han, Z., Ren, L., and Chizari, H. (2020). Review and classification of bio-inspired algorithms and their applications. *Journal of Bionic Engineering*, 17:611–631.

Fister Jr, I., Yang, X.-S., Fister, I., Brest, J., and Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*.

Geng, Z., Cui, Y., Xia, L., Zhu, Q., and Gu, X. (2012). Compromising adjustment solution of primary reaction coefficients in ethylene cracking furnace modeling. *Chemical engineering science*, 80:16–29.

Guerreiro, A. P., Fonseca, C. M., and Paquete, L. (2020). The hypervolume indicator: Problems and algorithms. *arXiv preprint arXiv:2005.00515*.

Guerriero, F., Musmanno, R., Pisacane, O., and Rende, F. (2013). A mathematical model for the multi-levels product allocation problem in a warehouse with compatibility constraints. *Applied Mathematical Modelling*, 37(6):4385–4398.

Guresen, E. and Kayakutlu, G. (2011). Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3:426–433.

Hamdan, M. (2010). On the disruption-level of polynomial mutation for evolutionary multi-objective optimisation algorithms. *Computing and Informatics*, 29(5):783–800.

He, C., Tian, Y., Wang, H., and Jin, Y. (2019). A repository of real-world datasets for data-driven evolutionary multiobjective optimization. *Complex & Intelligent Systems*, pages 1–9.

Heragu, S. S., Du, L., Mantel, R. J., and Schuur, P. C. (2005). Mathematical model for warehouse design and product allocation. *International Journal of Production Research*, 43(2):327–338.

Hernández-Díaz, A. G., Santana-Quintero, L. V., Coello, C. A. C., and Molina, J. (2007). Pareto-adaptive -dominance. *Evolutionary computation*, 15(4):493–517.

Hlal, M. I., Ramachandaramurthya, V. K., Padmanaban, S., Kaboli, H. R., Pouryekta, A., Abdullah, T., and Ab Rashid, T. (2019). Nsga-ii and mopso based optimization for sizing of hybrid pv/wind/battery energy storage system. *Int. J. Power Electron. Drive Syst*, 10(1):463–478.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Horoba, C. and Neumann, F. (2008). Benefits and drawbacks for the use of epsilon-dominance in evolutionary multi-objective optimization. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, page 641–648, New York, NY, USA. Association for Computing Machinery.

Huang, C., Li, L., He, C., Cheng, R., and Yao, X. (2021). Operator-adapted evolutionary large-scale multiobjective optimization for voltage transformer ratio error estimation. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 672–683. Springer.

Huang, V. L., Qin, A. K., Suganthan, P. N., and Tasgetiren, M. F. (2007). Multi-objective optimization based on self-adaptive differential evolution algorithm. In *2007 IEEE Congress on Evolutionary Computation*, pages 3601–3608. IEEE.

Huang, X., Lei, X., and Jiang, Y. (2012). Comparison of three multi-objective optimization algorithms for hydrological model. In *International Symposium on Intelligence Computation and Applications*, pages 209–216. Springer.

Jain, A., Lalwani, S., and Lalwani, M. (2018). A comparative analysis of mopso, nsga-ii, spea2 and pesa2 for multi-objective optimal power flow. In *2018 2nd International Conference on Power, Energy and Environment: Towards Smart Technology (ICEPE)*, pages 1–6. IEEE.

Ju, L., Tan, Z., Li, H., Tan, Q., Yu, X., and Song, X. (2016). Multi-objective operation optimization and evaluation model for cchp and renewable energy based hybrid energy system driven by distributed energy resources in china. *Energy*, 111:322–340.

Karafotias, G., Hoogendoorn, M., and Eiben, A. (2015). Evaluating reward definitions for parameter control. In *European Conference on the Applications of Evolutionary Computation*, pages 667–680. Springer.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

LaTorre, A., Molina, D., Osaba, E., Del Ser, J., and Herrera, F. (2020). Fairness in bio-inspired optimization research: A prescription of methodological guidelines for comparing meta-heuristics. *arXiv preprint arXiv:2004.09969*.

Li, K., Fialho, A., Kwong, S., and Zhang, Q. (2013). Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130.

Li, Z., Low, M. Y. H., and Lim, R. Y. G. (2009). Optimal decision-making on product allocation for crossdocking and warehousing operations. *International Journal of Services Operations and Informatics*, 4(4):352–365.

Liagkouras, K. and Metaxiotis, K. (2013). An elitist polynomial mutation operator for improved performance of moeas in computer networks. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–5. IEEE.

Lin, Q., Liu, Z., Yan, Q., Du, Z., Coello, C. A. C., Liang, Z., Wang, W., and Chen, J. (2016). Adaptive composite operator selection and parameter control for multiobjective evolutionary algorithm. *Information Sciences*, 339:332–352.

Lindauer, M., Hoos, H. H., Hutter, F., and Schaub, T. (2015). Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778.

Lu, L., Anderson-Cook, C. M., and Robinson, T. J. (2012). A case study to demonstrate a pareto frontier for selecting a best response surface design while simultaneously optimizing multiple criteria. *Applied Stochastic Models in Business and Industry*, 28(3):206–221.

Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395.

McClymont, K. and Keedwell, E. C. (2011). Markov chain hyper-heuristic (mchh) an online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 2003–2010.

Miralles, C., García-Sabater, J. P., Andrés, C., and Cardós, M. (2008). Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled. *Discrete Applied Mathematics*, 156(3):352–367.

Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., and Ouni, A. (2015). Many-objective software remodularization using nsga-iii. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):1–45.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Monsef, H., Naghashzadegan, M., Jamali, A., and Farmani, R. (2019). Comparison of evolutionary multi objective optimization algorithms in optimum design of water distribution network. *Ain Shams Engineering Journal*, 10(1):103–111.

Moore, J. and Chapman, R. (1999). Application of particle swarm to multiobjective optimization: Dept. *Comput. Sci. Software Eng., Auburn Univ.*

Mutlu, Ö., Polat, O., and Supciller, A. A. (2013). An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-ii. *Computers & Operations Research*, 40(1):418–426.

Nebro, A. J., Durillo, J. J., Garcia-Nieto, J., Coello, C. C., Luna, F., and Alba, E. (2009). Smpso: A new pso-based metaheuristic for multi-objective optimization. In *2009 IEEE Symposium on computational intelligence in multi-criteria decision-making (MCDM)*, pages 66–73. IEEE.

Nobleo (2021). Over nobleo manufacturing.

Odeniyi, O., Omidiora, E., Olabiyisi, S., and Aluko, J. (2015). Development of a modified simulated annealing to school timetabling problem. *International Journal of Applied Information Systems*, 8(2):16–24.

Pandit, R. and Palekar, U. S. (1993). Response time considerations for optimal warehouse layout design.

Parsopoulos, K. E. and Vrahatis, M. N. (2008). Multi-objective particles swarm optimization approaches. In *Multi-objective optimization in computational intelligence: Theory and practice*, pages 20–42. IGI global.

Roodbergen, K. J. and Vis, I. F. (2006). A model for warehouse layout. *IIE transactions*, 38(10):799–811.

Roodbergen, K. J., Vis, I. F., and Taylor Jr, G. D. (2015). Simultaneous determination of warehouse layout and control policies. *International Journal of Production Research*, 53(11):3306–3326.

Rosenblatt, M. J. and Roll, Y. (1984). Warehouse design with storage policy considerations. *The International Journal of Production Research*, 22(5):809–821.

Rosenblatt, M. J. and Roll, Y. (1988). Warehouse capacity in a stochastic environment. *The International Journal Of Production Research*, 26(12):1847–1851.

Rouwenhorst, B., Reuter, B., Stockrahm, V., van Houtum, G.-J., Mantel, R., and Zijm, W. H. (2000). Warehouse design and control: Framework and literature review. *European journal of operational research*, 122(3):515–533.

Sammons Jr, N., Yuan, W., Eden, M., Aksoy, B., and Cullinan, H. (2008). Optimal biorefinery product allocation by combining process and economic modeling. *Chemical Engineering Research and Design*, 86(7):800–808.

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

Seada, H. and Deb, K. (2015). Effect of selection operator on nsga-iii in single, multi, and many-objective optimization. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2915–2922. IEEE.

Sewak, M. (2019). Deep q network (dqn), double dqn, and dueling dqn. In *Deep Reinforcement Learning*, pages 95–108. Springer.

Sharma, M., Komninos, A., López-Ibáñez, M., and Kazakov, D. (2019). Deep reinforcement learning based parameter control in differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 709–717.

Shukla, P. K. and Deb, K. (2007). On finding multiple pareto-optimal solutions using classical and evolutionary generating methods. *European Journal of Operational Research*, 181(3):1630–1652.

Si, B., Wang, J., Yao, X., Shi, X., Jin, X., and Zhou, X. (2019). Multi-objective optimization design of a complex building based on an artificial neural network and performance evaluation of algorithms. *Advanced Engineering Informatics*, 40:93–109.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888.

Sun, J., Liu, X., Bäck, T., and Xu, Z. (2021). Learning adaptive differential evolution algorithm from optimization experiences by policy gradient. *IEEE Transactions on Evolutionary Computation*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tan, Z. and Li, K. (2021). Differential evolution with mixed mutation strategy based on deep reinforcement learning. *Applied Soft Computing*, 111:107678.

Thepphakorn, T., Pongcharoen, P., and Hicks, C. (2014). An ant colony based timetabling tool. *International Journal of Production Economics*, 149:131–144.

Tian, Y., Li, X., Ma, H., Zhang, X., Tan, K. C., and Jin, Y. (2022). Deep reinforcement learning based adaptive operator selection for evolutionary multi-objective optimization.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Vila, M. and Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research*, 44:105–114.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

Xia, L., Cui, Y., Gu, X., and Geng, Z. (2013). Kinetics modelling of ethylene cracking furnace based on sqp-cpso algorithm. *Transactions of the Institute of Measurement and Control*, 35(4):531–539.

Xian-Ying, M. (2012). Application of assignment model in pe human resources allocation. *Energy Procedia*, 16:1720–1723.

Younas, I., Kamrani, F., Schulte, C., and Ayani, R. (2011). Optimization of task assignment to collaborating agents. In *2011 IEEE Symposium on Computational Intelligence in Scheduling (SCIS)*, pages 17–24. IEEE.

Zitzler, E. and Künzli, S. (2004). Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*, pages 832–842. Springer.

Appendix A

Product flow in warehouse operations

This appendix further explains the internal process of the warehouse under consideration. The overview shows the process from a product view, which implies that it shows the journey of a single product in the warehouse from arrival within an inbound truck and departure in an outbound truck.

First a truck containing a set of product arrives to the warehouse. The truck is docked, after which the product is deloaded into the assigned consolidation area. Thereafter the products are separately transported to a storage location in the warehouse, where it sits in inventory until an order for the product arrives.

An order of an outbound truck arrives two hours before arrival of the actual truck. This allows to prepare the requested product in the assigned consolidation area. If a broken pallet is requested, i.e. less-than-pallet size, the pallet is broken in the consolidation area after which the remaining part is taken back to inventory. If all products are present in the consolidation area and the outbound truck has arrived, the truck is loaded. If all product are loaded onto the truck it departs, which completes the process.

Based on the schematic overview of the process, different events are dissected. These events are used as the basis for the Python replication of the simulation used by Nobleo. The replication is made as a Discrete Event Simulation (DES), which only simulates events instead of continuous time. The following, consecutive, events are dissected:

1. Inbound truck arrival
2. Deload truck complete
3. Quality check done
4. Product to storage complete
5. Outbound truck order arrival
6. Product to consolidation complete
7. Outbound truck arrival
8. Load truck complete

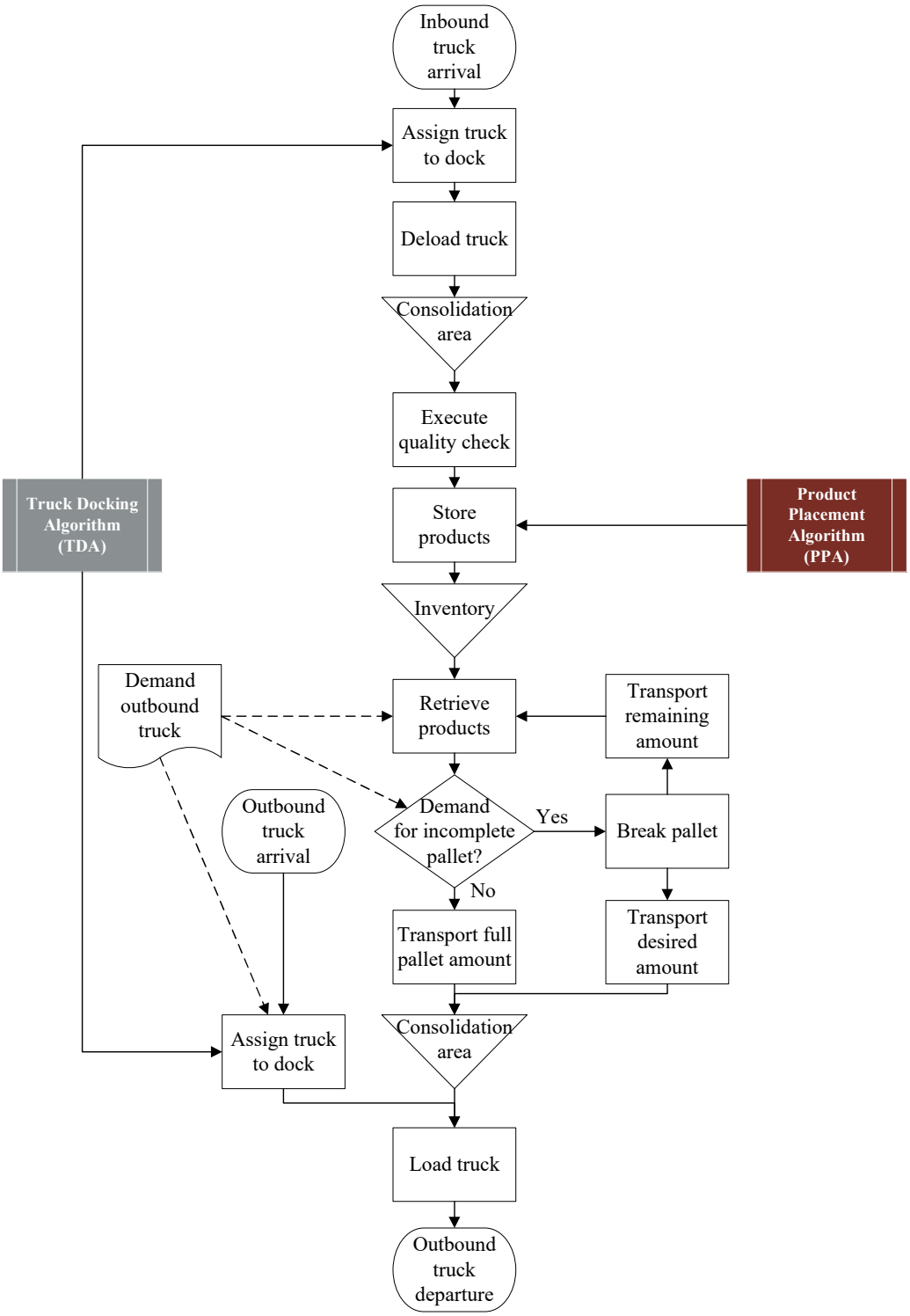


Figure A.1: Visualization of the process flow

Appendix B

Discrete Event Simulation

The discrete event simulation is constructed based on the events retrieved from the product flow, described in Appendix A. As the complexity of the simulation does not allow for a complete overview, it will be explained in two levels of detail. First a global overview of the process is shown, after which certain relevant processes will be visualized in greater detail.

The first level of detail shows the flowchart of the main process, visualized in Figure B.1. It shows the initialization and termination of the simulation process. During initialization all truck arrival events, for both in- and outbound, are created and added to the FES. Every cycle of the simulation functionality a single event is handled, which concerns the main functionality of a Discrete Event Simulation (DES). The eight events shown, are the events described in Appendix A. Every event is handled accordingly after which creation of a follow-up event is executed if needed.

The second level of detail will further explain the process of handling different events. Per event the handling of the event is further explained in more detailed flowcharts. Figure B.2 explains the process executed in simulation upon arrival of an inbound truck arrival. This process creates a follow-up event, namely Deload truck complete, as visualized in Figure B.3. After all pallets are taken from the truck it departs and the quality check is executed. Upon completion of this check the first product to storage job is created, as can be seen in Figure B.4. The product to storage jobs are repeated until the entire content of the consolidation area is placed within the warehouse, as shown in Figure B.5. These steps constitute the process for inbound trucks.

For outbound trucks the process consists of the following steps. If the outbound truck does not concern a rush order, the requested products will be known two hours in advance. Upon receiving of this list of products, the process as visualized in Figure B.6 is started. If the outbound truck does concern a rush order identical steps are executed, shown on the left side of Figure B.7. This process creates the first product to consolidation job. Which is repeated until all products are retrieved from inventory and placed in the assigned consolidation lane, as shown in Figure B.8. Either if the truck has already arrived when the last product is received or upon arrival of the truck all products are already in the assigned consolidation lane, as can be seen on the right side of Figure B.7, a load truck job is created. Upon completion of this job, the truck departs and the performance is measured, as shown in Figure B.9.

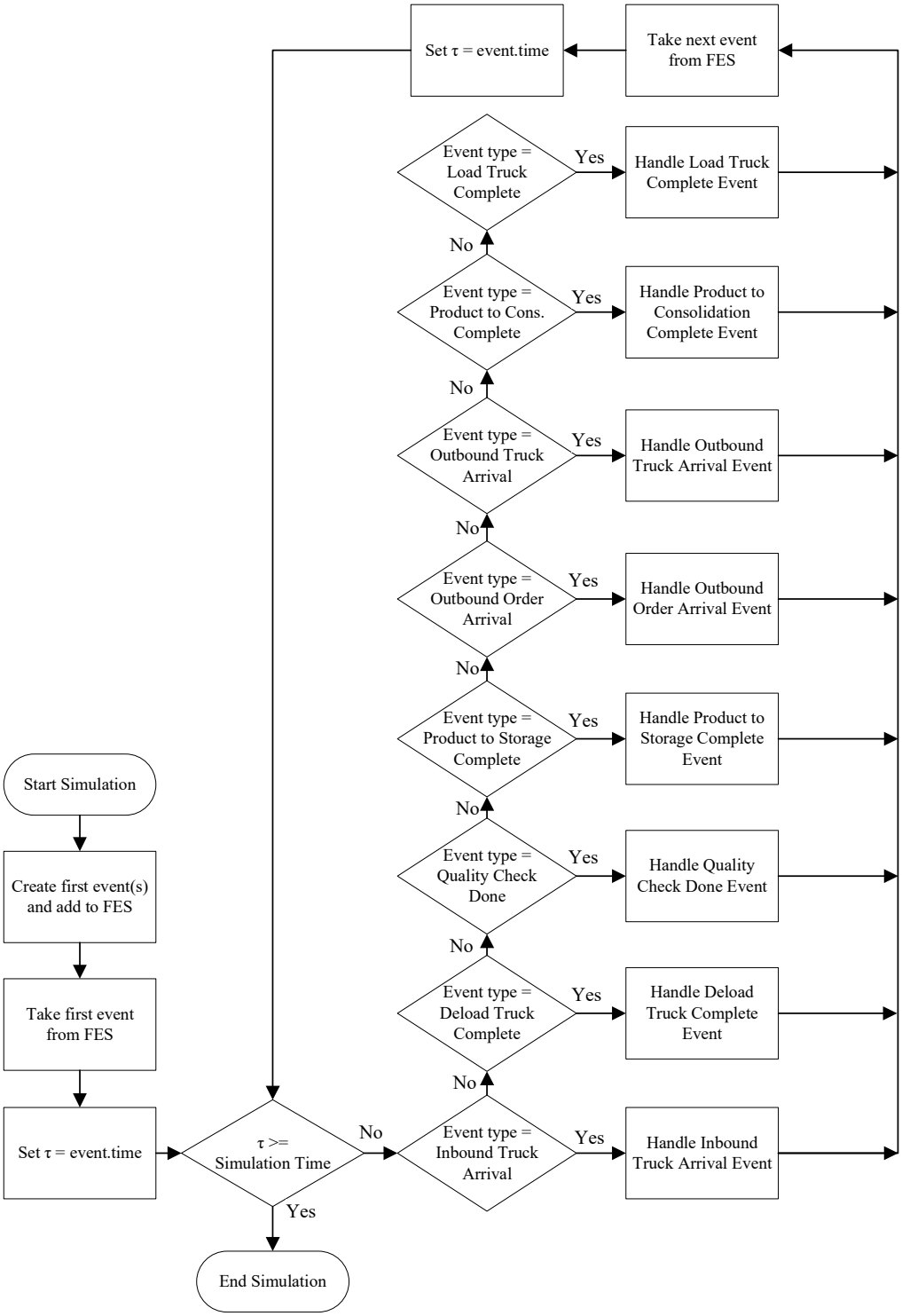


Figure B.1: Flowchart DES- High level basis model

B.1 Event 1: Inbound truck arrival

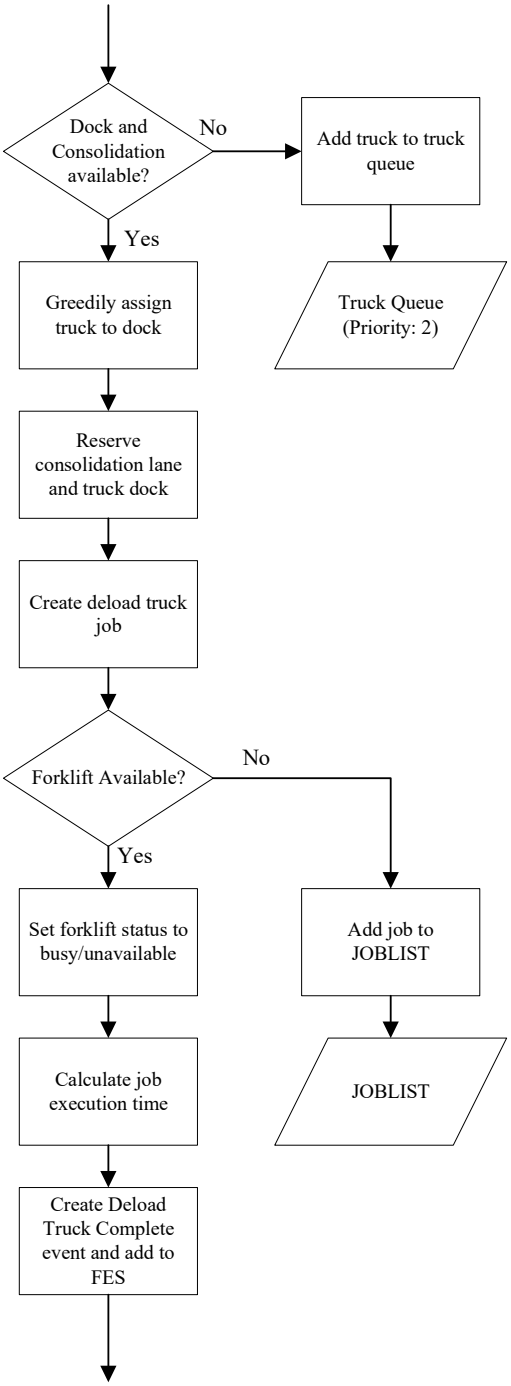


Figure B.2: Flowchart DES - Inbound truck arrival event

B.2 Event 2: Deload truck complete

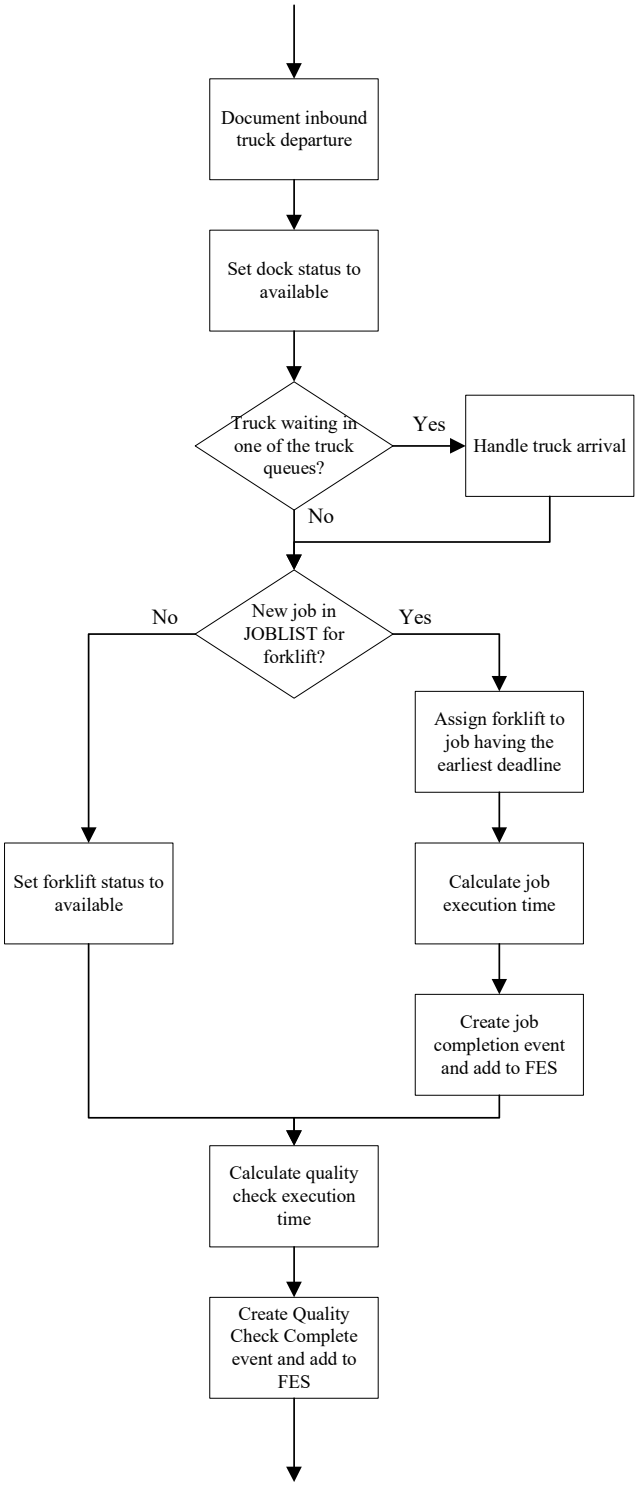


Figure B.3: Flowchart DES - Deload (inbound) truck complete event

B.3 Event 3: Quality check done

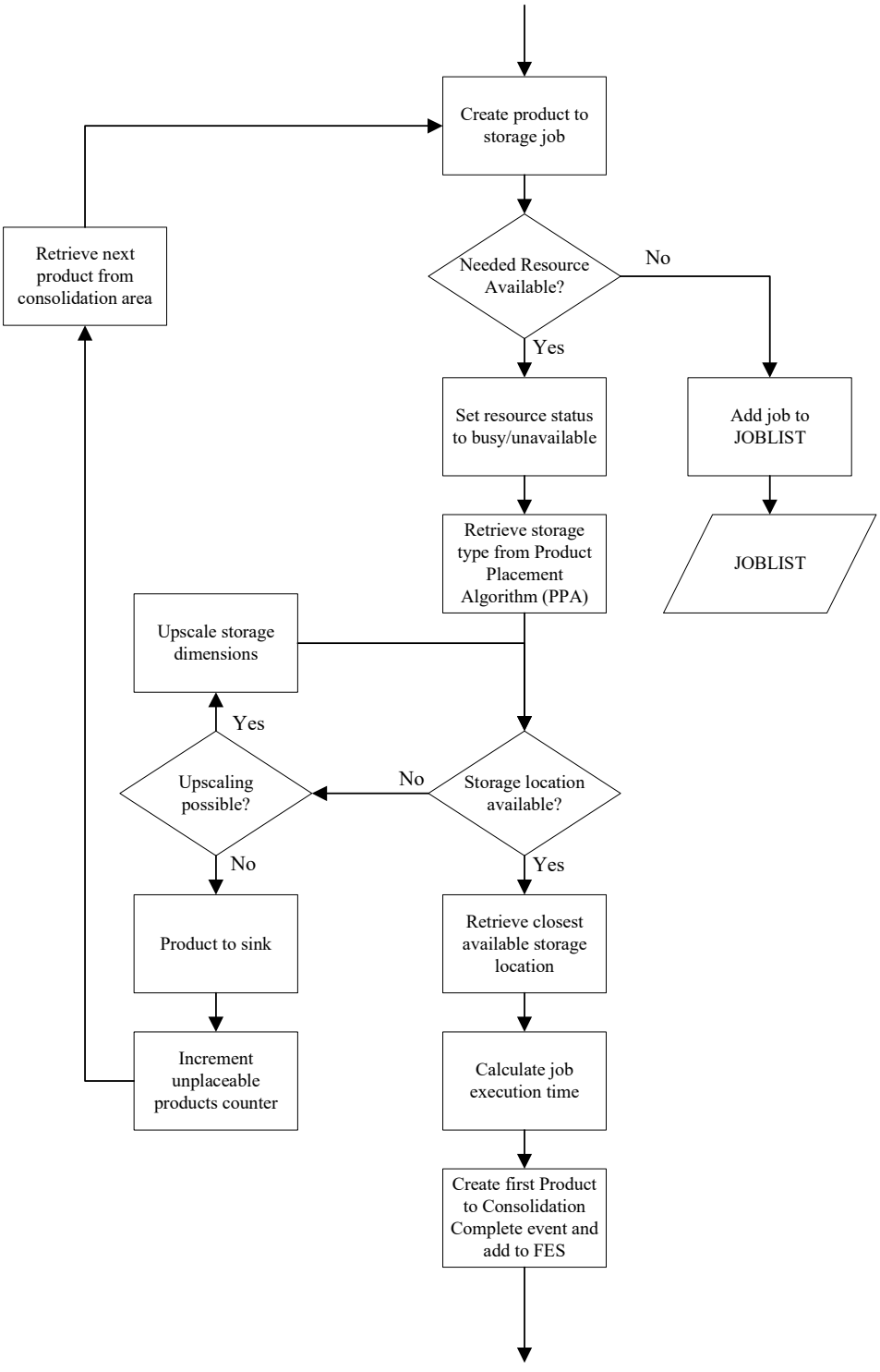


Figure B.4: Flowchart DES - Quality check complete event

B.4 Event 4: Product to storage complete

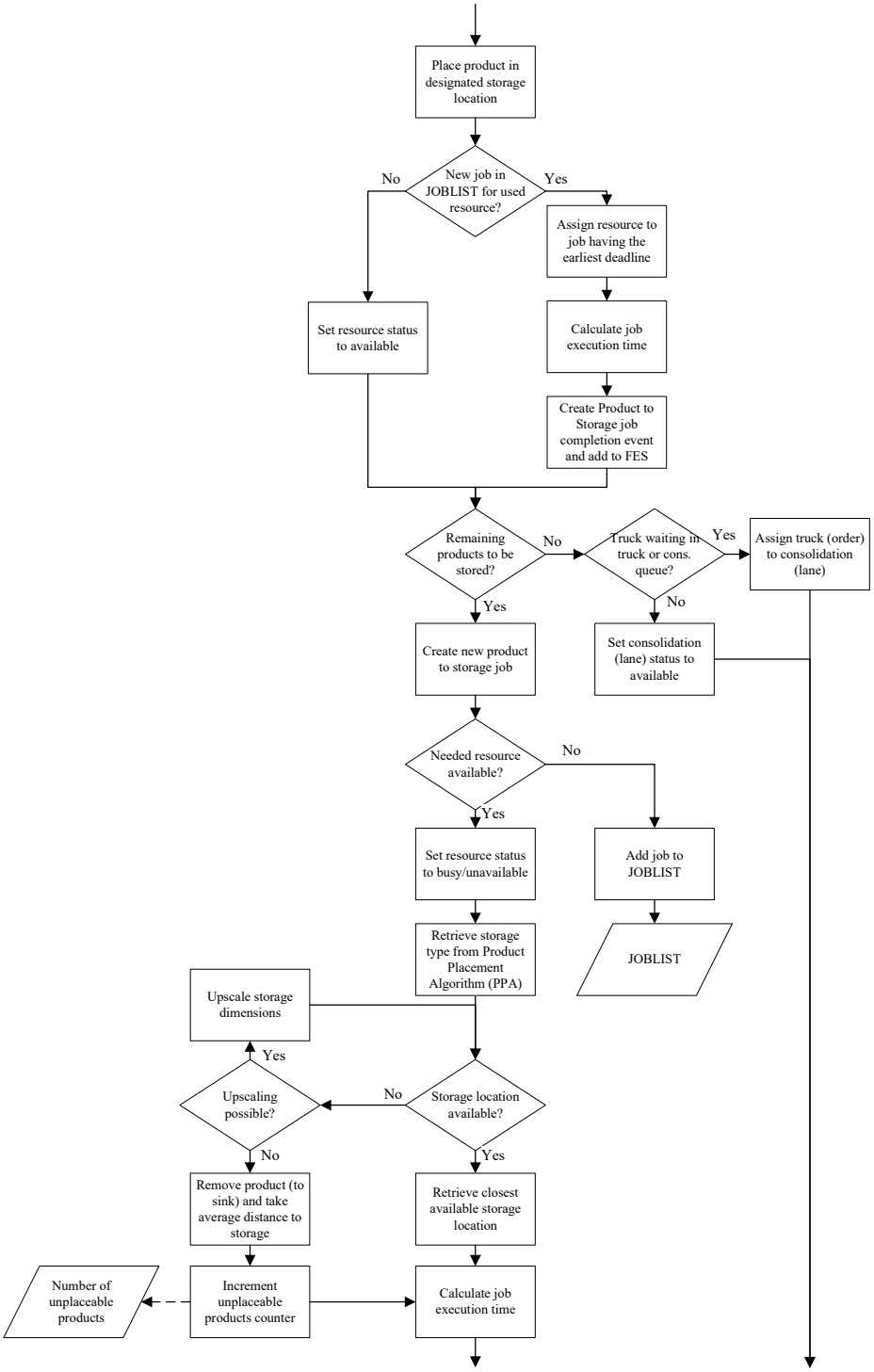


Figure B.5: Flowchart DES - Product to storage complete event

B.5 Event 5: Outbound order arrival

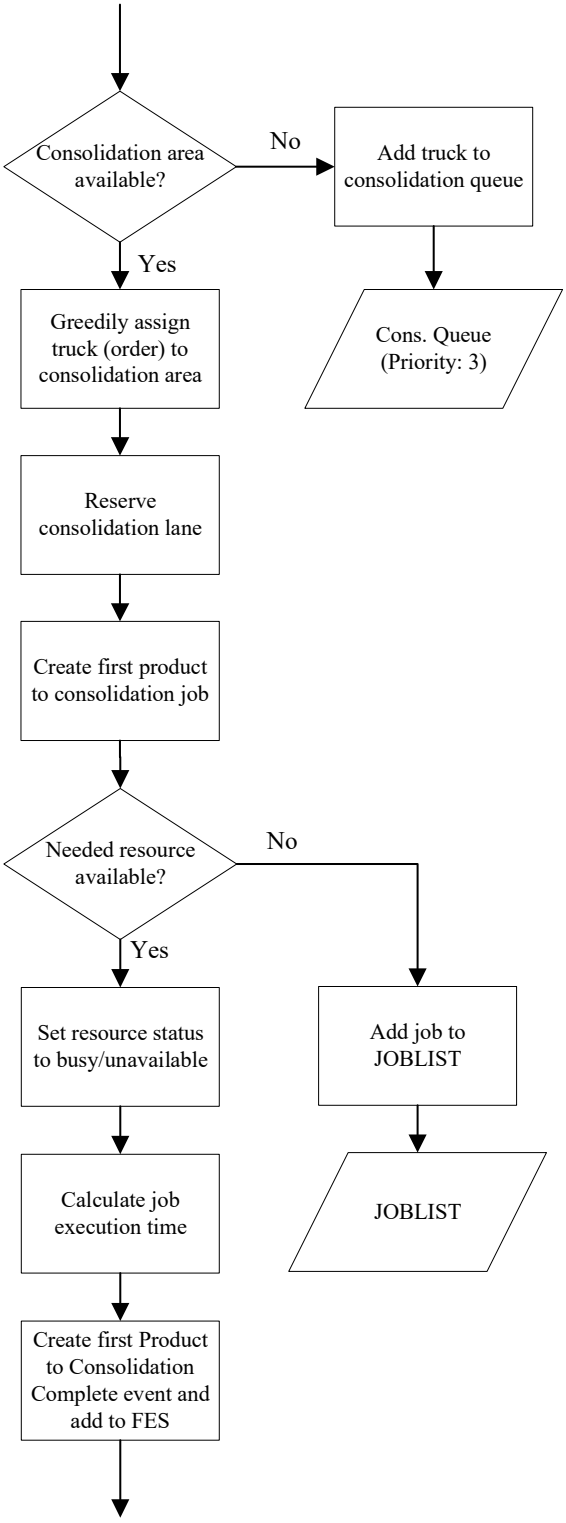


Figure B.6: Flowchart DES - Outbound order arrival event

B.6 Event 6: Outbound truck arrival

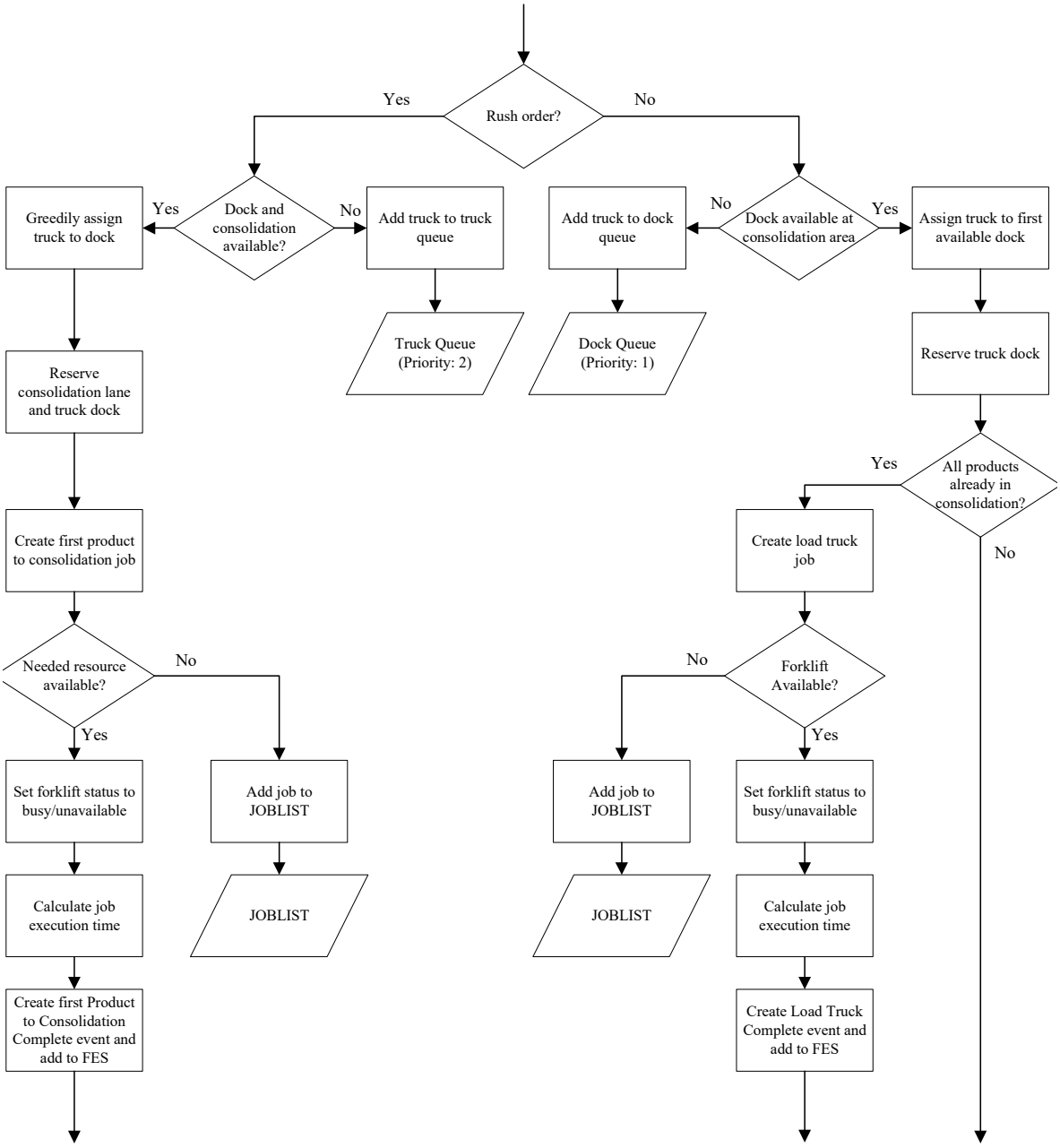


Figure B.7: Flowchart DES - Outbound truck arrival event

B.7 Event 7: Product to consolidation

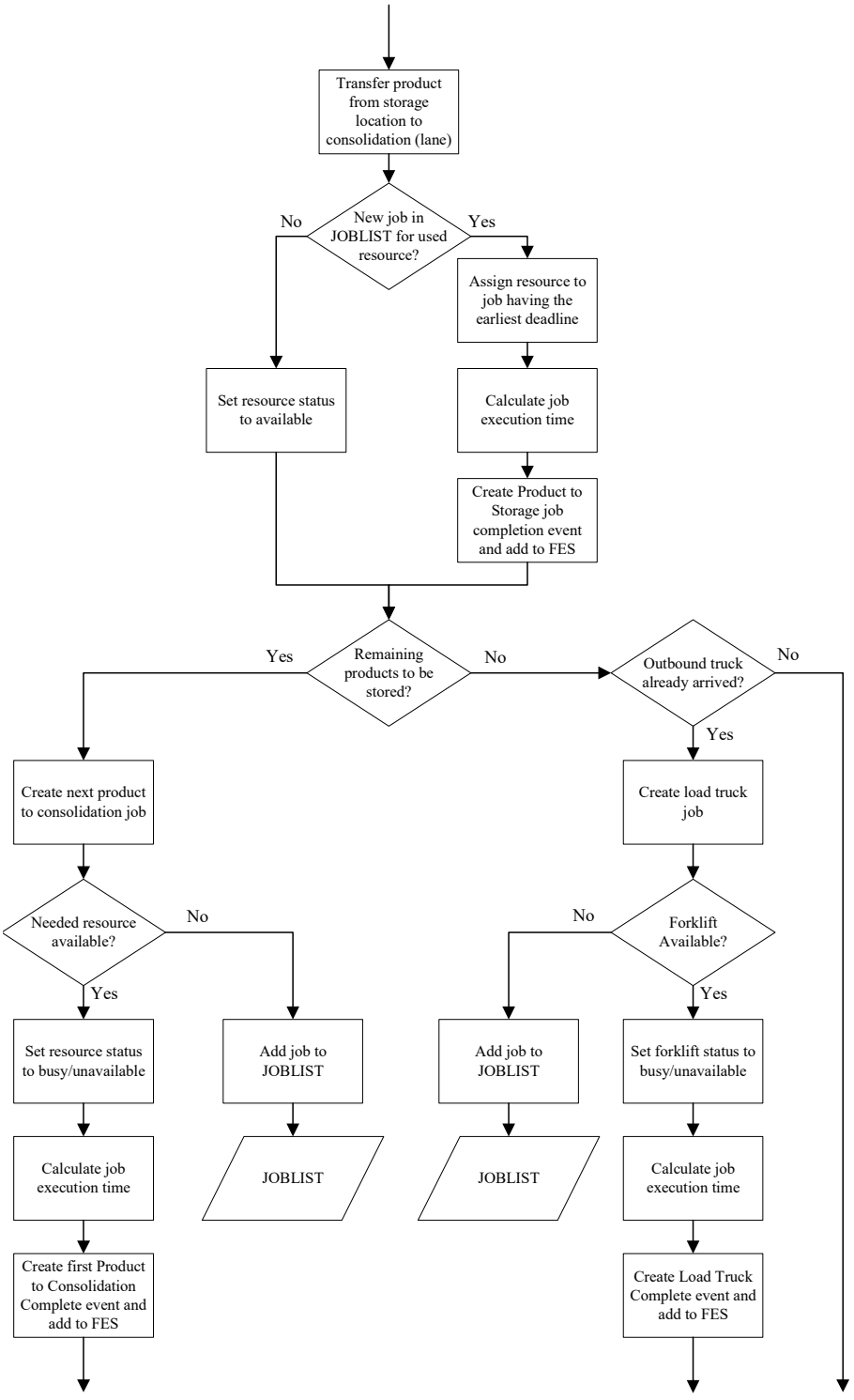


Figure B.8: Flowchart DES - Product to consolidation complete event

B.8 Event 8: Load truck complete

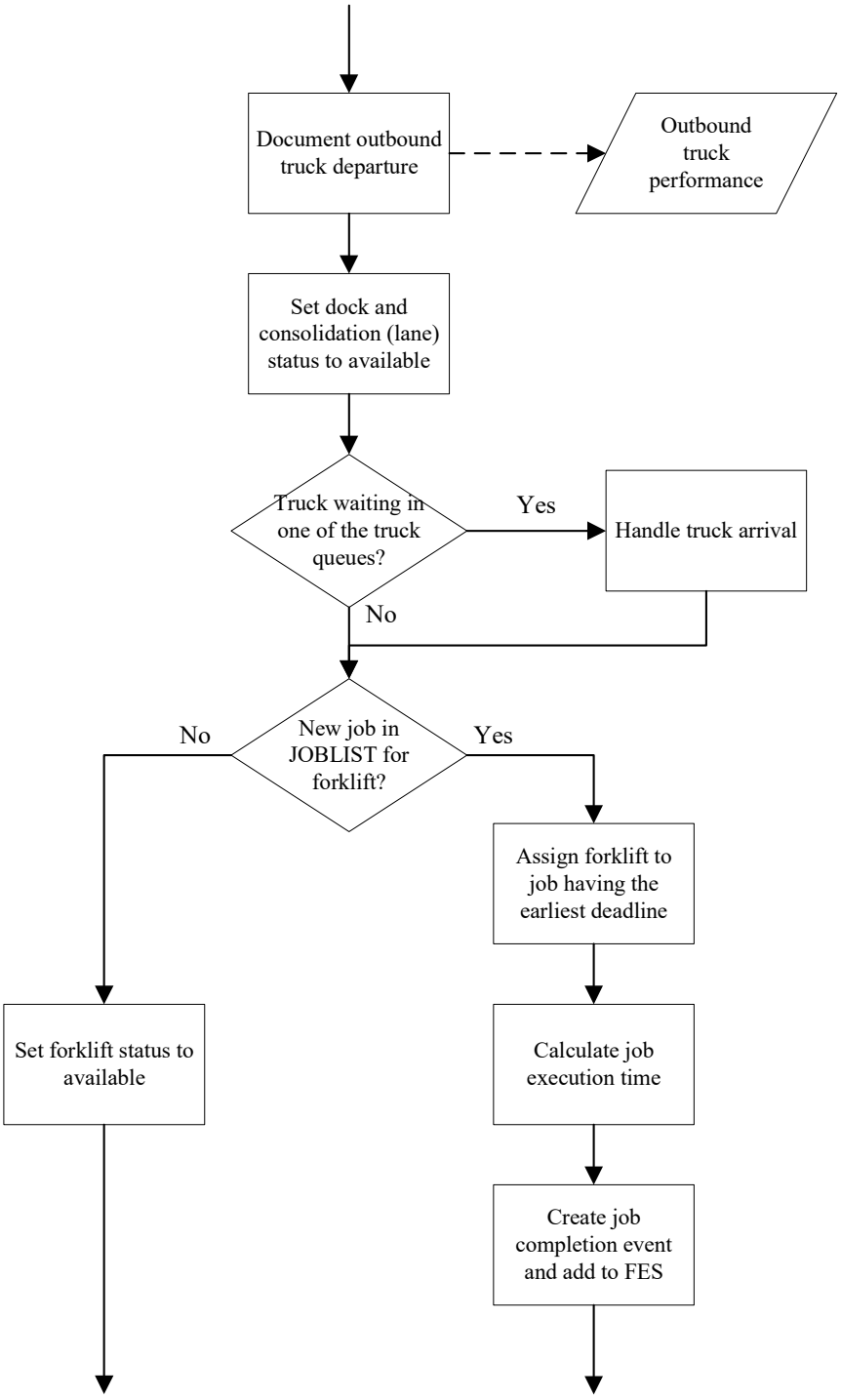


Figure B.9: Flowchart DES - Load (outbound) truck complete event