AALBORG UNIVERSITY
DENMARK

TENTH SEMESTER TECHNICAL REPORT

Task Allocation for Mobile Robot Fleets with Decentralised Genetic Algorithms

Technical report of
ROB10 - gr. 1062

Authors:
Alexander Schøn Staal
Ditte Damgaard Albertsen
Rasmus Finderup Thomsen

Spring 2021

Attributions

This report was typeset using LaTeX, Weights and biases for experiment tracking and data visualisation and Webots for 3D environment construction and validation.

AALBORG UNIVERSITY
DENMARK

Department of Electronic Systems
Robotics

Fredrik Bajers Vej 7
9000 Aalborg
http://www.es.aau.dk

Title:

Task Allocation for Mobile Robot Fleets with Decentralised Genetic Algorithms

Theme:

Master Thesis

Project Period:

Spring 2021

Project Group:

Group 1062

Participants:

Alexander Schøn Staal
Ditte Damgaard Albertsen
Rasmus Finderup Thomsen

Supervisor:

Simon Bøgh

Number of Pages: 65
Date of completion: 03-06-2021

Synopsis:

This project presents the development of an algorithm for assigning tasks to a fleet of decentralised Autonomous Guided Vehicle (AGV)s also known as a fleet manager. The task assignment is done by using a Genetic Algorithm (GA) to handle the multiple travelling salesmen problem (mTSP). The mTSP is a problem definition, which closely resembles the challenge faced in this project thereby making it a prime target to test the GA against. The GA is inspired by evolution and is divided into five parts; initial population, fitness function, selection process, breeding and mutation. These parts are iterated through for several generation. In this case the GA is programmed to optimise the sequence of tasks present in a pick and place environment over time. Furthermore, decentralisation methods are explored for algorithm optimisation, and it was shown that it could be used as such conceptually. The project ended out with a GA that was within 20% of the global minimum 90% of the time. However, the decentralisation did not work as intended the methods were therefore simulated to see the effectiveness.

1 Summary

This project presents the development of an algorithm for solving the challenges involved with setting up a fleet manager to assign tasks to a fleet of decentralized AGVs along with insuring that the sequence in which the tasks occur is optimal. Task assignment is a part of a fleet manager, which can be further split into four stages: The high level fleet manager of which task assignment is a part of, scheduling which schedules tasks on minute to minute basis, an routing algorithm for traffic control and at last a path planning algorithm for generating a path between the different tasks in the environment.

A use case was provided by a company and was used to focus the project by supplying requirement specifications, constraints and a close approximation of a real factory environment constructed for pick and place objectives using Autonomous Guided Vehicles (AGVs). With respect to the use case related works concerned with similar challenges were explored leading to the choice of using a Genetic Algorithm to solve the presented challenge. An genetic algorithm is inspired by the concept of evolutionary theory, mainly the ability of an organism to adapt and improve over time through consecutive generations. In computer science evolution is imitated mathematically by representing the genes of the organism as a sequence of numbers, the order of which represents a solution to specific problem, like deciding in which order a set of cities/tasks should be visited in order to minimise distance travelled. To improve the solution over time a fitness function is included to asses multiple solutions, ordering them from best to worst. Along with the fitness function a selection method is introduced which job it is to select the best solutions for mixing or mating in the case of a biological analogy. The process of mating two solutions to find a better one is referred to as performing a crossover operation since the genes in both parent solutions are inherently crossed to form a new hopefully better solution. At last a process know as mutation is added to the process in order to introduce small tweaks to the new solutions doing runetime since using the crossover operator alone can lead to inadequate solutions. The methods covered so far is what makes up the backbone of a genetic algorithm. Throughout this report the core genetic algorithm was modified in order to accomplish the final goal of developing an Genetic Algorithm (GA) that could solve the challenges presented in this project. While a GA capable of doing tasks allocation between multiple AGVs was created, decentralisation of approach was only tested in concept since a full implementation could not be completed due to a software issue faced late in the project.

Preface

This report was written by Group 1062 of 10th-semester Robotics from Aalborg University. The report focuses on the design, modelling and implementation of a decentralised genetic algorithm for a fleet of Autonomous Guided Vehicles.

Alexander Schøn Staal

Ditte Damgaard Albertsen

Rasmus Finderup Thomsen

Project Structure

The following report is structured in chapters that documents the main components that influenced the development of the decentralised genetic algorithm in this project. The main chapters that form the foundation of this report are Related works, Genetic Algorithm, Problem Formulation and System Design, Modelling and Testing.

Reading directions:

- It is recommended to first go through the list of abbreviations. Abbreviations are defined in the order they appear in the report, and are gathered on page 7.
- The citation style of this report is IEEEtran. Citations are referred to by [1], [2] and correspond to references in the bibliography. The order of the citations is based on their appearance in the report.
- Tables, equations and figures are referenced with numbers related to the order and the chapter in which they appear in.
- The List of Figures/Tables that are not made by the authors of this report, are referenced below them.
- Sources are written in order of appearance throughout the report in the Bibliography.
- Reference list of figures and tables are on the last pages of the report.

Abbreviations List

AGV	Autonomous Guided Vehicle
AI	Artificial intelligence
ANN	Artificial Neural Network
BFS	Breadth-First Search
CNN	Convolutional Neural Network
CNP	Contract Net Protocol
DNN	Deep Neural Network
DPX	Distance Preserving Crossover
DRL	Deep Reinforcement Learning
GA	Genetic Algorithm
GNN	Graph Neural Network
JSSP	Job Shop Scheduling Problem
MB	Market-Based
ML	Machine Learning
mTSP	Multiple Travelling Salesman Problem
RL	Reinforcement Learning
OX1	Order Crossover
PMX	Partially Mapped Crossover
RDO	Research and Development Objective
RODAA	Resource-Oriented, Decentralised Auction Algorithm
TCX	Two-part crossover
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem

Contents

2 Introduction

As a company grows so does its demands for storage space and workers capable of completing the ever increasing number of work orders. However, some work orders/tasks are often tedious or even dangerous for humans to perform, thereby making it a prime candidate for automation. Furthermore, through automation of repeating tasks the efficiency of production can often be drastically increased, which among other benefits drives the automation of many industries in the modern age. [1]

For this project the focus is aimed towards constructing a fleet manager for the allocation of tasks between multiple AGVs working together in an environment resembling an industrial setting. The environment is described in a use case that is used throughout this project as a reference for developing the algorithm that serves as the solution to the fleet management problem. The description of the use case can be found in section 2.1. The project does not undertake the remaining processes involved with task execution as illustrated on Figure 2.1, those being scheduling the time at which the tasks are executed, routing traffic for deadlock avoidance and planning a concise path between tasks. The project will instead solely focus on the effective distribution of the tasks among agents and the order in which they should be completed.

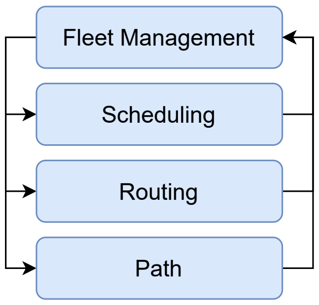


Figure 2.1: The main components of managing a fleet include a high level fleet manager, which assigns tasks to specific agents in the fleet. The tasks are then scheduled on a minute basis, and then a routing algorithm is used for traffic control. At last a path between tasks is planned for each agent.

The challenges of allocating tasks for multiple robots can be classified as a Multiple Travelling Salesman Problem (mTSP)[2], where the goal for each salesmen/robot is to not only select a number relevant tasks with respect to resource constraints, but to also determine the order tasks should be completed, to improve efficiency or decrease distance travelled. To solve the mTSP a branch of search algorithms called GA[3] is used to approximate the most optimal solution to the mTSP. Exploring, setting up and tuning a GA to solve the mTSP and in extension the challenges of fleet management with respect to the use case is the aim of the project. Through this report, the concept and architecture behind GA's is explored and the most suitable candidate for solving the presented problem is selected and modified to suit the needs of the project.

The report is structured as follows. Following this introduction and the definition of the use case the current related works within the branch of research that focuses on genetic algorithms is covered in chapter 3. In chapter 4 the core concepts behind GA's is covered along with different approaches and techniques for constructing the algorithm. Following the chapter, the primary goals and research objectives of the project is covered in chapter 5, which leads into the description of the system architecture for the modified genetic algorithm used to accomplish the goal of the project and the tests performed on the GA in chapter 6. Further discussion of the test results can be found in chapter 7. After the discussion the results of the project is covered in chapter 8, which is followed by a coverage of possible improvements to the algorithm in chapter 9.

2.1 Use Case

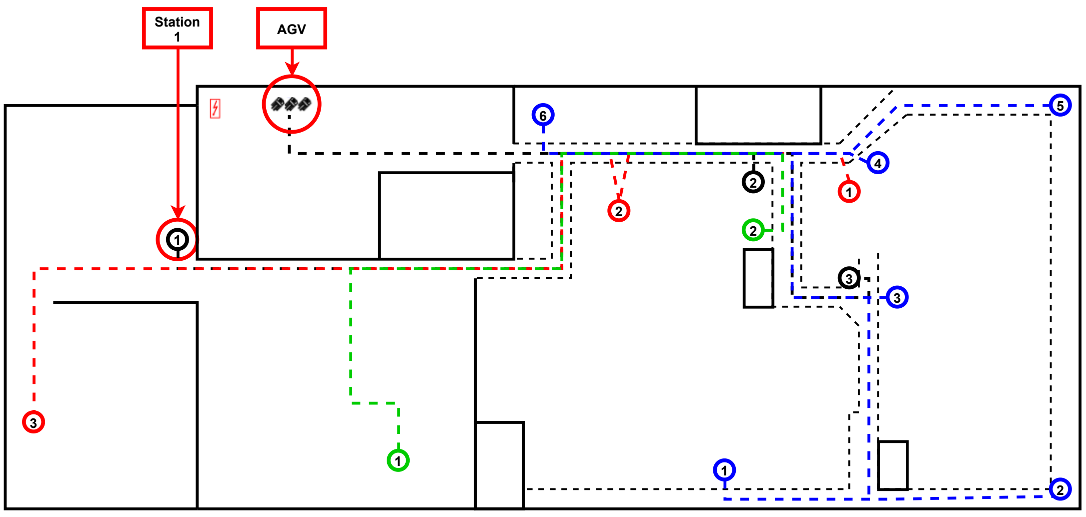


Figure 2.2: Example of a production layout with multiple AGVs, where all the task are transportation of goods between the colour coded stations. The colour of the stations represents which stations are connected and the coloured dotted lines illustrates the travelling route. The small black dotted lines represents the road the AGVs have to travel on in the production hall where most of the stations are positioned.

The illustrated production layout in Figure 2.2 is used to test the result of this project to determine the usefulness on a near realistic case. Three AGVs works together on completing all the tasks in the environment. The tasks in this case are pick and place assignments between the different stations, which are colour coded to represent connected tasks.

A different part of the fleet management system controls the movement of the AGVs, since this project solely focuses on solving the task assignment problem. The output of the task assignment algorithm is a set of ordered tasks, which make up the route to each station. Obstacles, such as other AGVs and static structures are not taken into consideration in this route. The AGVs for the sake of simplicity will move with a constant velocity of $0.7m/s$ To be able to validate the developed algorithm a number of pick and place tasks have been made that resembles that of a realistic production facility setup, these pick and place assignments are defined below:

Red
Station 1 - 2: Move 93 pallets per day
Station 2 - 3: Move 93 pallets per day

Blue
Station 1 - 6: Move 8 pallets per day
Station 2 - 6: Move 6 pallets per day
Station 3 - 6: Move 12 pallet per day
Station 4 - 6: Move 5 pallets per day
Station 5 - 6: Move 3 pallets per day

Green
Station 1 - 2: Move 45 pallets per day

Black
Station 1 - 2: Move 10 pallets per day
Station 1 - 3: Move 2 pallets per day

The solution generated by the GA should be optimised with respect to resource constraints and task completion time. For this project the resource constraints takes the form of the battery level for each AGV, which when fully charged lasts around 8 hours. It is a requirement that each AGV should be set to charge when the battery level reaches 30% to avoid shutting down and blocking a route for other machines. To be effective, the optimised route from the GA must not deviate more than 20% from the global minimum, and must be able to repeat this result 90% of the time. There are in total 277 task to be carried out over 24 hours, which on average is 12 tasks per hour. All tasks in the system are static, meaning that no new tasks are added doing run-time

3 Related Works

This chapter presents the analysis of the problem presented in the introduction and its related works. This related works section is used to explore possible or similar solutions to challenges faced in this project and give an overview of what topics have been investigated in the past. The challenge of assigning multiple mobile robots to do pre-defined tasks can be generalised to other popular research problems. Among these are:

Travelling Salesman Problem (TSP): It considers the challenge of having a given number of n cities, which have to be visited by a salesman once, and finding the shortest path of doing this. Another characteristic is that the salesman has to start and end in the same depot. [4]

Vehicle Routing Problem (VRP): It is a generalisation of the TSP. It defines the problem of finding the shortest route for multiple vehicles, with capacity constraints, to traverse in order to deliver to a given set of customers. All the vehicles have a shared depot, which they return to upon completion. [5] If the capacity of the VRP vehicle is sufficiently large and the size is not restricting, it is basically the same as mTSP.

Multiple Travelling Salesman Problem (mTSP): This is a generalisation of TSP and a relaxation of VRP, where m salesmen have to visit a set number of $n > m$ cities, travelling the shortest path. The individual m_i salesman has, like TSP, a home depot it has to return to. The depot is the same for all $m_1, m_2, ..., m_i$ salesmen. [6]

Job Shop Scheduling Problem (JSSP): It defines the problem where jobs are assigned to resources at particular times. Specifically, a given set of n jobs $J_1, J_2, ..., J_n$ of varying processing times, which have to be scheduled for m machines with varying processing power. The goal is to minimise the makespan, which is the total length of the schedule. [7]

All of the above problem definitions are categorised as combinatorial optimisation problems, with few definitions separating them. However, the most fitting definition for the problem researched in this report is either VRP or mTSP. TSP only applies to a single agent setup, and JSSP mostly focuses on problems with stationary machines and time intervals. On the other hand, VRP and mTSP are both routing problems with many of the same variations. These variations include: [8, 9]

Multiple depots: Instead of one depot, the multi-depot mTSP has a set of depots, where a number of m_j salesmen are assigned at each one j.

Fixed charges: If the number of salesmen/vehicles in a problem is not fixed, then each agent is usually associated with a fixed cost incurring whenever this salesman is used in the solution.

Time windows: Associated with each city/node is a time window during which it must be visited in the tour.

In summery, VRP and mTSP are similar in that they both define routing problems, and have some of the same variations defined for them. The difference is mainly that the plain VRP is a constrained form of mTSP, which is why the challenge in this report is defined as a mTSP from this point onward.

3.1 Market-Based Approaches

The principle behind Market-Based (MB) approaches is based on the idea that multiple agents can, through a common contact or link communication, distribute tasks or resources in order to optimise an overall objective function. The distribution of tasks/resources for this approach is mainly inspired by the concept of auctions, meaning that each agent can bid on a particular task, the bid being a representation of how capable or fit the individual agent is to take and execute the presented task. The fitness of a particular individual can take the form of a cost value that can be calculated based on multiple factors such as distance from the task, battery level in case of a robot application or the current workload already assigned to an agent. After each agent has presented a bid it is up to a central system or a selected agent to act as an auctioneer that selects the best suited bidder for the task [10].

While the underlying concept behind auctions remains the same in MBs, the way is which the auction is designed can wary depending on which architecture best suits the problem at hand. Two auction designs relevant to solving the mTSP are the Contract Net Protocol (CNP) [10] and the Resource-Oriented, Decentralised Auction Algorithm (RODAA) [11]. When using the CNP for solving the mTSP the auction process can be split into four stages, those being the announcement stage, submission stage, selection stage and the contract stage.

1. **Announcement stage:**
   If the mTSP is centralised a central system or network takes up the position of an auctioneer and announces the tasks that are to be distributed, for a decentralised approach the position of the auctioneer is taken by one of the agents.

2. **Submission stage:**
   Here each agents starts by calculating their own fitness which is based on multiple factors represent how suited they are to take up the current task being presented by the auctioneer. Upon finishing the calculation each agent communicates it's fitness/bid to the auctioneer.

3. **Selection stage:**
   After all bids from the agents are received the auctioneer chooses the best suited agent to receive the announced task according to defined optimisation strategy.

4. **Contract stage:**
   In the final stage the selected agent inserts the task into it's schedule and thereafter the four stages are repeated until no tasks remain to be distributed.

The primary benefit of using CNP is the ability to include it in both a centralised and decentralised approach and that the number of agents can be increased and decreased doing run-time. The latter is achievable due to tasks only being distributed among the agents posting bids doing the submission stage. While, the CNP approach is relatively straightforward it is implemented with the assumption that the connection between the auctioneer and bidders are maintained from the initial to final stage. While the way in which the fitness score is calculated and the agent is selected can differ based on the problem being solved, it is assumed that the task being "sold" to the most fitting agent will be executed. The CNP approach is not constructed around the idea that the selected agent could shut down or get stuck, thus resulting in the allocated task not being completed. These issues among others are what form the basics for the RODAA algorithm covered in [11].

Like CNP, RODAA is heavily inspired by the concept of auctions. However, the algorithm is built solely around a decentralised approach to solving the mTSP along with additional constraints. In RODAA the constraints take the form of limited resources, a max time limit on auctions and that the fitness score for each agent is probabilistic. RODAA can, like CNP be broken down into multiple stages/processes, which takes the form of a bid generation stage, multihop auction stage, and a task execution stage.

1. **Bid generation stage:**
   In RODAA the auctioneer is referred to as a customer requesting a service, while the bidders are referred to as labours willing to perform the service. In this stage, the customer requests a service, and the labours within the effective range of the customer calculate a probabilistic value based on its currently available resources. The value represents the individual agents' probability of being able to execute the task.

2. **Multihop auction stage:**
   Doing the bid generation stage, the customer and labours within range construct an ad-hoc network structured as a Breadth-First Search (BFS)-tree for dynamic communication. Doing the multihop auction stage this network is used to effectively allocate the requested service to the best fitting agent. It is important to note that communication with the selected agent is maintained until service completion or service failure due to an exceeded time constraint, agent shut down or other issue. Should a problem occur the customer has the ability to reassign the service to another agent.

3. **Task execution stage:**
   The selected agent monitors its resources throughout the execution stage, after the it receives its task. Should the resources drop below the estimated level the agent

has the ability to, based on probability, decide whether or not to continue executing the assignment or go to a charging station.

While the RODAA algorithm requires additional layers like an ad-hoc network and hard constraints to function as intended, the approach does address many of the challenges inherent in mTSPs. RODAA is especially relevant when physical machines are involved, since it sets up communication between moving agents, calculates the risk of an unscheduled shutdown, deals with uncertainty and the organisation of teamwork. However, as mentioned in [11] RODAA was tested successfully in a simulation environment but is yet to be tested with physical robots.

3.2 Learning-Based Approaches

This section focuses on learning based approaches for solving the challenge present in this project. Generally learning based approaches are referred to as Machine Learning (ML) and are seen as part of Artificial intelligence (AI). ML is defined as computer algorithms which improve automatically through experience. First a model is build based on sample data, also called "training data", in order to make predictions and decisions without being programmed to perform a specific task. Data is introduced to the model iteratively, in order for it to learn and adapt to new situations over time. The ML family contains several subfamilies of algorithms, such as Reinforcement Learning (RL) where the focus is on having the agent take actions which maximise a cumulative reward. Another area is Artificial Neural Network (ANN), which are algorithms designed to simulate the way human brains analyses and processes information. ANN contains three types of layers, an input layer where each component of the state is passed, a hidden layer where the core processing of the input states takes place, and finally an output layer consisting of the actions that can be taken in the environment. If the Neural Network architecture consists of more than one hidden layer, it is called a Deep Neural Network (DNN) [12]. The concepts and methods used for ANN can also be applied or combined with other architectures, such as RL called Deep Reinforcement Learning (DRL). DRL is generally used when the state and action space becomes too large [13]. Other similar approaches include: Convolutional Neural Network (CNN) designed for computer vision, and Graph Neural Network (GNN) designed for working directly on graph structures. However, the mTSP is rarely researched in the machine learning domain, mainly due to difficulties such as the explosive increase in the search space as the number of agents and tasks increase. Furthermore, the lack of data with ground truth optimal solution also presents a problem, since it is computationally expensive and time consuming to get the optimal solution for the mTSP. [14]

In relation to learning based approaches, the most relevant work include a pooling network [15], a structured prediction [16], and a RL approach [14] for solving mTSP.

The pooling network [15] aims to solve the mTSP by trying to minimise the sum of the travelled distance (MinSum). By using pooling operations, the network gets a hidden representation of the path for the k^{th} agent from the i^{th} to the j^{th} vertex. The size of this representation is $m \times n \times n$, where m and n are the number of agents and cities respectively. To decode the representation, they implement beam search under the constraint, that a city (except the depot) is allowed to be visited only once, by exactly one agent.

A hierarchical structured prediction approach for generalisation is introduced in [16]. It focuses on agent-task assignment and works on a grid world environment for up to fifteen tasks and eight agents. However, this task-assignment should be regarded as a relaxation of mTSP, since it only matches cities to agents and does not care about the travel order. Their solution learns pairwise scores between agent-task and task-task on relative small-sized data. The scores are later combined with a quadratic inference procedure to generate solutions. On the other hand, this way of finding the solutions are also computationally expensive. This is especially true when the number of agents or tasks increases, which only allows the approach to be trained and used in small instances.

A reinforcement learning approach for optimising mTSP over graphs is proposed in [14]. They construct an architecture consisting of a shared GNN and a distributed policy network to extract a common policy for mTSP. Furthermore, a two step approach is implemented, where RL is used to learn an allocation of agents to vertices, and a regular optimisation method is used to solve the individual agent's associated TSP. A S-samples batch training method is introduced to reduce the variance of the gradient. Other examples of GNN based solutions are [17] and [18], which both are DRL solutions of the JSSP.

However, out of the solutions mentioned above only [17] focuses on making a decentralised system. This seems to be because the combination of ML and mTSP is a fairly new research area, with papers being published in the last couple of years. In [17], they design a fully distributed learning scheme with a decentralised state and action space and an individual reward structure per agent. Research has been done on decentralised systems without mTSP [19, 20]. A class of decentralised RL algorithms, where a relationship is established between the society (global policy) and the agent (actions) at different levels of abstractions are proposed in [19]. In [20], given a game, each agent knows the system dynamics and their own reward function. Furthermore, each agent receives their own current reward at time step k and is able to deduce the previous controls of other agents but has no access to the current controls, actual rewards, their reward functions or other agents' control laws.

A more common method that is being used to solve the TSP and mTSP is a meta-heuristic method called GA. GA is part of a sub category of AI referred to as combinatorial optimisation algorithms and is inspired by evolution. It works by improving sub-optimal solutions over multiple generations. Each generation consists of a set of solutions to the mTSP where the best solutions among them are combined and mutated to form new ones for the next generation. This process is further explained in Chapter chapter 4.

A GA is developed in [21] as a clustering method for finding the minimum distances between tasks in the different clusters with different starting/end points. This was made in a decentralised system that used a marked-based method to trade tasks between the robots. A solution in [22] that also worked with a decentralised system, used a GA in parallel between the robots, to find the shortest distance while keeping track of completion time. Through this approach the robots would attempt to use their resources in the most optimal way by splitting the computation out and share the findings while executing tasks.

In the paper [6], the GA is being used to optimise task order and allocation, but also to generate the optimal number of agents needed to complete the assigned tasks. The paper also introduces an optimised auction-based task allocation method. The papers [23, 24] focuses on the effectiveness of the GA, where [24] looks at the parameters and how they affect the optimal solution and [23] tests different crossovers, which results in the creation of a new crossover. Along with a new crossover the paper additionally looks into the idea of not initialising the algorithm with a random population.

3.3 Summary

While MB solutions are practical when having multiple decentralised robots in a dynamic environment where tasks are added doing run-time, they struggle when the order of the tasks to be completed plays an important role. This challenge is addressed through the use of combinatorial optimisation methods where a GA is implemented alongside the MB in order to optimise the order in which tasks are completed.

ML solutions are not as researched as other solution methods. This is mainly due to the increase in search space resulting from the size of the state and action space, and the lack of labelled data, since it is computationally expensive to solve for the optimal solution, especially for large task sets. However, some have tried to solve the mTSP with ML methods, among those are multiple GNNs. These methods perform satisfactory or in some cases better than meta-heuristic methods. On the other hand, the better performing GNN method also used regular optimisation methods to solve the individual agent's TSP, while using RL to assign tasks to agents [14]. In terms of decentralised system solutions, it seems to be a an unsaturated research area within RL. However [17] proves it is feasible to solve a decentralised mTSP modelled system.

Within the field of GA there exits multiple approaches for solving the mTSP, thereby leaving room for customisation. It is for this reason along with the the problem statement that GA is the algorithm of choice to solve the task assignment challenge for a decentralised fleet of AGVs. While a combinatorial method can be applied to solve the mTSP it is redundant due to the static nature of the task set, thereby making it more straightforward to find a solution solely using an GA

4 Genetic Algorithm

This chapter looks into the different aspects of the genetic algorithm, and some of the different ways it can be defined in a mTSP context.

When dealing with a combinatorial problem like TSP or mTSP, optimisation algorithms like GA can be used to rapidly compute a solution compared to a brute-force approach. GAs are based on evolutionary theory, where two organisms mix their genes in order to make a new offspring, this process is referred to as breeding. With each new offspring there exits a chance of mutation of its genes. In terms of GA, mutation is defined as either adding to, removing from or switching genes in the gene pool. When applied in computer science, genes takes the form of numbers, the order of which can define a solution to specific problem, like the problem of deciding in which order a set of cities should be visited in order to minimise distance. Compared to a gradient decent algorithm it is the role of breeding to take large steps towards the global minimum while mutation on the other hand is used for small adjustments/steps. In the algorithm mutation is also used to avoid converging to a local minimum. When implementing the algorithm in practice the first set of solutions generated by the GA are sub-optimal. The strength of the algorithm comes from the ability to evaluate the current solutions, select those evaluated to be the most optimal and feed them back into the algorithm in a iterative loop. Each iteration of this loop is referred to as generating a new generation. This process is illustrated in Figure 4.1.

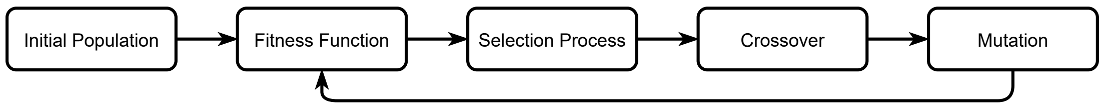


Figure 4.1: A GA can be divided up in five main functions: Initialisation of parameters and the creation of an initial population. The fitness function calculating the individuals fitness score, and based on that the selection function chooses the individuals to be mated in the crossover. The offspring then have a chance of being mutated. This process is then repeated for multiple generations.

4.1 Initial population

A population is defined as a set of solutions in a generation, and in this report a single solution is referred to as an individual. An individual contains genes (tasks), split between different chromosomes (AGVs). There are different techniques for defining an individual in an implementation context. The most common methods includes: One individual technique, two individual technique, two-part individual technique, and multi-chromosome technique.

One individual technique represents a solution for the mTSP using a single individual of length n, where n is a permutation of n genes. The individual is split into m sub-routes by a divider representing the shift from one chromosome to the next. An example of this can be seen in Figure 4.2, where $n = 10$ and $m = 3$. In the example one AGV performs tasks 4, 5 and 7, a second AGV performs tasks 2, 8 and 9, and a third performs tasks 1, 3, 6 and 10. All tasks are performed in the order in which they appear. [25, 26]

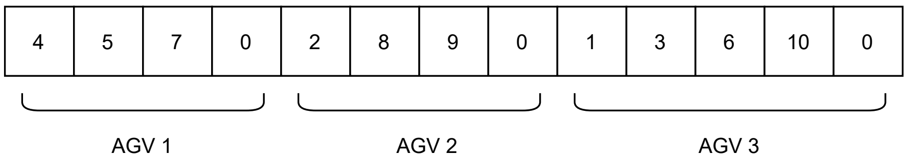


Figure 4.2: In one individual technique an individual contains a list of integers representing specific tasks. A specific AGV's task list is indicated by a divider, which indicate the split from one AGV's task set to another.

Two individual technique uses two individuals of length n to represent a solution. One individual represents a permutation of n genes, while the other indicates the genes assigned to a chromosome, with a value ranging from 1 to m. In the example shown in Figure 4.3 tasks 4, 5, and 7 is performed by the first AGV in that order, tasks 2, 8 and 9 is performed by the second AGV in that order, and tasks 1, 3, 6 and 10 is performed by the third AGV in that order. [25, 26]

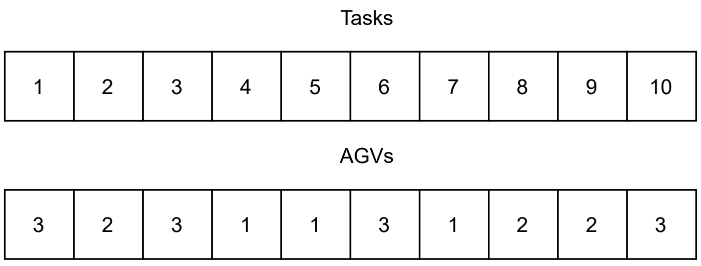


Figure 4.3: Two individual technique employs two lists to describe the assignments of tasks to AGV 1, 2 and 3. The placement of the task corresponds to the specific AGV in the individual containing the AGV list.

The two-part individual technique of length $n+m$ represents a solution in two parts. The first part of the individual of length n contains a permutation of n tasks, and second part of the individual of length m gives the number of tasks assigned to each AGV. The values assigned to the second part of the individual are constrained to be m positive integers, where their sum is equal to the total number of n tasks. For example is Figure 4.4 the first AGV performs tasks 4, 5 and 7, the second performs tasks 2, 8 and 9, and the third performs tasks 1, 3, 6 and 10. [25, 26]

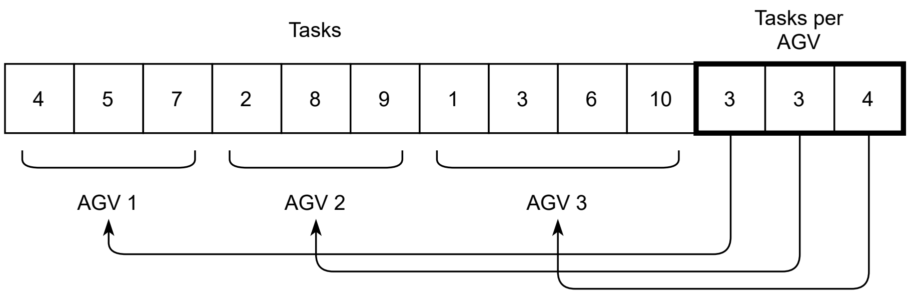


Figure 4.4: The two-part individual technique represents its individuals in two parts. One part indicate the task order, while the other indicates the number of tasks assigned to each AGV.

Multi-chromosome, as the name implies makes use of multiple separated chromosomes to define the solutions for the set of tasks. Here an individual is represented as a set of m routes, i.e. there is no ordering among the routes. The length of a chromosome depends on the number of tasks assigned to the AGV, where the length of the first chromosome is defined as k_1 and the second as k_2, and so on. The sum of the length of the chromosomes must be $\sum_{i=1}^{m} k_i = n$, where n is the total number of tasks to be performed. An example of a multi-chromosome setup can be seen on Figure 4.5, where ten tasks are divided among three AGVs. One AGV is assigned tasks 4, 5 and 7, another is assigned tasks 2, 8 and 9, and the last is assigned tasks 1, 3, 6 and 10. [25, 26]

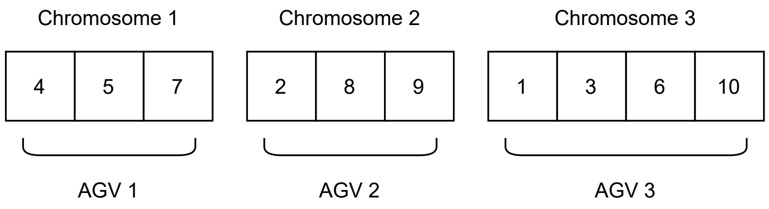


Figure 4.5: Multi-chromosome individuals is a nested list containing multiple lists consisting of the task for each AGV.

All of the above mentioned techniques suffer from redundant solutions, meaning that several individuals represent the same solution. However, the multi-chromosome representation has no redundancy other than the inherent redundancy in representing individual routes. The redundancy mainly occur due to the representation of the mTSP, because routes are represented by linear permutations, whereas in reality they are circular permutations. For example, all the three linear permutations 10, 5, 3; 5, 3, 10 and 3, 10, 5 correspond to the same circular permutation, and hence are the same route. The disadvantage of this redundancy is that the representation space (space containing all possible individuals) is larger that the problem space (space of all possible solutions to the problem). This means that the GA has to search a larger space, since GA works in representation space. This can have consequences for the time it takes to converge.

However, the redundancy is reduced to a minimum for mTSP instance where all AGVs have to start and end at the same task called the home location. [26]

When initialising a population it is important to keep in mind that the diversity of the population should be maintained otherwise it could lead to premature convergence. Furthermore, an optimal population size needs to be found, since a large population size slows down computation, while a small population might not be optimal for a good mating pool. There are two primary methods to initialise a population in a GA. One is random initialisation, where the initial population contains completely random solutions. The other is a heuristic initialisation, where a known heuristic for the problem is used for the initial population. [27] However, when the entire population is initialised using a heuristic, it can result in a population with similar solutions and low diversity. On the other hand, random initialisation effects the initial fitness of the population, but it has a high diversity. Therefore, a population with few heuristic solutions mingled with an otherwise randomly generated population, might be favoured compared to the two extremes. [26, 27, 28]

4.2 Fitness function

The fitness function is defined as a function, which takes in a candidate solution as an input and outputs a score representing how "fit" that particular individual is with respect to the defined problem. This calculation is done for each individual in the population for each generation, and could therefore slow down computation time if not sufficiently setup. [28] An example of a fitness function with respect to minimising the travel distance for multiple AGVs can is seen in Eq. (4.1):

$$fitness = \frac{1}{\sum_{j=1}^{m} \sum_{i=1}^{n_m} (d_{ij}(n_i, n_{i+1}))} \tag{4.1}$$

$$d_{ij}(n_i, n_{i+1}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.2}$$

Where Eq. (4.2) is the distance between the previous and current task defined by an x and y coordinate, m are the AGVs, m_j is a specific AGV, n are the tasks, and n_i is a specific task for AGV m_j. The same concept can be applied to fitness calculations with respect to time.

4.3 Selection process

Selection is the process of selecting parents (individuals) from the current population for the mating pool, based on the fitness values. The selection step is crucial for the convergence rate of the GA, since good parents contain gene combinations, which can be mixed in order to drive individuals towards better solutions. However, it is also a balancing act where good individuals are wanted, but too many of them leads to a loss of diversity, since the same gene pattern occurs in similar solutions. A population populated by extremely fit solutions mostly result in what is known as premature convergence or convergence at a local minimum. The most known selection methods are the ones listed below, where the most used method for mTSP is tournament selection. The selection methods described solve the diversity problem by adding an element of randomness to the selection process, thus insuring a more diverse mating pool. [23, 29, 30, 28]

Roulette Wheel Selection

Roulette Wheel selection is an implementation of fitness proportionate selection. Here each individual is represented with a probabilistic value proportional to the fitness score. The value defines the likelihood of a particular individual being selected. An illustration of this can be seen in the pie chart in Figure 4.6, where ten individuals are weighted based on their fitness scores.

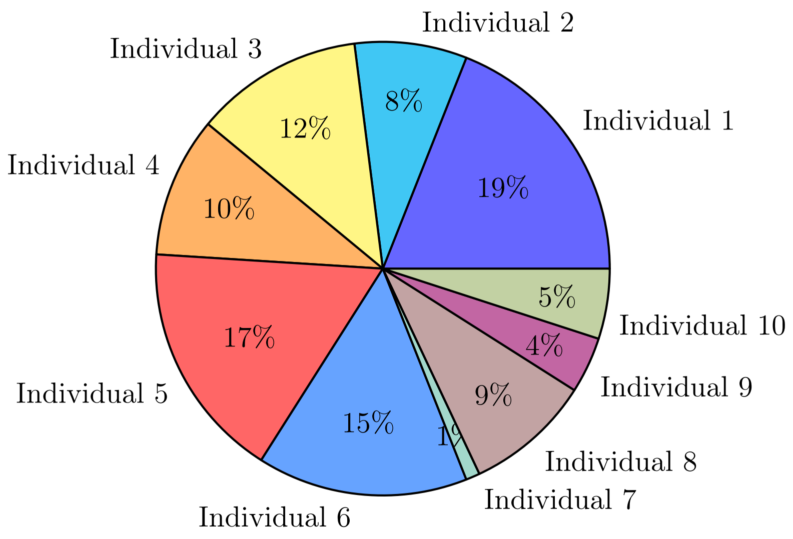


Figure 4.6: Each individual in the population is assigned a probabilistic value proportional to their fitness score. A higher fitness score increases the likelihood of being selected.

Tournament Selection

In tournament selection a set number of randomly chosen individuals from the population are ranked based on their relative fitness score. The best ranked individual among the chosen set is passed to the mating pool. This process can be seen illustrated in Figure 4.7.

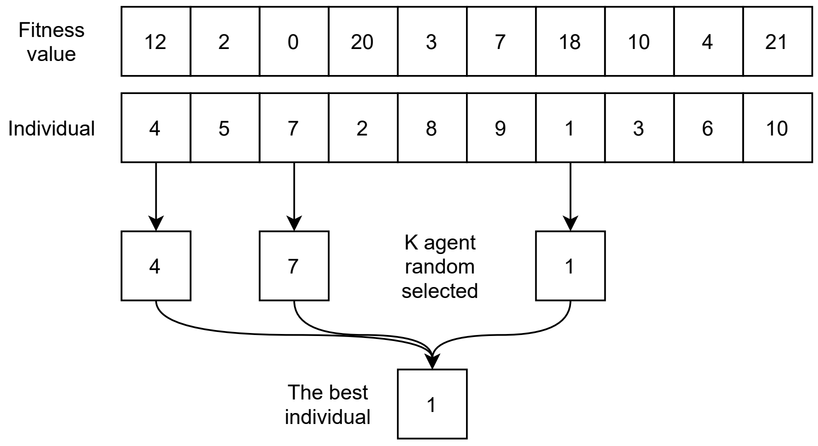


Figure 4.7: Illustration of tournament selection. Based on a defined selection size, a number of individuals are randomly selected for the tournament, where the most fit one carries over. Thus repeats this action until the population is filled again.

Stochastic Tournament

The Stochastic Tournament is a combination of the two selection methods above. Roulette Wheel is used to select the individuals that thereafter are used as the input for tournament selection.

Elitist selection

Elitist selection is used in combination with other selection methods to guarantees that the solution quality obtained by the GA does not decrease from one generation to the next. This is done by allowing a set number of individuals with the best fitness scores from the current population to carry over to the next unaltered. On Figure 4.8 an example of Elitist selection can be seen.

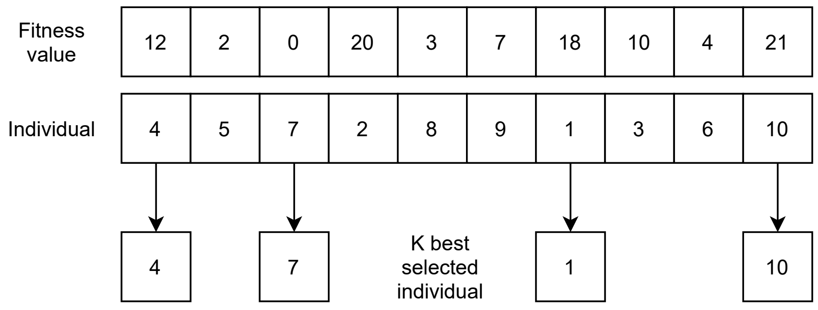


Figure 4.8: In elitist selection, a defined number of the most fit individuals proceed directly to the next generation unaltered. In this example individual 1, 4, 7, and 10, have the best fitness scores, so these individual are guaranteed a spot in the next generation.

4.4 Crossover

The crossover function also known as breeding, has one of the biggest influences on the next generation since its job is to merge two individuals together to form a new, hopefully more optimal solution. This process can be done in numerous ways, each with their own benefits and drawbacks. [31]

Each crossover attempts to mimic nature mathematically by taking some parts of the first individual, and combining it with a second. The two individuals involved in this process is known as a parent pair, and the result of the crossover is referred to as the offspring of parent one and two. The most promising crossover methods found in literature for solving the mTSP are described below and are tested in section 6.4 to find the most suitable candidate for this project's problem definition.

The papers researching solutions for mTSP using GA, used several different crossover methods including: Two-part crossover (TCX)[32, 33, 34, 23], Distance Preserving Crossover (DPX)[35], Order Crossover (OX1)[36, 37], and Partially Mapped Crossover (PMX)[35, 23, 38]. However, among them TCX was mentioned more frequently than the others and described to be the most promising one based on the literature [32, 33, 34, 23].

In order to give an overview and understanding of how the different crossovers works, an example of each of the four crossovers can be seen below.

Two-part crossover (TCX)

The TCX makes one offspring, which can be seen in the illustration in Figure 4.9. The TCX is made in five steps and works as follows, utilises the two-part individual technique shown in Figure 4.4 for defining the individuals:

1. Two parents from the mating pool are chosen, where the first part of the parents is the task sequence, while the second part indicates the number of AGVs and how many tasks they are assigned. An illustration of this type of individual can be seen in Figure 4.4. Figure 4.9 shows this technique in use, where the orange boxes are the second part containing the AGVs. Illustrating that task 1, 2, 8, and 6 is assigned to the first AGV, and the remaining five are assigned to the second AGV for the first parent.

2. A random sequence within the number of tasks assigned to the AGV is chosen (illustrated in blue). The number of chosen tasks per AGV is used as the basis of the task assignment in the offspring in step 5. This is illustrated by the dotted boxes in the figure. For example in step 2 task 1 and 4 is chosen from the first AGV, and task 2, 5 and 9 is chosen from the second, which is two tasks chosen from AGV one and three for the second.

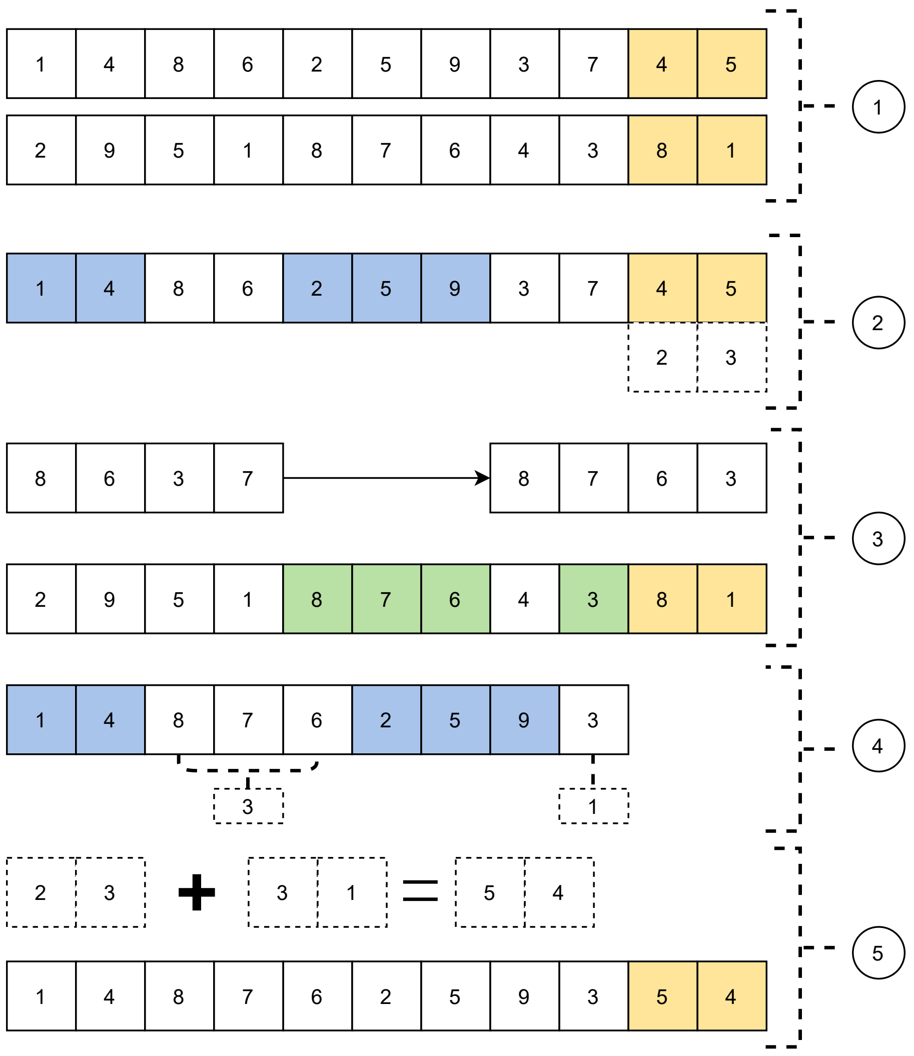


Figure 4.9: The number of AGVs and how many task they have assigned is illustrated with orange boxes and the number inside (task). (1) two parents are selected, (2) a random sequence from each AGV is chosen. Both sequences are coloured blue. (3) the remaining tasks from parent 1 are ordered based on appearance in parent 2, ending up in two sequences coloured green. (4) a new task sequence is constructed based on the previous steps. (5) a number of tasks are assigned to each AGV based on the number of tasks in each sequence constructed in previous steps.

3. The leftover tasks from parent 1 are then ordered in respect to their order of appearance in parent 2. They are then split up into two sequences based on how they are arranged in parent 2. Sequence 1 consist of tasks 8, 7, 6 and the second sequence consists of task 3. Both sequences are represented by the colour green.

4. The sequences that were derived in step 3 (illustrated in green) are used in the off-

spring. The task sequence for the offspring is reconstructed from the first randomly chosen sequence from step two which is coloured in blue. The first green sequence from step 3 is then added. After that the second sequence from step 2 is added, and at last the second green sequence is joined. This makes up the new task list for the offspring.

5. The number of tasks in each blue and green sequence is then combined to define how many tasks each AGV should have.

Distance Preserving Crossover (DPX)

The DPX produces only one offspring, as illustrated in Figure 4.10. As the title implies, the DPX tries to preserve the shortest distances while breeding. The DPX is made in three steps, that works as follows:

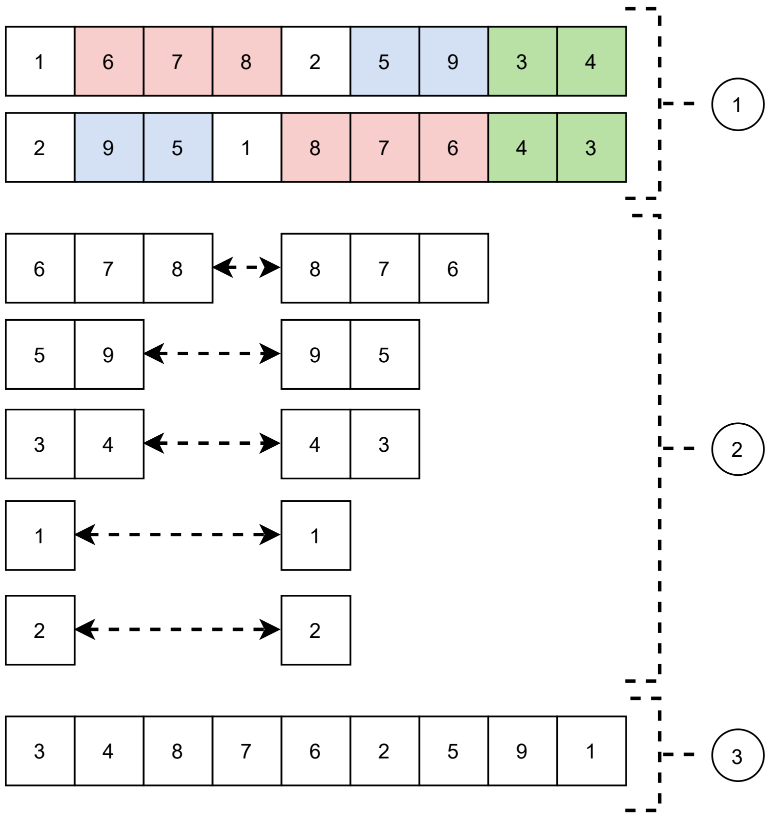


Figure 4.10: Sequences that are present in both parents are illustrated with the colours blue, red and green, where the colourless is defined as remainders. Two arrangements of the sequences can be seen in step 2. A random number is then chosen in step 3 to define a starting point and the rest of the offspring is constructed by adding the sequences with the lowest distance to the end of the first sequence.

1. Two parents from the mating pool are chosen, here the algorithm finds sequences in the first parent that either are the same or mirrored in the other parent. These are illustrated in blue, green and red. The tasks without a connection are defined as remainders.

2. This step shows the different arrangements the sequences can have which will highly influence the shortest distance found as this creates two possibilities to get a short distance.

3. A random number between 1 and 9 is selected to define a starting point of the offspring. In Figure 4.10 the starting number was chosen to be 3 and the sequence with the number 3 as the start will be chosen. The sequence that has a starting number with closest distance to number 4 will be the next one etc. until the offspring is fully defined.

Partially Mapped Crossover (PMX)

The PMX produces two offspring and is fairly simply. An example can be seen in Figure 4.11, that shows it split into three steps. It works as follows:

1. Two parents from the mating pool are be chosen.
2. A random sequence size and position is chosen, which is the same for both parents. This sequence is illustrated with blue. The sequences are then positioned in the same place in the offspring, where they are mirrored. This means that the sequence from parent 1 is used in offspring 2 and vice versa.
3. The rest of the tasks in parent 1 go to offspring 2 in the same order, where the arrows in step 2 show that the two numbers are connected. As shown in the example in Figure 4.11 an already occurring number is changed to its connected number.

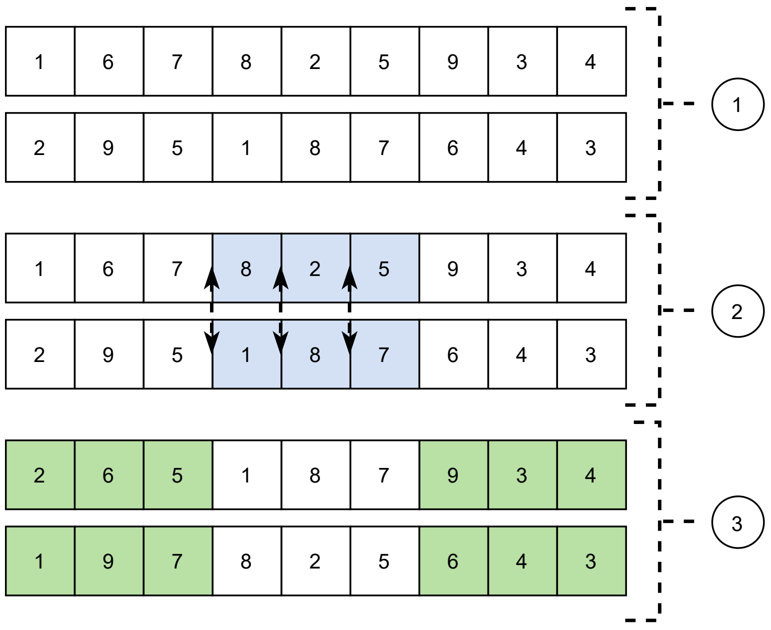


Figure 4.11: The sequences are illustrated with blue, which is moved directly to the offspring where sequence from parent 1 goes to parent 2, while the sequence from parent 2 is inserted into parent 1. The arrows shows connection to be swapped around if a reoccurring number should happen when the rest of the opposite parents tasks are moved to the offspring

Order Crossover (OX1)

The OX1 is derived from OX and does also have an OX2. However, the OX1 is still the most used of these three. An example of the steps making up the OX1 can be seen in Figure 4.12 and is further explained below:

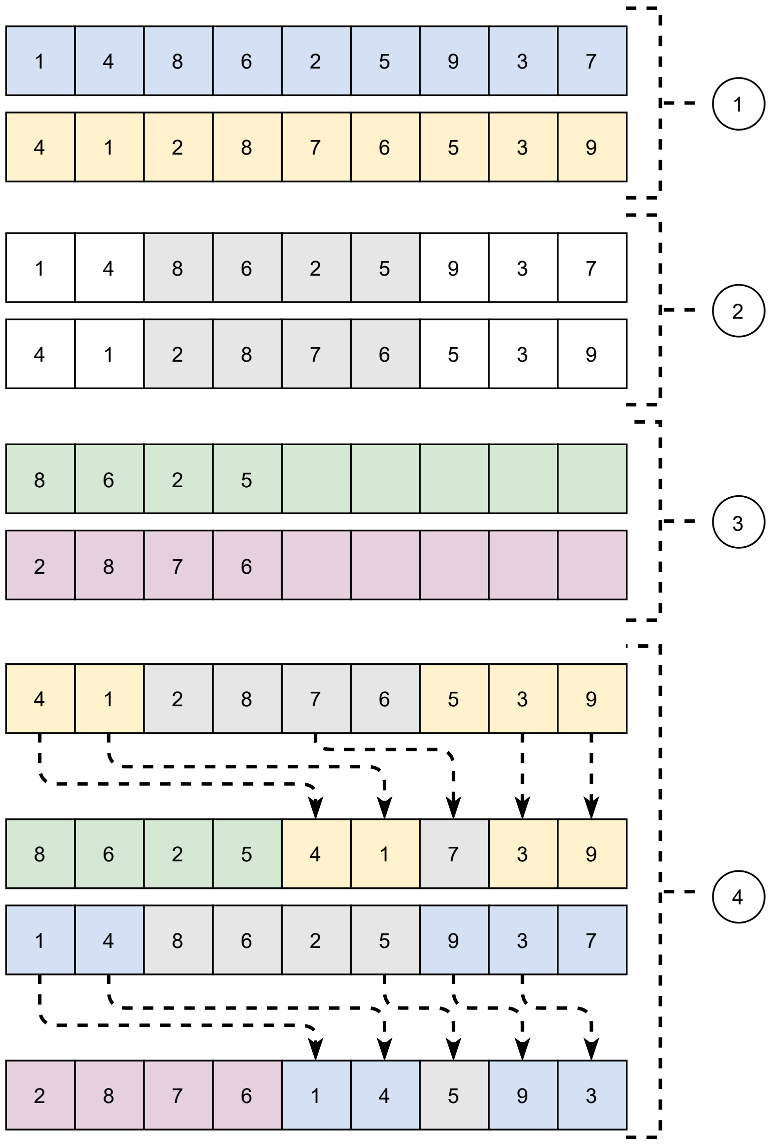


Figure 4.12: OX1 produces two offspring. (1) Two parents are chosen. (2) A random sized sequence and position is chosen. (3) The sequences from parent 1 (blue) and 2 (orange) is defined as the beginning of offspring 1 (green) and 2 (purple). (4) The rest of offspring are constructed, based on the order of the other parent.

1. Two parents from the mating pool is chosen where parent 1 is coloured blue and

parent 2 orange when looking at Figure 4.12.

2. A random sized sequence and position is chosen. The sequence size and position is the same for both parents and is illustrated with grey.

3. The sequences from step 2 is then placed as the start of the each offspring.

4. The remaining missing numbers in offspring 1 (green) is taken from parent 2 (orange) and vice versa for offspring 2 (purple) in the same arrangement they appear in the parent. If the number is already there it will just skip it at take the next number.

4.5 Mutation

Mutation is used to maintain and introduce diversity in the population and is generally defined as a small random tweak in a individual. A balanced mutation probability is important for its usefulness. If the probability is too high, the GA is reduced to a random search. On the other hand, if it is too small it has no impact. Mutation is the part of GA, which is related to the exploration of the search space. On Figure 4.13 to Figure 4.18 some of the most commonly used mutation operators are shown. The figures include examples of how sequence inversion (Figure 4.13), transposition (Figure 4.14), insertion (Figure 4.15), sequence transposition (Figure 4.16), chromosome contraction (Figure 4.17), and chromosome partition works (Figure 4.18). The first half are in-route mutations where the mutations only affect a single chromosome in an individual while the last three are categorised as cross-route mutations that can affect two chromosomes in the individual. Cross-route mutations are especially effective at determining the optimal number of agents needed for performing a certain number of tasks, because they change the task size per AGV. On the other hand, in-route mutations' purpose is to prevent premature convergence by applying randomness, and/or apply small tweaks an AGV's route. [39, 40, 28]

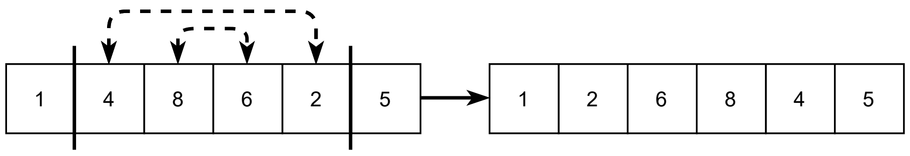


*Figure 4.13: **Sequence inversion:** Randomly selects a sequence of tasks in an chromosome and inverts them.*

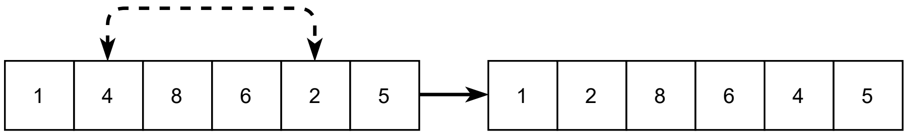


Figure 4.14: ***Transposition:*** *Randomly Swaps two genes.*

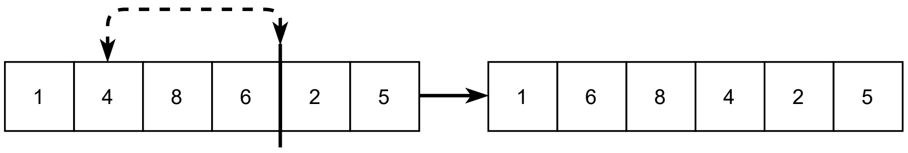


Figure 4.15: ***Insertion:*** *Moves one random task to a random placement in the chromosome.*

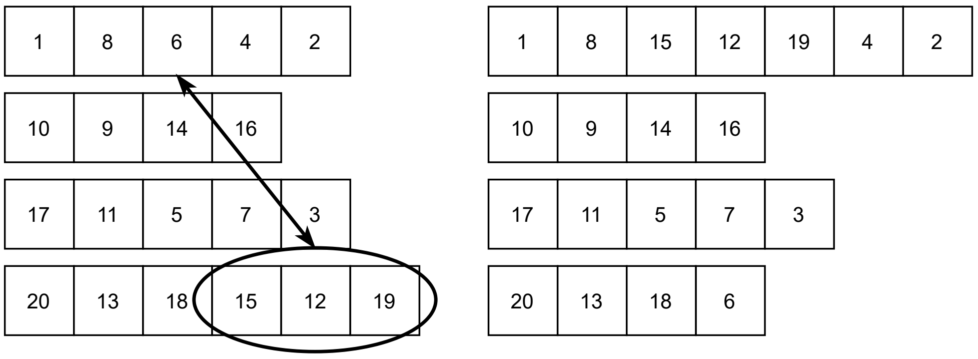


Figure 4.16: ***Sequence transposition:*** *Swaps a random sized sequence of genes from one chromosome with another randomly sized sequence from a second chromosome this can optimise the number of task per AGV*

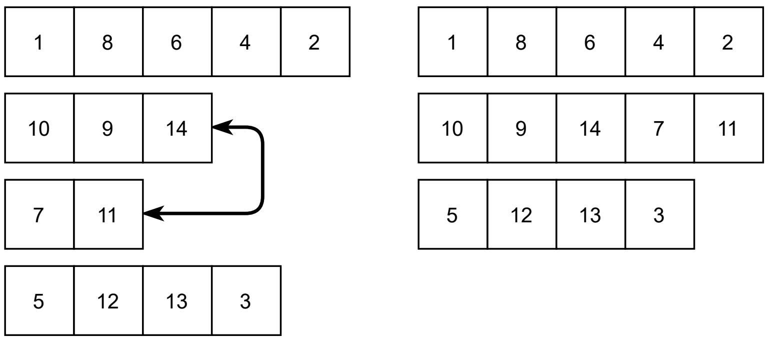


Figure 4.17: ***Chromosome contraction:*** *Merges two random chosen chromosomes together which can be used to test the amount of AGVs needed to complete the tasks*

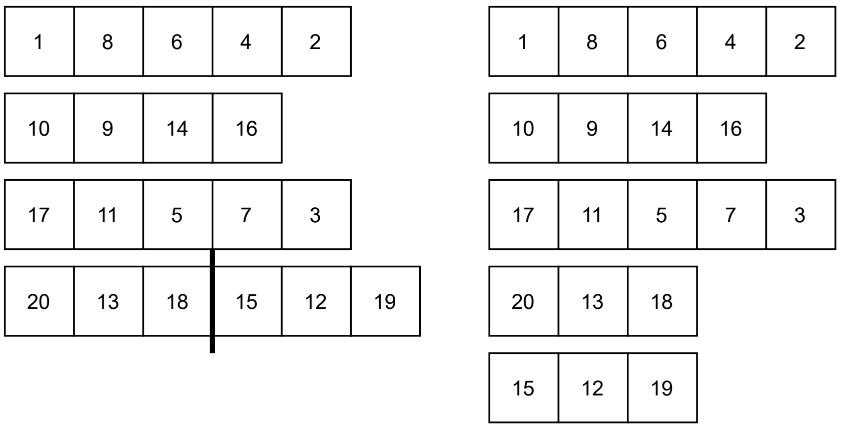


Figure 4.18: ***Chromosome partition:*** *A chromosome is split in half, in a random place to create two new chromosomes. This does the opposite of the chromosome contraction by testing with increasing the amount of AGVs needed.*

4.6 Summary

Knowing GA is the solution method to be used in this project, the general architecture for a mTSP solved with an with GA is explored in this section. The architecture design can be split up into five building blocks: The **initialisation** of the starting population which is either based on random or heuristic generation. **Fitness** calculation and **selection**, where each individual in the population is given a fitness score based how good they are at solving the problem, and depending on that score each individual is either selected for the crossover process or disregarded. **Crossover** is used for combining two solutions with good traits, into a potentially better offspring populating that makes up the next generation. The fifth building block is **mutation**, where a small random tweak is applied to an individual thereby introducing diversity and preventing premature convergence. These five blocks are iterated throughout usually hundreds of generations to optimise, in this case, the order of the tasks assigned to the agents. However, each block can be solved by several different techniques as presented throughout the different sections in this chapter. For example four crossover methods are mentioned, but there exist even more which are not listed, and because of the large impact the crossover technique has on the solution together with the mutation all the crossovers described in this chapter, will be tested to explore which one is best suited for the specific GA constructed in this project. The GA structure described in this chapter is also the one, that forms the basis of the structure developed in chapter 6.

5 Problem Formulation

This chapter focuses on defining the problems, which the project aims to solve. The specific problems are described in the use case in chapter 2, which explains the specifications of the AGVs and the environment in which they work. The requirements defined there are directly used for defining the validation parameters where possible. However, not all the problems stated in chapter 2 can be explored in this report or is relevant for the context. The problems are narrowed down and defined in the problem statement, and further detailed in the research and development objectives. The main take away from the introduction in chapter 2 is the problem that arises when having multiple AGVs, in this case three, working together to solve a pre-defined number of (12) tasks per hour. The AGVs are performing the tasks in a factory environment, where only static structures and other AGVs are present.

The task assignment system should be decentralised as specified by the company providing the use case. The reason for this is the desire for a system, which allows for easy implementation of more or fewer AGVs during the task assignment phase along with providing the ability to select a specific number of AGVs based on the user's needs. As specified in chapter 2 the tasks assignment problem will be solved using a GA. However, other solution methods are also presented in the state of the art analysis in chapter 3 along with the added problem of decentralisation. Among the three methods (marked-based, machine learning and genetic algorithm) MB and GA are the most explored in regards to the classic mTSP. However, among the two MB is more commonly used in a decentralised context, either in combination with other methods or by itself, which makes the decentralisation of GA an interesting research area. Furthermore, the general structure of a GA is described in chapter 4.

The final problem statement is defined based on the problems presented in the introduction and the knowledge gained by analysing related works in chapter 3 together with the general architecture of a GA. The final problem statement is therefore defined as follows:

> *"How can a genetic algorithm be used to solve a task allocation problem for multiple decentralised AGVs transporting goods between several workstations in a static environment, while taking the battery level into consideration?"*

5.1 Research and Development Objectives

The Research and Development Objective (RDO)s described in this section are defined as goals for the research group to work towards and build upon to reach the final objective.

RDO1: Develop a GA framework for a centralised mTSP setup

This first step of the project focuses on the development of a GA framework for solving the mTSP, and is centralised for simplicity, since the general GA framework is also useful in RDO2. This framework is the foundation which is modified throughout the other objectives, and is defined as a mTSP, where $n > 1$ AGVs have to perform $m > n$ tasks. The product of this algorithm is a sequence of tasks to be performed by each agent, and the output is based on the minimisation of the travelled distance for each agent. All the agents start and end at the same station, and after initialising the algorithm no more tasks are added to the task list.

RDO2: Incorporate decentralisation capabilities in the mTSP framework developed in RDO1

The goal of decentralising the system is to reduce the dependency on a central system for task allocation between agents. While the available tasks are still communicated through a central system it is up the agents to determine the most optimal distribution of them. Furthermore since each agent has a copy of the algorithm the estimation process can be run in parallel, thereby improving efficiency.

RDO3: GA-agent resources and constraints design

The goal here is to modify the GA to manage resources, during optimisation of the AGVs' routes. Resources takes the form of power level which is depleted through travelling and task handling. The main problem here is the introduction of a constraint, which is only present in certain scenarios, e.g. low battery for one or more of the AGVs. The charging task is not like the other constant tasks. It is similar in that the low powered AGVs has to travel to and from it, but on the other hand it only has to do so depending on the battery level, meaning that the charging task is only optimised around if a certain condition is met and is therefore mostly removed from general optimisation.

RDO4: Modelling, testing, and validation

The goal is to finish modelling the GA with the most fitting methods for the initial population, fitness calculation, selection process, crossover, and mutation. Furthermore, the hyperparameters are also tested to find the best fit for the developed GA. The GA is validated against the requirements defined below and the problem statement

5.2 Requirements specification and assumptions

The requirements are mainly given from the company who provided the use case in section 2.1, or taken directly from it.

1. The algorithm must find a solution for three decentralised AGVs, which have to perform 12 tasks per hour on average, while driving with a maximum velocity of $0.7m/s$. It is assumed that the task list is static, meaning that no new tasks are introduced during run time. Furthermore, it is also assumed that the path planning and navigation between tasks is handled by separate algorithms. However, the

full extend of requirement cannot be tested, since it requires a good estimate of the execution part of the fleet manager. Instead an estimate of those parameters are given as constants. The route produced by the algorithm is made with the assumption of it being an obstacle free environment. Meaning the distance between tasks is defined as the euclidean distance between the two.

2. The algorithm must when estimating the optimal order and distribution of tasks take the battery level into account. When the battery level reaches less than 30% of the maximum capacity the AGV must recharge. The 30% was required by the company, to make sure that the battery level is large enough to make it to the charging station. It is assumed that all AGVs can recharge at the charging station at the same time.

3. The optimised route from the GA must not deviate more than 20% from the global minimum, and must be able to repeat this result 90% of the time as specified by the company. In other words, the algorithm must have a 80% accuracy and 90% repeatability. The nature of n-hard problems like mTSP makes it next to impossible to brute force the global minimum, but an close estimate is found by running the algorithm several hundred time and use the best minimum found there as the estimated global minimum, if the produced map looks satisfactory.

6 System Design, Modelling and Testing

The system architecture described in this chapter is derived from the knowledge gained in the related works analysis in chapter 3, the understanding of GA solidified in chapter 4, and the requirements defined in chapter 5. Several solutions to similar challenges are explored in chapter 3 for not only the chosen solution method GA, but also machine learning and marked-based approaches. The main take away from chapter 3 is that marked-based and genetic algorithms seem to be more explored than learning-based solutions, and this is mostly prevalent when looking at decentralised solutions. Furthermore, a combination of marked-based and genetic algorithm are also presented as a way to solve the decentralisation problem. Knowing GA is the algorithm of choice chapter 4 describe the general setup of such an algorithm. The five building blocks is presented and examples of implementations in regards to mTSP are given. These five building blocks are also used as the GA's main structure in this implementation which is explained throughout the rest of the report, and can also be seen as the backbone of the overall system architecture shown in Figure 6.1. This chapter include the design, modelling and testing of this project solution to the decentralised mTSP.

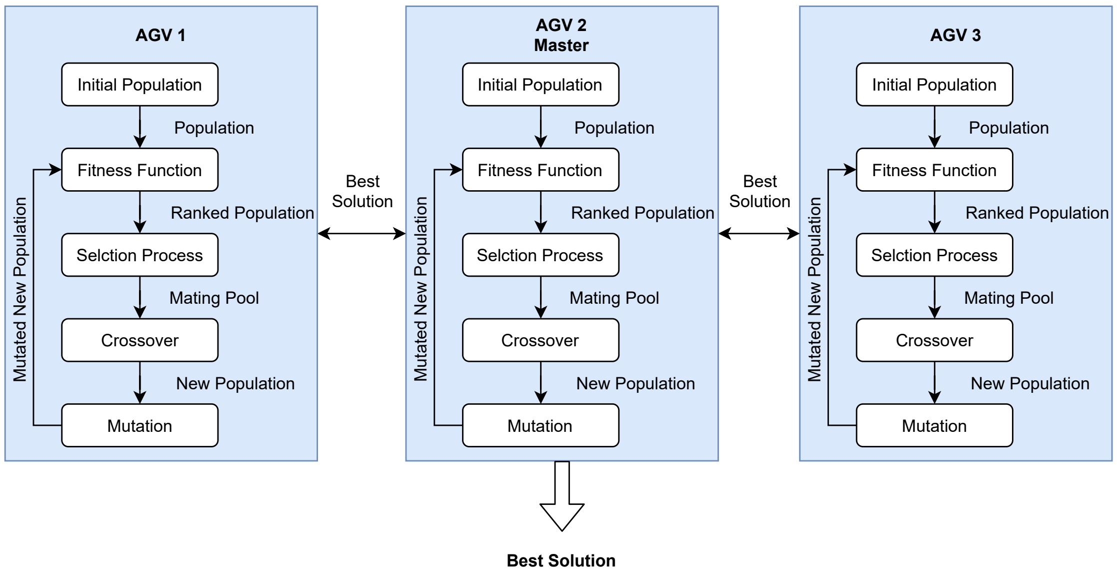


Figure 6.1: The system architecture is build around the GA, which outputs the best estimated solution. The decentralised AGVs communicate with each other and selects a master, which then compare their results and communicates it to the slaves.

Figure 6.1 shows a solution architecture of a decentralised GA. It is of a master/slave structure with the GA as its backbone. A master is chosen to handle the decision making. The AGVs work together to produce an optimal solution, by establishing a communication link sending their results to the master who compared them. The master then outputs the optimised solution, based on the comparison. The optimised solution is obtained by the GA.

6.1 Initial population

A population is a set of solutions to a problem where each new generated population is referred to as the next generation. The initial population defines the initial and future stages of the algorithm, such as how the individuals in the population are constructed, and the size of the population. It can also have an impact on the convergence rate, based on how random the initial population is, but before initialising it the individuals have to be defined. The most common individual constructions are described in section 4.1, and based on the straightforward implementation the multi-chromosome setup is chosen. It is simple to implement and use, since it can be represented as a nested list with Python, where each AGV's task sequence are lists within another list.

Based on the conclusions found in section 4.1, a diverse near random initial population is preferred to hypothetically get the best results. However, in this case a completely random population is preferred, because of the implemented heuristics based crossover (DPX) in this project. The crossover is chosen and described in section 6.4. The chosen crossover uses euclidean distances between tasks to determine the sequence in which the two parents task lists should be combined. However, this also drives the GA to an premature convergence, which results in a need for randomness in the other GA functions, such as the initial population.

Initially a clustering method like k-means was considered as a way to determine the initial number of agents. However, since the use case uses a specific number of agents to arrive at the solution, a way to find the most optimal number of agents is not necessary in this case. Clustering methods for the tasks are more fitting for determining the optimal number of agents, or/and as an initial heuristic for assigning tasks, located in the same area, to a specific number of agents. Furthermore, if this direction was chosen additional mutations has to be implemented too for optimising it, to make sure other agent sizes are tried out, as k-means can only be used as an initial estimate.

The population size is one of the most important parameters to consider in a GA. Usually a "small" population size is said to lack diversity and so guide the algorithm to a poor solution. On the other hand, a "large" population size could make the algorithm expend more computation time to find a solution. A middle ground has to be found, where the trade-off is considered, so that the population size has just "enough" individuals to produce a "good" solution expending the least amount of computation time. There are various methods for estimating the optimal population size. Usually the best population size is estimated by using the empirical method, which entails testing the algorithm with different sizes. The result of this can be seen in Table 6.6, where different crossover methods is tested with various parameters and population sizes of 50 and 100. Those sizes are chosen based on initial testing results, and they show that the best fitting population size depends on the crossover. The main difference between the two sizes is the computation time. In regards to the chosen crossover DPX 50 is concluded to be the most fitting population size. However, as mentioned above DPX has a fast convergence rate, and a possible solution to that is tested out by initialising the GA with an initially large population of 250, 500, 750, and 1000. It is theorised that by having a larger initial population there is a bigger change of getting more diverse individuals and for the population to contain better individuals in that initial population.

Initial Population Size	Initial Selection Size	Mean Distance	Mean Time (s)
250	50	3080	206
500	50	3142	232
750	50	3193	357
1000	50	3117	532
250	100	3109	151
500	100	3121	249
750	100	3202	355
1000	100	3122	549
250	150	3122	183
500	150	3172	233
750	150	3152	352
1000	150	3136	529
250	200	3151	174
500	200	3147	250
750	200	3119	411
1000	200	3138	577

Table 6.1: The results were generated by running the algorithm with the different combinations of initial population and selection size in the first generation ten times, with the centralised version of the GA. The best combinations are highlighted by green

It is tested by running the first generation with a population size of 250, 500, 750 and 1000, and initial selection sizes of 50, 100, 150, and 200 in the centralised setup of the algorithm. After running the first generation with those parameters they are reset to the optimal population and selection size of 50 and 5 found in section 6.5. The algorithm is run ten times for each combination of population and selection size. The initial selection size is tested along the initial population size, since the optimal selection size is heavily affected by the population size. A detailed description of the implemented selection method and its parameters can be seen in section 6.3. The results of the test can be seen in Table 6.1. These results has to outperform the baseline shown in Table 6.7, since it is tested with the same parameters beside the initially large population and selection size. Both are tested with a population size of 50, selection and elite size of 5 and 15 respectively, and a mutation rate of 90 %.

The results of the test are shown in Table 6.1, with the best combination of parameters highlighted in green. As expected there is a correlation between a high initial population size and a rise in computation time. The computation time used with an initial population size of 250, 500, 750 and 1000 is 178.5, 241, 368.75, and 546.75 on average respectfully.

Compared to the baseline this increase in computation time does not yield a significant lowered distance. Weighting a low mean distance higher than a low computation time means that the best result produced by the introduction of an initially large population is a mean distance of 3080 with a population size of 250, and computation time of 206. However, compared to the baseline it does not show a significant difference. The baseline has a mean distance of 3120.75 and an average time of 179.60, which gives a positive difference of 19.25 at the cost of 6.4 seconds. However, that difference is not deemed significant enough to justify the added complexity added to the code to arrive at that result, especially not with the implemented decentralisation method described in section 6.6, increases accuracy and repeatability while maintaining a relatively low run-time. Same can be concluded of the other combinations of population and selection size.

In summary, using the multi-chromosome technique for constructing individuals a population of 50 individuals is randomly generated, based on the algorithm shown in algorithm 1. The algorithm outputs a randomly generated population, based on the parameters population size, number of AGVs, and the tasks that need to be performed. New individuals are generated and what they contain is based on the number of AGVs and the number of tasks which should be assigned to each AGV. The number of individuals generated depends on the population size. After generating the population the individuals have to be evaluated to see if they are fit to produce newer and better solutions. The individuals are evaluated using a fitness function, which can be seen in the section 6.2.

Algorithm 1: The algorithm outputs the population based on the wanted population size, the tasks to be performed, and the number of AGVs.

```
Input  : Population size, task list and number of AGVs
Output: Initial population
Number of tasks for each AGV = TaskList/number of AGVs
for number of individuals wanted in population do
    for number of AGVs - 1 do
        for number of tasks for each AGV do
            Selected_task = randomly selected task from task set
            Append Selected_task to temporary list and remove from task set
        end
        Append temporary list to individual
        Reset the temporary list
    end
    Append individual to population
    Reset individual
    return Population
end
```

6.2 Fitness function

The fitness function is used to determine how good an individual is at solving the defined problem. A high fitness score is preferred, meaning an individual with a higher fitness score than another is considered to be a better solution candidate. The fitness score is calculated by a fitness function, which is designed based on a goal. In this case, the goal is to find the shortest combined path all AGVs have to travel to perform all tasks. Furthermore, each task must only be performed the exact number of times it is present in the task list. With the problem in mind a fitness function is designed. In this case either total distance or time is a fitting measure of fitness, since the minimised time spent or distance travelled both solve the same problem. Distance is chosen as the fitness measure in this algorithm, since distance calculation has to be implemented in both cases, so one might as well use distance alone for it, without the added calculation of travel time. This leads to the definition of the fitness function, which is the sum of the distance travelled for each agent. The equation can be seen in Equation 4.1, and the implemented algorithm can be seen in algorithm 2. The algorithm takes in the current population as an input, and outputs a sorted list, the same size as the population, of indexes associated with a specific individual in the population and its fitness value. The fitness score is calculated trough several steps. First the euclidean distance between the individual's current task and its next, based on that the fitness value is calculated by taking the inverse of the total distance, since a high fitness score has to represent a low total distance . After calculating the fitness score it is stored in a list variable together with its associated index number describing which individual it belongs to. This is done for all individuals in the population.

With the fitness scores calculated it can from then on be used to determine, which individual contains the better sequence of tasks compared to others. The method in which these individuals are selected can be seen in the next section (section 6.3).

Algorithm 2: The fitness function takes in the current population and the home_station, and outputs a sorted list containing an individual index and its corresponding fitness value

```
Input  : Current population and home_station
Output: List of sorted indexes of each individual's fitness
for number of individuals in population do
    Calculate euclidean distance between current task and the next
    Calculate fitness
    Store fitness in FitnessList
end
return Sort FitnessList based on highest fitness score
```

6.2.1 Battery constraint concept

The use case described in section 2.1, specify a need for battery constrained AGVs. An algorithmic concept solution to this is given in algorithm 3 as an altered fitness function, where a charging task is added to an AGV's task list if a battery threshold is exceeded. The battery threshold is defined to be 30% in the use case. However, it was not possible to get the specific number of this in regards to travel distance, instead an arbitrary threshold of 1500 is given. This means that an AGV can travel a distance of 1500 before reaching the "battery threshold", after which a charging task is added. The fitness function is most suited for the implementation of the constrain, since it already goes through every AGV and calculates its travel distance iteratively, which is the same process needed for determining where a charging task should be introduced. Furthermore, the charging task placement in the task list cannot be optimised in the same manner as a normal task, since an AGV is required to charge if it reaches the battery threshold. This makes it ideal to implement in the fitness function, since after placing the charging task the fitness score can be directly calculated. The charging task can then be removed, so its placement cannot change. This method is the same one used for taking the home station into account, when calculating an AGV's total travel distance.

Algorithm 3: The fitness function takes in the current population and the home_station, and outputs a sorted list containing an individual index and its corresponding fitness value. If the battery threshold is exceeded a charging task is added to the AGV where the threshold was exceeded in the task list.

```
Input  : Current population and home_station
Output: List of sorted indexes of each individual's fitness
for number of individuals in population do
    Calculate euclidean distance between current task and the next
    if Combined distance of an AGV > battery threshold
        Add charging task to AGV's task list at the point where the threshold
         was exceeded
    end
    Calculate fitness
    Store fitness in FitnessList
    Remove charging task from AGV's task list
end
return Sort FitnessList based on highest fitness score
```

6.3 Selection

The selection function takes the fitness score from the fitness functions into account when selecting the individuals for the next generation. The selection function can be done with different methods as explained in section 4.3, but in overall the fitness scores is being used to define how big of a likelihood an individual has to be chosen as a parent. An hyperparameter is here used to define a specific number of the best individuals, called elites which will be carried over to the next generation without being subject to a crossover operation. Out of the elites the best one is chosen and is in this project referred to as a king which makes sure that the best solution is saved until a new king is found. The tournament selection and roulette wheel method were implemented along with the initial setup of the GA. Here the tournament selection showed to generally give the best results for the use case of this project along with being highly customizable and straightforward to implement, so no further testing were performed. The tournament selection will therefore be the chosen selection method for this project. The tournament selection process can be seen illustrated in Figure 4.7 and further details can be found in section 4.3. The implemented code can be seen in algorithm 4, where the selection is defined as a function, that takes the Population, the ranked population and the predetermined hyper parameters elite size and selection size as input. A selection pool is made to hold a copy of a randomly mixed ranked population list, where the selection pool is then used to make a *temporary_sub_list* in the second *for loop* of a random sample with the size of the Selection Size. The one with the highest score is then found and added to *selectionResults*. In the first *for loop* the elites are added to a list called *selectionResults* which will be filled in the second *for loop* with the rest of the tournament winners until the specified population size has been reached.

Algorithm 4: The Selection function uses the ranked list to generate the elites and the rest of the list is found by taking the individual with the highest score and add it to the selectionResult which will be passed along to the crossover.

```
Input  : Population, Ranked Population, Elite Size and Selection Size
Output: List of the elites and all the tournament winners
selectionResult list;
A copy of a random mixed Ranked Population is added to selection_pool;
for number of Elite_Size do
    Adds elites to selectionResult;
end
for Number of Population without Elites do
    Temp_list = random_sample(of selection_pool, with a selections size);
    Highest in Temp_list is added to selectionResult;
end
return selectionResult
```

6.4 Crossover

A GA optimises a solution by running through multiple generations of evaluating individuals, and based on that selecting the most fit individuals. The selected individuals are used to produce new and hopefully better individuals by combining them, which is the job of the crossover method. Four crossover techniques were presented in section 4.4, and are tested in this section. To get comparable results the different combinations of parameters, such as elite size, population size, and selection size, are tested along with the four crossovers. Those parameters are tested along with the crossover, since they can heavily influence the performance. The crossover with the best result is used for further testing with in mutation section 6.5 and then decentralisation, which is detailed in section 6.6.

All the crossover tests are repeated ten times in order to see identify a pattern and get an average. One of the hyper parameters tested are population sizes of 50 and 100, meaning 50 or 100 individuals are generated each generation. The two sizes were chosen, based on initial testing which showed that a population size of 150 could have a computation time as long as half an hour for no significant lowered mean distance. The elite size is also tested with the other parameters in an interval of $5-10-15$. The elite size control how many individuals go through the crossover unaltered. The last parameter to be tested along with the others is the selection size, which is the sample size of the population from where the most fit individual is selected. No mutation is used in order to note how the crossover methods perform without, such that the results with mutation can be compared later. The tests are done for 25 tasks, three AGVs, and map size of $500x500$. The tasks placements are randomly-generated within the map, and stay the same throughout all the tests. A parameter called *breakpoint* is introduced to be able to use computation time as a comparable variable, when discussing the different crossover methods. The *breakpoint* terminates the algorithm if the best optimised route has stayed the exact same a defined number of generations. By having the *breakpoint* the results also show how fast the algorithm is at optimising the route, instead of only showing how fast the algorithm is in general. The tests are performed with a *breakpoint* of 100, which was found to be satisfactory during initial testing, since in most cases the best individual would not change from that point onward. The four crossovers that are tested are detailed in section 4.4. The best crossover is chosen based on its ability to produce the lowest distanced route repetitively, using minimal computation time through multiple generations. In order to evaluate how good the crossovers are they should be compared to the global minimum. However, the method of brute forcing the best solution by calculating the travel distance for all possible permutations of routes is not possible, since number of permutations for 25 tasks is equal to 25! which might take way to long. This is why combinatorial optimisation techniques are preferred. Instead an estimate of the global minimum is found by running the algorithm repeatedly, with the various crossovers and noting the best route for each iteration. The global minimum is then estimated to be the best route found, and the result of this is shown in Figure 6.2 with the accompanying distance of 2744.

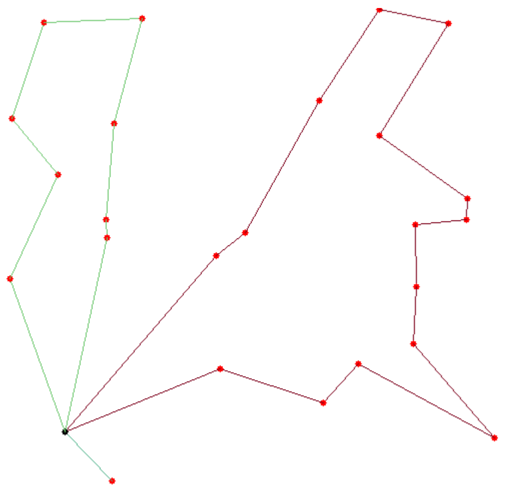
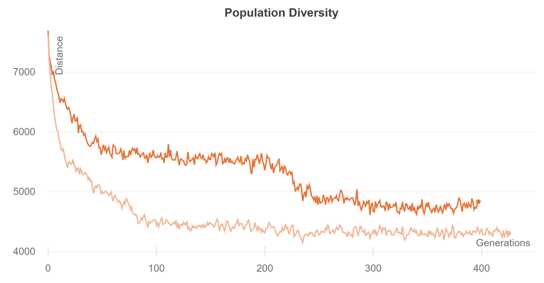


Figure 6.2: An illustration of the estimated shortest route of 2744 consisting of 25 tasks divided between three agents. The three agents are each defined by a different colour.

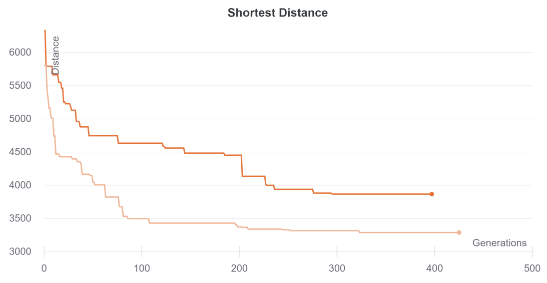
Two-part crossover (TCX)

The TCX is noted to be the most promising crossover among the four, based on the analysis in section 4.4, and should be able to produce a consistent low distance route compared to the others. In Figure 6.3b, two examples out of 180 tests showcase the general behaviour the crossover produces, which is a decrease of the shortest distance and an even diversity in the population throughout the process. The dark orange coloured run is the average representation of the expected outcome, where the decrease in distance happens over time in multiple steps. The light brown, on the other hand, showcases a "lucky" run, where a desirable sequence is discovered early in the process. However, progress in the lucky run is sparse and not as significant as the average run, this is most likely due to the chance of finding a shorter path decreasing when closer to the global minimum.

The dark orange is the most common result of the TCX, where it gradually finds a better solution over the course of generations and the outlier is one of the lucky runs that randomly generate a good solution early on and then has small less frequent improvements, but this does not happen often enough to be reliable. In Figure 6.3a the population diversity can be seen showing each population tries out both bad and good combinations.



(a)



(b)

Figure 6.3: (a) visualises an example on how the population diversity of the TCX through each generation and where (b) shows the Shortest distance that gradually decreases till a minimum has been found

Elite size	Selection size	Population size	Mean distance	Time (s)	Population size	Mean distance	Time (s)
5	5	50	4196.06089	152.5	100	3745.53	651
5	10	50	3912.022416	180.9	100	3813.084	615
5	15	50	3922.991023	156.5	100	3751.892	609.8
10	5	50	3937.134656	168.8	100	3771.317	784.9
10	10	50	3943.845701	159.7	100	3672.426	648.1
10	15	50	3991.946112	179.7	100	3651.103	964.1
15	5	50	3834.136248	160.4	100	3679.085	577
15	10	50	3749.840178	164.5	100	3736.728	560.4
15	15	50	3872.362693	175.6	100	3705.333	757.5

Table 6.2: An overview of the results, comprised to a give a better perspective on how the different combinations perform. The best results are marked in green for both populations

The different mean results of the tuning test for the population size of 50 and 100 can be seen in Table 6.2, where the ones marked in green are those with the parameters that created the best results and are those used in section 6.5 to fine tune the mutation parameter. It can be seen together with Figure 6.4 that the results are concentrated between 3700 and 4000, which is roughly 1000 meter longer routes compared to the global minimum.

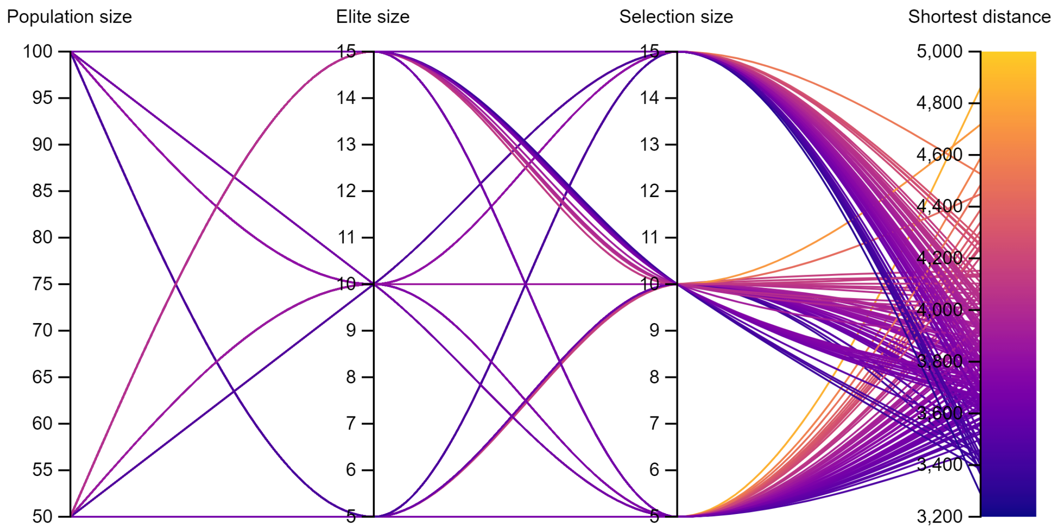


Figure 6.4: The TCX is tested with the experiment parameters, and the different combinations' results are shown on the right. The solutions generated by the crossover mostly varies between the distance values 3,400 - 4,400. [41]

Distance Preserving Crossover (DPX)

The DPX as the name applies produces a result with respect to a distance heuristic, however from the algorithm in section 4.4 it can be expected that very few changes will happen after a while as there is only limited solutions the population can have without the mutation. As can be seen on Figure 6.5a, the population deviation goes rapidly to a local minimum without any further changes. This also keeps the local shortest distance until the break point is met, as can be seen in Figure 6.5b.

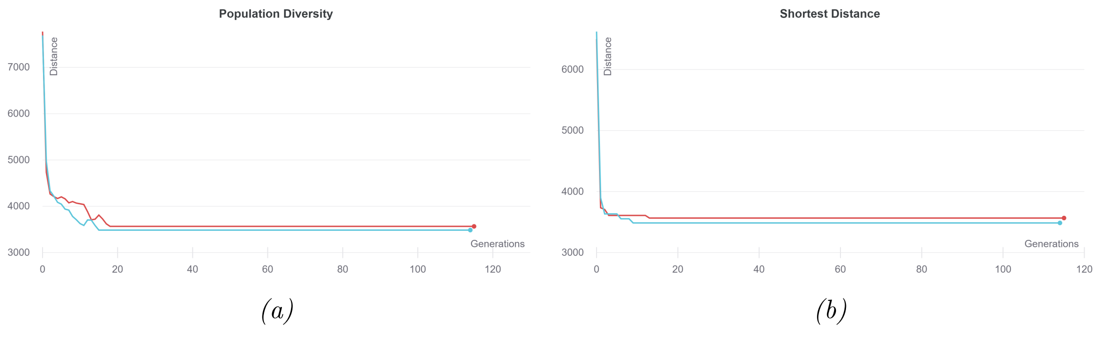


Figure 6.5: (a) shows the population diversity does not change much after 20 generations where (b) shows the Shortest distance are found equally quickly and stays with it to the break point is met. [41]

The two examples from Figure 6.5b and Figure 6.5a, show one of the most common results (red) and one outlier (blue). The differences are not that significant and are within a distance of 100. This is also supportet by the data tabulated in Table 6.3 where both test with a population on 50 and 100 does not change that much in distance. The only real difference is the computation time, which is to be expected by the difference in population sizes.

Elite size	Selection size	Population size	Mean distance	Time (s)	Population size	Mean distance	Time (s)
5	5	50	3623.912	70	100	3578.326	278.6
5	10	50	3626.91	69.5	100	3564.422	280.2
5	15	50	3624.381	70.5	100	3564.902	277
10	5	50	3614.952	72.3	100	3586.846	307.3
10	10	50	3600.094	71	100	3586.785	272.1
10	15	50	3609.714	70.8	100	3573.452	274.7
15	5	50	3589.424	70.7	100	3568.154	275.9
15	10	50	3631.004	72.6	100	3590.447	281.1
15	15	50	3615.924	71.9	100	3589.836	277.8

Table 6.3: The DPX's results does not differ that much from each other, however the best tuning parameters are marked with green for each population size.

The best test tuning parameters can be seen in Table 6.3 highlighted with green, these two result are the ones that will be tested along with mutation which might have a positive influence on the results by making some small changes, thus decreasing the chance of being stuck in a local minimum. In Figure 6.6 all the results from the test are visualised. It shows that the results are very distorted which might be due to the algorithm rapidly finding one of the local minimums. All the results are place between 3,400 - 3,800.

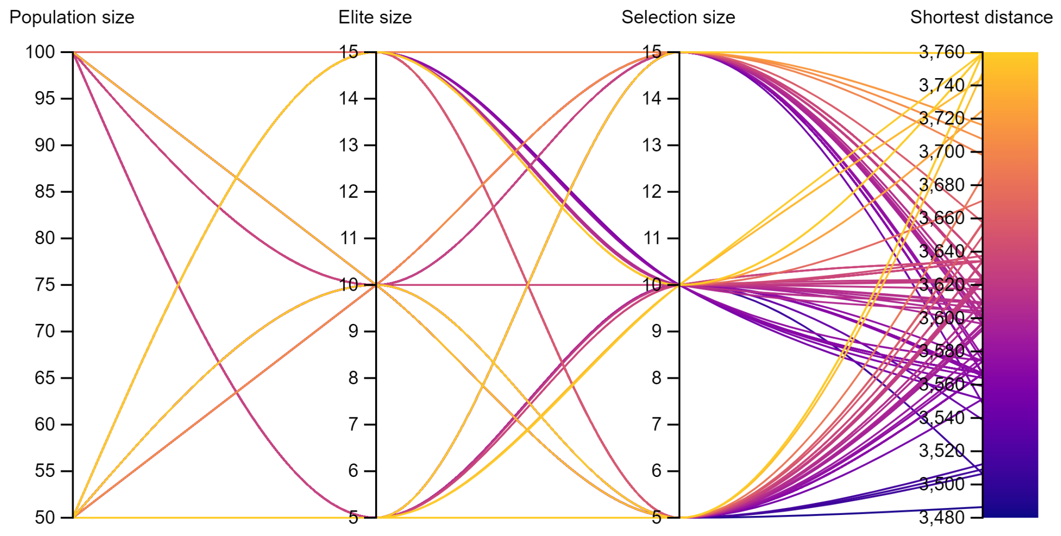
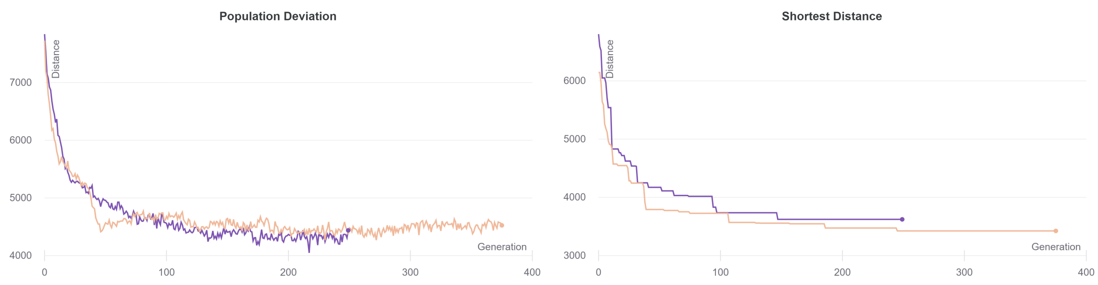


Figure 6.6: Illustration of the entire parameter tuning test results, which shows the results landing between an interval of 3,400 - 3,800. [41]

Order Crossover (OX1)

The result of the different tests OX1 went through are show in Figure 6.8. As can be seen in Figure 6.7 the two examples shown are the more common example depicted with purple and light brown for the outlier. The Figure 6.7a shows how the population like TCX changes through the generations, meaning that the OX1 is capable of doing small adjustments itself and might after a sufficient amount of generations find the global minimum. Here mutation process could still help by making a even more diverse population.



(a) *(b)*

Figure 6.7: The population diversity in (a) does have a steady diversity in its population that shows the crossover is able to mix its population without mutation. In (b) the decrease of the common (purple) example has a more gentle slope compared to the outlier (light brown). [41]

The Figure 6.7b shows that the outlier (light brown) quite quickly finds a good combination of tasks for a local minimum whereas the most common (purple) gradually decreases and finds a local minimum rapidly and gets ended by the breakpoint. The effect of having a greater population size can be seen in Table 6.4 where most of the mean distances for a population size of 100 are under 3600 in comparison to the ones with a population size of 50, which all are above 3600. The best parameter connections and results are marked with green, and are carried over to the mutation testing.

Elite size	Selection size	Population size	Mean distance	Time (s)	Population size	Mean distance	Time (s)
5	5	50	3672.757	240.9	100	3596.934	971.8
5	10	50	3722.639	237.9	100	3715.333	688.2
5	15	50	3836.004	192.1	100	3596.935	1055.8
10	5	50	3806.552	226.5	100	3651.52	780.3
10	10	50	3730.854	199	100	3596.572	819.2
10	15	50	3774.442	216.1	100	3592.275	706.5
15	5	50	3871.489	188.2	100	3643.754	775.2
15	10	50	3751.735	225.5	100	3635.855	884.3
15	15	50	3722.749	208.5	100	3528.868	874.3

Table 6.4: The different combinations of parameters for the OX1 are tested, and the best combination for a population size of 50 and 100 are highlighted with green.

Something worth noting in the computation time is that a population size of 100 takes around three times as much time compared to a population size of 50. The result interval of the entire tuning test can be seen in Figure 6.8, where the OX1 is more focused between 3500 and 3750, which is still quite far from the global minimum, which the mutation could help improve.

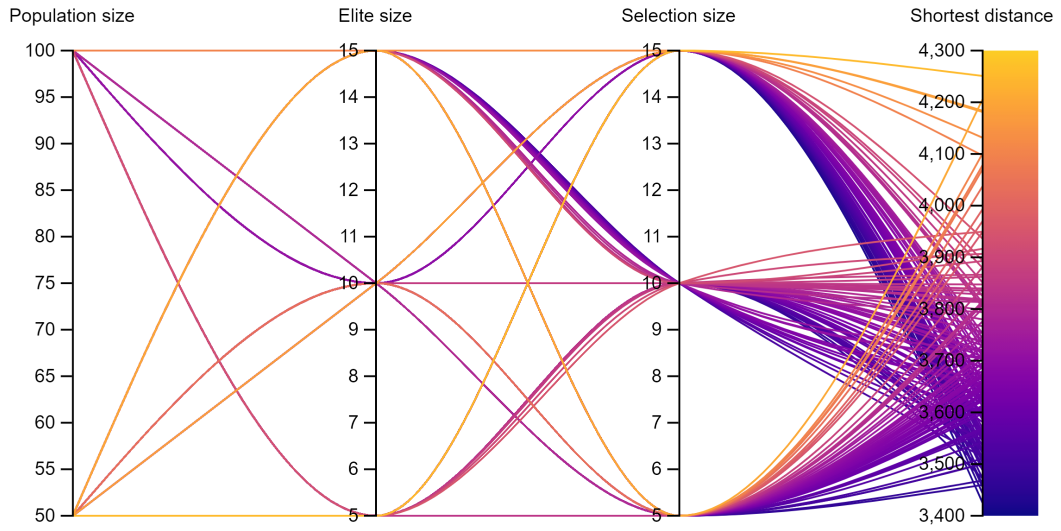


Figure 6.8: The results of the tests for all the different combinations of tuning parameters are visualised here. The results are concentrated in the 3,400 - 4,300 interval. [41]

Partially Mapped Crossover (PMX)

The PMX, in case of coding and understanding is quite straightforward to implement as can be seen in section 4.4. The PMX highly resembles the DPX in terms of population diversity, which can be seen in Figure 6.9a as it also finds the local minimum around generation 20. The rapid convergence might be due to most of the population being identical and that no diversity is created without mutation.

The two examples in Figure 6.9 shows the common result as brown and the outlier with grey, which both reach the breakpoint rapidly, due to no diversity in the population. The outlier is the best result out of all 180 tests made and is not very common, which is why the the mean value gives a better picture of the overall test, these results can be seen in Table 6.5.

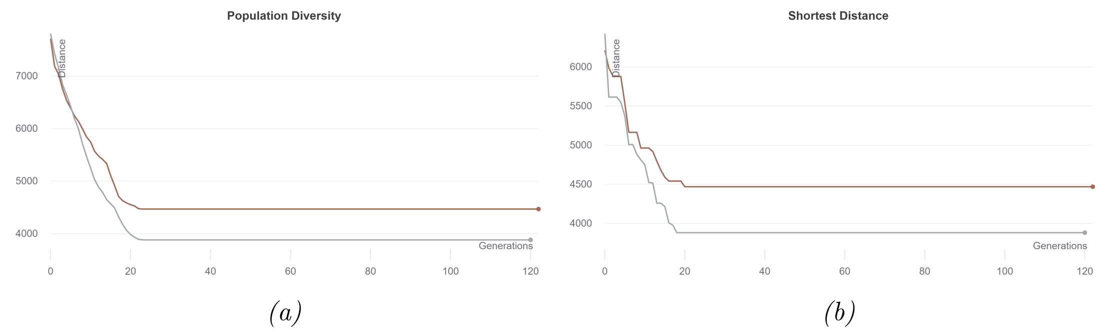


Figure 6.9: (a) shows the population diversity does not change much after 20 generations where (b) shows the Shortest distance are found equally rapidly and stays with it to the breakpoint is met. [41]

Elite size	Selection size	Population size	Mean distance	Time (s)	Population size	Mean distance	Time (s)
5	5	50	4850.346	76.9	100	4398.954	308.5
5	10	50	4793.079	77.4	100	4246.167	315.4
5	15	50	4684.526	75.8	100	4492.333	305.7
10	5	50	4810.021	77.8	100	4505.411	312.4
10	10	50	4946.287	75.8	100	4384.484	308.9
10	15	50	4931.558	76	100	4502.359	309.3
15	5	50	4703.754	77.1	100	4324.158	302.7
15	10	50	4849.593	76.9	100	4405.402	309.8
15	15	50	4691.766	75.9	100	4498.618	305

Table 6.5: The green highlighted combinations are the best hyperparameters derived through testing all the possible combinations of tuning parameters for PMX.

As can be seen in the table the once with a population size of 50 is quite high and lays around 4800, whereas with 100 they lay around 4400. The difference here might be the greater diverse population made in every generation. Due to the fact that the local minimum are found rapidly the computation time cannot give the best image of how fast it would go with mutation, but it shows like the others a difference of when the population are increased. The best result and parameters are marked with green.

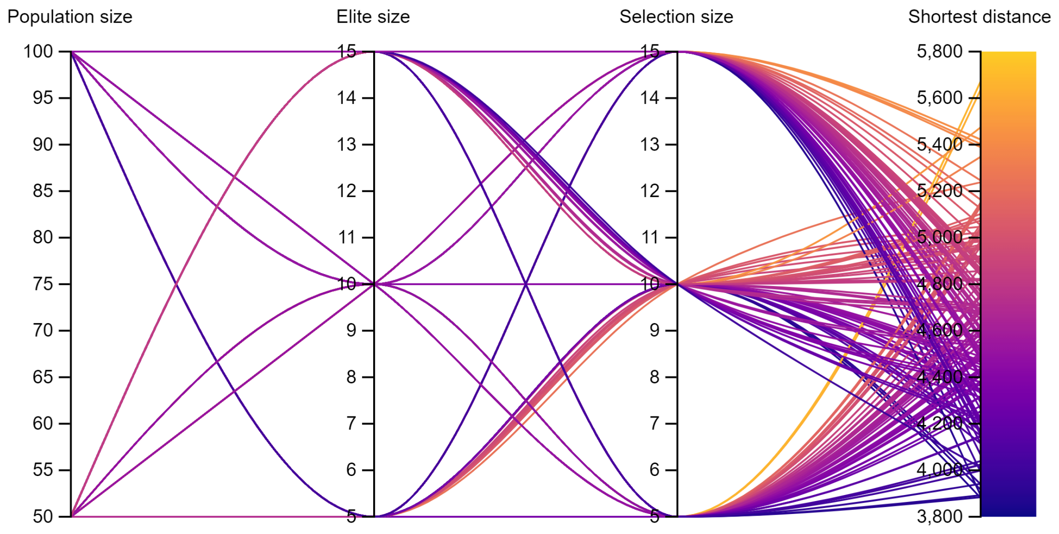


Figure 6.10: Illustration of all the results from the PMX crossover tuning, which is not very focused and far from global minimum. All the results are spread out between an interval of 3,800 - 5,800. [41]

The distortion of all the results from the tuning test can be seen in Figure 6.10, where the focus is quite wide as it goes from 4200 to 5000. the high results might be the fast settling or that the crossovers algorithm itself does not works very well without mutation.

In Table 6.6 the best hyperparameters can be seen for each crossover to be test tuned with the mutation as the ones most likely to give a good result. The DPX with a population of 50 has the shortest distance, however it does rely on the mutation to create enough diversity together with PMX. This could mean that the mutation needs to happen in almost every generation in order to make population diverse enough to produce a good result. A mixing of two crossovers could also work to benefit by for example shifting the crossover with the generation if the mutation is not diverse enough. The OX1 also shows to be quite good as it is the second shortest with a population of 50 and actually beats DPX with a distance of 100. In terms of distance the PMX has the worst score, but could show to be much better with the mutation.

Crossover	Elite size	Selection size	Population size	Mean Distance	Time (s)
TCX	15	10	50	3749.84	164.5
DPX	15	5	50	3589.42	70.7
OX1	5	5	50	3672.76	240.9
PMX	5	15	50	4684.53	75.8
TCX	10	15	100	3651.10	964.1
DPX	5	10	100	3564.42	280.2
OX1	15	15	100	3528.87	874.3
PMX	5	10	100	4246.17	315.4

Table 6.6: Summery of all the best results and hyperparameters to be tested with mutation, with both population sizes.

6.5 Mutation

The mutations as described in chapter 4 provides small random changes to the individual, that can happen at some point in the process of finding the shortest path. The different mutation operations will be chosen randomly, so that each mutation method have an equal chance to be used. The introduction of the mutation to the individual will be seen as a hyperparameter of how likely a certain individual will be to mutated. This hyperparameter will be referred to as the *mutation rate*. The new parameter is going to be tested in this section, spanning from 10 % to 100 % as it has already been tested in the previous section with a mutation rate of 0 %. Each test are repeated 10 times following the trend set in previous tests. The mutation methods used are

- Sequence inversion
- Transposition
- Insertion
- Sequence transposition

The chromosome contraction and chromosome partition will however not be used in these test as the amount of AGVs are fixed to three. The mutation rate will be tested with the desired parameters for each crossover found in the previous section. The breakpoint is set to a 100 generations, the generation maximum to 500 generations and the task number will remain at 25 tasks. However as the population rate was not that conclusive in the previous section and can also have an effect on the chosen method in section 6.6 the mutation will therefore continue to be tested on a population rate of 50 and 100. The mean value of the 10 repeated test are used to determine the best method and mutation rate together with the time spend. However, the time spend are not weighted as high as the distance, due to the fact that the program is going to be distributed between multiple agents and the computation time will therefore decrease depending on the solution chosen in section 6.6.

In Table 6.7 it can be seen that both DPX and PMX works well with a higher mutation rate than OX1 and TCX, along with getting relatively close to each other in terms of the distance. However the DPX shows the best result (marked in green) with a population of 50 and a mutation rate on 90%

Population 50	DPX		OX1		PMX		TCX	
Mutation rate	Mean distance	Time (s)	Mean distance	Time (s)	Mean distance	Time (s)	Mean distance	Time (s)
0.1	3438.12	153.40	3638.52	288.60	3567.10	256.00	3747.05	163.40
0.2	3437.79	153.60	3562.36	279.30	3474.92	268.60	3754.57	193.60
0.3	3417.77	199.60	3592.42	260.10	3314.67	218.10	3769.90	180.80
0.4	3422.52	152.90	3646.19	265.30	3441.12	257.50	3851.31	145.70
0.5	3449.18	119.50	3636.52	278.60	3423.94	246.40	3864.69	167.40
0.6	3466.68	129.10	3788.82	228.90	3197.92	281.00	3930.18	156.90
0.7	3276.99	153.80	3789.52	235.50	3178.33	268.70	3993.36	132.30
0.8	3194.04	192.00	3986.12	204.90	3299.36	278.70	4132.18	118.40
0.9	3120.75	179.60	3821.71	214.10	3445.83	262.00	4103.57	123.70
1	3123.67	207.60	4075.28	200.80	3285.42	282.40	4088.06	144.30

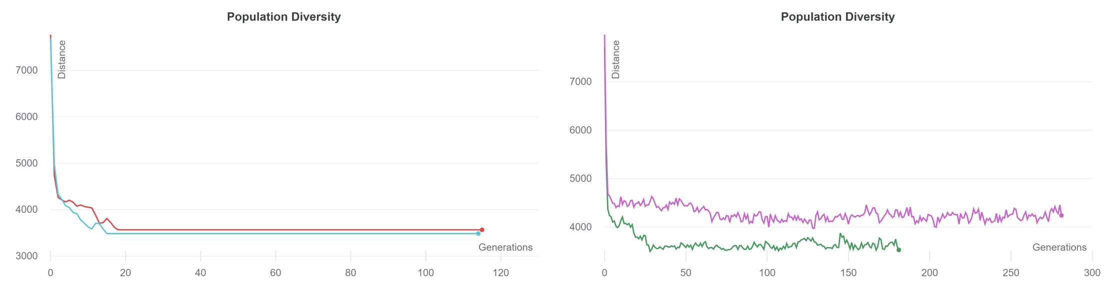
Table 6.7: Mutation test results of the GA algorithm with a population of 50. The best result for each crossover is marked in green where the shortest distance weights higher than the time spend

In Table 6.8 it can be seen that DPX still works well with a higher mutation rate than the others OX1 and TCX, but also interesting to see that the PMX works best with a lower mutation rate than in Table 6.7 and still gets quite close to the previous result in relation to distance, but takes more time to compute. However the DPX still shows the best result (marked in green) with a population of 100 and a mutation rate a bit higher on 100%.

Population 100	DPX		OX1		PMX		TCX	
Mutation rate	Mean distance	Time (s)	Mean distance	Time (s)	Mean distance	Time (s)	Mean distance	Time (s)
0,1	3435,31	572,30	3480,30	859,20	3452,59	979,50	3841,70	670,40
0,2	3423,08	544,40	3446,44	765,30	3414,36	849,80	3814,05	707,70
0,3	3381,44	609,20	3441,48	755,00	3199,59	1014,90	3862,36	610,20
0,4	3408,72	536,70	3598,27	698,60	3439,13	850,70	3877,75	632,50
0,5	3385,55	606,20	3550,19	636,30	3265,51	834,90	4021,05	692,30
0,6	3252,56	607,70	3570,21	612,30	3344,20	777,40	3831,25	789,50
0,7	3096,97	723,00	3445,24	721,80	3314,20	874,80	3958,71	601,30
0,8	3082,58	681,60	3640,06	663,20	3361,19	912,60	4039,14	599,00
0,9	3120,55	639,00	3766,62	607,70	3411,27	997,60	4181,00	540,00
1	3087,15	546,90	3756,83	566,10	3370,72	941,10	4046,33	699,70

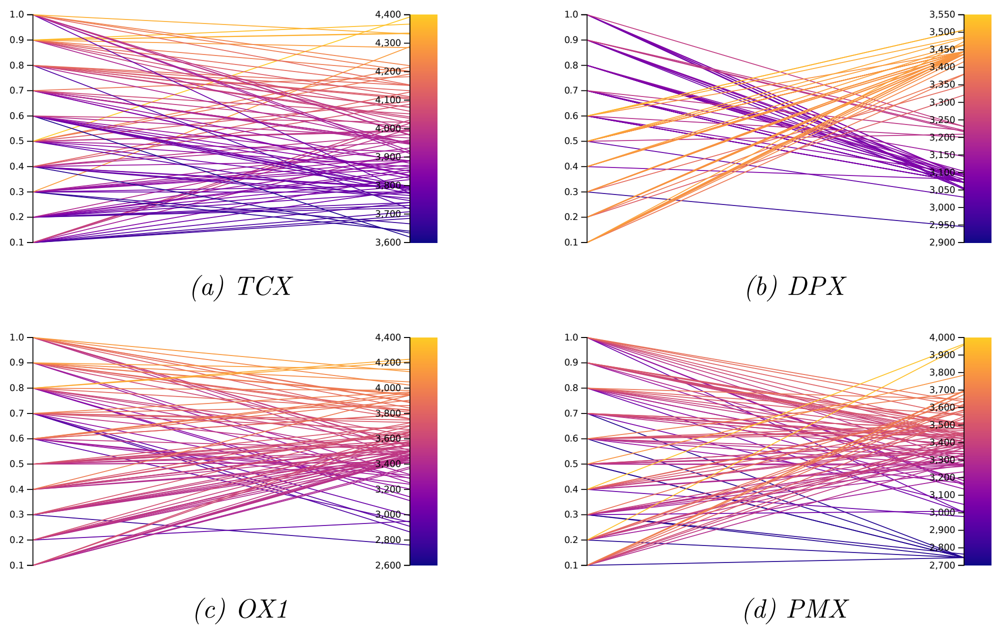
Table 6.8: Mutation test results of the GA algorithm with a population of 100. The best result for each crossover is marked in green where the shortest distance weights higher than the time spend.

This also makes some sense when looking at the Figure 6.11a which shows that the DPX needs an external mixer as it uses a distance heuristic to find the shortest route as illustrated in Figure 4.10 and described in section 4.4. This will happen to all individuals in the population and after one generation the local minimum is found and very few to no changes will happen before mutation is introduced as seen in Figure 6.11b. Whereas OX1 and TCX crossover also does some mixing that creates new results every time the PMX more or less follows DPX when no mutation rate is introduced.



(a) *(b)*

Figure 6.11: Illustration of how important mutation is for the DPX in order to make a more diverse population which creates even more possibilities to find the shortest distance. [41]



(a) TCX *(b) DPX* *(c) OX1* *(d) PMX*

Figure 6.12: All four graphs are derived with a population size of 100. (a) span the largest interval, while (b) has two clear separate cluster that show the effect of different mutation rates. [41]

When looking at the data shown in Figure 6.12 it can be seen clearly that both the PMX and especially the DPX are more concentrated with some outliers. When looking at Figure 6.12b it can be seen that there is a turning point around 60% where it gets closely fixed to 3100, whereas in Figure 6.12d all the tests are fixed between 3300 - 3500, with some outliers where some of them goes directly towards the global minimum at 2744. This however cannot be trusted as much as the DPX as it is show to have more consistency and accuracy than the others. Due to the fact that the DPX gives the best result out of the four crossovers, it will be the chosen crossover method for this project.

6.6 Decentralisation

As briefly mentioned in section 3.1 there exits the possibility of decentralising the algorithm by assigning the task of finding an solution to the given problem to the AGVs operating in the environment instead of assigning it to a central system. The main benefit of decentralisation is to avoid the problems that inherently comes with having a single point that connects every part of the system, the primary problem being an accidental shutdown of the central master leading to a fleet of unguided AGVs.

In the case of this project, when using this approach it is up to the AGVs to cooperate in order to find a solution to the mTSP by first setting up a communication link and thereafter using a protocol to share and explore the different solutions generated by the GA covered in chapter 4. For the project, two methods of decentralising were discussed. The primary goal of the discussion was to decide which approach best complimented the underlying centralised GA algorithm covered throughout this chapter.

6.6.1 Method 1: Split and Compare

This method works by utilising the internal processing power of each machine connected together in a network cluster to run multiple GA algorithms in parallel. Here a single AGV acts as a master that distributes work out among all other connected AGVs referred to as workers while still handling some of the workload itself. Each algorithm run in parallel returns a population of n solutions to the mTSP that are then compared. Following the comparison the best population among them is picked and improved upon by feeding it through the same process again for a set number of generations. This approach can be split into three steps those being setup, split and compare.

Setup

Here a select number of GA algorithms are chosen to be run in parallel along with a specification of hyperparameters such as elite and selection size. Before the algorithm moves on, it is assumed that the network cluster is already initiated by a master through some secondary process.

Split

An initial population is generated to be fed into each of the multiple algorithms chosen to be run in parallel. Following the GA setup the master distributes/splits out the task of processing among all AGVs connected to the network cluster while still contributing to the task of processing.

Compare

Upon finishing and returning the populations generated by the GA's running in parallel to the master, the process of comparing each population takes place. Here each population is assigned a fitness score which is stored in a list and thereafter compared to all other scores in the same list. The best score among them is picked and the population from which it was generated is identified. The population with the best score is then fed back into the "split" process, taking the same role as the initial generation.

Repeats	Parallel GAs			Single GA		
	Distance	Runtime (s)	Generation	Distance	Runtime (s)	Generation
1	3072	262	291	3121	1762	287
2	3.088	529	384	3.231	912	167
3	3.080	401	299	3.069	1083	188
4	3.071	349	388	3.062	1649	281
5	3022	463	349	3065	2314	395
6	3.023	319	355	3.000	1574	253
7	3.231	323	237	3.080	1527	273
8	3.069	273	303	3.071	1273	212
9	3.088	376	280	3.083	883	157
10	3072	172	190	3088	948	169
11	3093	184	202	3071	1358	240
12	3220	274	300	2968	1497	238
13	3.088	417	297	3071	1638	269
14	2839	360	402	3080	1556	248
15	3.055	446	335	3088	1187	200
16	3080	203	226	3062	1258	211
17	3.071	348	254	3080	1648	208
18	2998	369	413	3071	1995	338
19	3507	111	120	3071	1367	226
20	3072	217	239	3071	1645	276
Mean	3092	319,8	293,2	3075,15	1453,7	241,8

Table 6.9: An overview of the test results from method 1. On the left side of the table are the results generated by the three GAs running in parallel while the results from the single GA running on a separate machine is shown to the right. Each parallel GA runs with a population size of 50 and a mutation rate of 90% while the single GA uses a population size of 150 with the same mutation rate as the other algorithms. Where the distances is pretty close compared to the run time

The primary goal of this method is to improve the precision and consistency of the underlying GA by comparing multiple individually processed populations on a generation by generation basics. By splitting up this task through parallel processing and cloud computing an otherwise slow algorithm can be sped up greatly compared to running a single GA algorithm with a considerable larger population size. The method was tested by comparing the results of running three GAs in parallel on one machine while running a single GA on another. The three GAs running in parallel used a population size of 50, mutation rate of 90%, breakpoint of a 100 generations and max limit of 500 generations. The single GA running on the separate machine however ran a population size of 150 to approximate running a GA without the inclusion of the three algorithms running in parallel. Mutation rate, breakpoint and generation limit mirrors that of the first machine. To better approximate the effectiveness of the algorithm, the system was run for 20 iterations. The result of the test are presented in Table 6.9

6.6.2 Method 2: Run and Compare

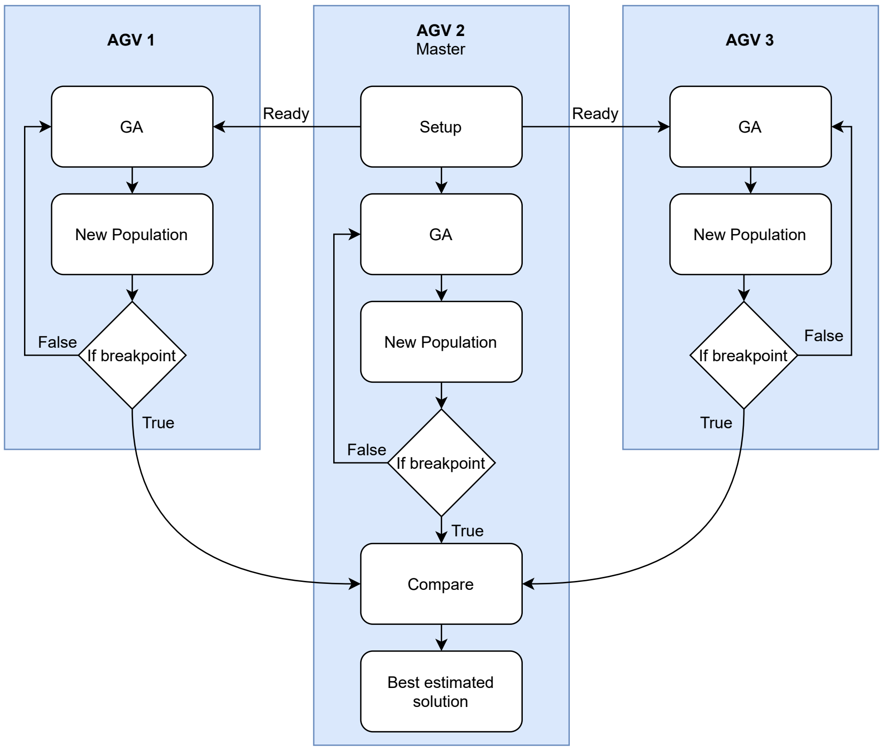


Figure 6.13: In the process illustrated above the master takes care of the initial setup which includes setting up the parameters that defines how the GA in should run. Upon completing the setup the parameters along with a ready signal is sent to all agents connected to the master which initiates a GA running on both the master and all connected AGVs. Upon the completion of each GA running separately the best individual from each generation along with a fitness score is sent back to the master for comparison. Through this comparison the best estimated solution is selected for execution

The method works by running separate GAs on different machines from start to finish, only comparing the last generation in order to select the best solution among them which is then distributed to all AGVs in the system for execution. This method consists of three steps those being setup, run and compare. This process can be seen illustrated in Figure 6.13

Setup
First an AGV is selected to act as the master. The job of the master is to initialise the parameters specifying how the GAs should operate which it sends along with a ready signal to all AGVs connected to the master.

Run
Upon receiving the ready signal from the master, the master along with all connected AGVs begins their own GA with the specified parameters.

Compare
Upon completing a select number of generations each AGV sends back the best individual found in its population along with its associated fitness score to the master. The master then compares each individuals fitness score and then selects the best solution among them. The best solution is then send back to all the AGVs.

	Distance	Generation	Runtime	Distance	Generation	Runtime	Distance	Generation	Runtime	Distance	Generation	Runtime
Repeats		AGV 1			AGV 2			AGV 3			The Best	
1	3093	378	255	3137	220	138	3121	344	143	3093	378	255
2	3217	295	191	3114	353	231	3088	216	91	3088	216	231
3	3137	238	161	3106	164	104	3089	361	152	3089	361	161
4	3184	437	297	3104	302	194	3543	132	55	3104	302	297
5	3231	205	138	3088	311	206	3466	136	55	3088	311	206
6	3231	238	162	3076	280	186	3088	195	81	3076	280	186
7	2896	309	212	3088	361	240	3120	278	111	2896	309	240
8	3080	380	255	3220	212	139	3104	254	112	3080	380	255
9	3200,42	345	233	3095	274	187	3013	293	122	3013	293	233
10	3114	223	150	3115	247	161	3088	411	184	3088	411	184
11	3264	206	138	3121	348	238	3095	182	79	3095	182	238
12	3088	286	192	3088	208	150	3121	274	111	3088	208	192
13	3048	332	226	3.104	314	215	3231	232	93	3048	332	226
14	3090	263	183	2982	400	273	3114	254	103	2982	400	273
15	3071	228	156	3120	292	199	3120	288	111	3071	228	199
16	3088	335	224	3088	328	217	3095	390	153	3088	328	224
17	3111	186	123	3080	410	271	3115	282	116	3080	410	271
18	3077	274	181	3087	214	143	3003	382	161	3003	382	181
19	3088	305	203	3080	296	193	3058	304	132	3058	304	203
20	3114	281	185	3.088	290	196	3121	241	94	3.088	290	196
Mean	3121	287	193	3099	291	194	3140	272	113	3061	315	223

Table 6.10: An overview of the test results from method 2 using a populations size of 50 and a mutation rate of 90% for three individual GAs running on three separate machines/AGVs

Like method 1 the goal of this approach is to improve precision and consistency in the result by selecting a solution among a bigger sample space. However since each algorithm is run separately there exits no correlation between them like in the previous method which could decrease the chance of ending in a collective local minimum. In order to test the method, three machines each with their own processing specs were used to imitate the process that should run on the AGVs presented in the use case section 2.1. Each machine was tasked with completing a full run of the GA using a population size of 50, mutation rate of 90%, a breakpoint of 100 generations and max generation limit of 500 generations. Upon completion the result is returned for comparison in order to find the best among them. This process was repeated 20 times in order to converge to a better final estimate of how the method compares to the baseline set by the test presented in Table 6.8 which can be found in section 6.5. The result of the method 1 test is presented in Table 6.10 below:

Algorithm 5: A GA consists of five functions and outputs an individual with the lowest overall distance. The first function generates an initial population and is only performed once. After that a fitness score is calculated for each individual, and it is used as the basis of the selection process. The selected individuals are combined in a crossover function, and later mutated. The last four are iterated through a given number of generations.

```
Input  : Task list, population size, maximum number of generations, and
         number of AGVs
Output: Optimised task order for each AGV
Establish connection to AGVs in range
Select a master among available AGVs
Create initial population based on input parameters.
for number of generations do
    Calculate fitness of individuals in the population
    Select individuals to form a mating pool the same size as the population
    Combine two randomly sampled individuals from the mating pool and
      combine them with a crossover function
    Randomly mutation some individuals
end
if master then
    Compare best individuals from all AGVs
else
    Send best individual to master
end
return Individual with lowest overall distance
```

7 Discussion

The selection method was not tested in the same way as the crossover and mutation as the tournament selection showed to be best among those tested initially, it was therefore kept throughout future iterations of the algorithm. It could however be beneficial to apply the same testing to the selection methods in order to insure that it truly was the best one chosen for the project. From the testing of the different crossover methods it could be seen that the DPX and PMX behaved in the same manner without mutation by rapidly converging to a local minimum but with a difference of 1000 in their mean distance value. Whereas the TCX and OX1 both were capable of creating a more diverse population while testing different combinations they were more or less equal in terms of mean distance values.

However, in the end DPX came out with the best result in the case of distance and runtime with a population of 50 due to its rapid convergence. Whereas the OX1 beats the DPX in distance it is comparatively slower. In terms of distance and time the DPX is the overall winner even though no mutation was applied which also contributes to the short processing time.

When the crossover was provided with mutation the focus of the DPX and PMX became clear and showed to be more reliable around a specific point, whereas the TCX did not make as much progress as the others, which can be seen in Figure 6.12. The PMX managed to hit global minimum most often, but is not reliable enough to be used as the main crossover as the best results can be interpreted as outliers since they are not present with specific hyperparameters. They seem to appear randomly for mutation rates between 10% and 70%, and only seven global minimum solutions were produced out of 100 tests total.

However, a combination of DPX and PMX might be able to push the DPX further down towards the global minimum, which is a process further explored in 9. As it can be seen in Table 6.7 and Table 6.8 the DPX needs a high mutation rate to produce a good result, which also could mean that changing few genes around is not enough to create a diverse population. More aggressive mutations might benefit DPX. A combination with another crossover like TCX could also make enough diversity to get the DPX to the global minimum.

An idea for adding diversity and accuracy to the algorithm was tested. In the first generation of the algorithm an initially large population size was generated either of size 250, 500, 750 and 1000. The theory was that the increase in search space would improve the chances of selecting better individuals, and in general include more diverse solutions. However, the results shown in Table 6.1 does not show a significant difference when compared to the baseline results from Table 6.7. The initially large population results only show a slight positive difference of 19.25 at the cost of 6.4 seconds. To get better results, multiple generations of larger populations could be a way to do it. However, this will always come at the cost of computation time.

Through the testing of the two decentralisation methods covered in section 6.6 it can be seen that both in terms of distance lies in close proximity to the baseline set by the test illustrated by Table 6.8 along with the single GA using a large population seen in Table 6.9. However, where the two methods truly shine is in regards to processing time. Here the process of running multiple GAs in parallel like in Method 1 or initiating multiple instances of the GA on different machines as done in Method 2 can greatly increase the accuracy and consistency of the algorithm, while still allowing for a relatively small population size that decreases processing time. When it comes to directly comparing the two decentralisation methods however the task of picking a particular approach is not nearly as straightforward since both methods are closely matched in both distance and processing time. The approach covered in Method 2 outperforms Method 1 by a small margin yet is important to note that only three GAs were run in parallel in the test performed for Method 1. This was primarily due to the fact that a comparison with a single GA was needed to put the process of parallelisation into perspective, if more GAs were chosen to run in parallel the population size on the corresponding algorithm would have to increase as well, resulting in drastically increased processing time. As seen on the right side of Table 6.9 the process of running the algorithm a single time from start to finish with an population of 150 already took on average 24 minutes. Furthermore, due to what is suspected to be a software incompatibility issue, the network cluster was unable to connect the PCs used to emulate the AGVs in the system, thereby resulting in Method 1 not being able to be tested to its full capability.

A battery constraint concept was given in subsection 6.2.1, and it was tried implemented in the algorithm. However, the results were poor. The developed task list did include the charging task, and it was included in the fitness score of the individual, but its representation in the map and the overall performance of the algorithm suffered for it. It is most likely due to implementation errors, but they were not studied or fixed because the focus was elsewhere.

8 Conclusion

This project has investigated the challenges associated with developing a GA for solving a task assignment problem for multiple decentralised AGVs. The case was provided by a company interested in the use of AI to control fleets of their AGVs. The algorithm developed in this project goes one step towards that goal.

The project work was separated into four research and development objectives which are defined in section 5.1. The first objective was to develop a GA framework for a centralised mTSP setup. This objective was completed in steps starting with the development of a simple GA used to solve a TSP. Following the successful implementation, the GA research was focused towards upgrading the TSP to a mTSP using the underlying source code as a foundation. In order to make a functional implementation of a mTSP the initial population, fitness, selection and mutation function were modified to work for multiple agents, the crossover was however completely replaced. This was done in order to make it compatible with the multi-chromosome representation described in section 4.1. Upon the implementation of the new crossover the GA for solving the mTSP was completely functional.

The second objective was to incorporate decentralisation capabilities into the mTSP framework. Here multiple approaches to setting up a communication link between AGVs were explored, ultimately resulting in the two methods described in section 6.6. Doing the implementation of the framework that was needed to establish a connection between the AGVs, a software problem was encountered that halted the implementation. It was therefore decided to instead focus on the parallelisation aspect that was meant to be part of the "split and compare" method along with emulating a connected system for the "run and compare" method.

The third objective was to design and introduce resource constraints, in the form of battery limitations. A concept solution was developed, but the tests did not show satisfactory results which could mark requirement 2 as fulfilled.

To fully finish the GA most of the algorithm except for fitness and selection due to that fitness is hard to improve upon as it only calculates the distances. The use of tournament selection was decided upon, during initial testing throughout completing RDO1, however this approach to selection needs some more testing like the other steps in the GA has undergone to ensure that it truly is the best method suited for this project.

In regards to decentralisation the chosen method is the "split and compare" method covered in section 6.6 primarily due to its potential when paired with other machines connected in a network cluster or a single machine capable enough to run multiple GAs in parallel without a significant slowdown. Furthermore, since the method functions on a generation to generation basis, different selection, crossover and mutation operators can be initiated and changed during the iterative process allowing for multiple ways to further push the generated result towards the global minimum. Due to the software issue encountered in the project, requirement 1 can not be fully defined as fulfilled, but through testing it was proved that the underlying concept performs just as good if not better than a classic GA.

The DPX and PMX crossover tested with both a population of 50 and 100 where the only two to have a mean distance within requirement 3 specifying that the results should be within 20% of the global minimum. However, DPX was the only one to also be within the 90% repeatability. Based on this performance the DPX crossover operator was the chosen method for this project's GA solution. With this in mind requirement 3 is concluded as fulfilled.

In conclusion, RDO1 and RDO4 were completed, while RDO2 was conceptually completed, RDO3 was however less so. Furthermore, the first and last requirement was fulfilled, while the second was not.

9 Future Works

- While the DPX crossover operator covered in section 6.4 yielded good results when using a high mutation rate it still has the tendency to get stuck in local minimums appearing close to the global one. This illustrates that the current mutations might not have the power alone to improve the algorithm which leads to the idea of combining the current crossover with another, the primary candidate for this second crossover is PMX. Since PMX were the only crossover method to ever arrive at the global minimum the idea would be to initially use DPX to rapidly converge to a local minimum close the global one and then switch to PMX to nudge the algorithm a few steps in the right direction or at least get unstuck.

- While setting up the networking infrastructure needed to connect the different AGVs in the system a software problem was encountered. This problem halted the implementation of decentralisation which led us to only being able to test the parallel processing that was meant to work in tandem with the possibilities connected agents could have provided. For future work one of the first things to be addressed would be the networking issue.

- As the resource constraints were not properly integrated into the system a future goal would be to reevaluate the implementation and improve upon it in order to create functional algorithm that optimises with respect to battery level, number of AGVs and travel time.

- For validating the final solution future work should go towards implementing the solution in a fully realised 3D environment created through the use of Webots. This environment should mirror the one detailed in the 2.1.

- Subjecting each selection method to a round of testing could help improve the algorithm by maybe pointing out a better alternative to the tournament selection process used in this project.

Bibliography

[1] N. Integra, "Automation in industry 4.0," , 18/06/2020, accessed: 27/05/2021. [Online]. Available: https://nexusintegra.io/automation-in-industry-4-0/

[2] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048304001550

[3] M. Mitchell, *An Introduction to Genetic Algorithms.* MIT Press, 1998. [Online]. Available: https://mitpress.mit.edu/books/introduction-genetic-algorithms

[4] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006. [Online]. Available: http://www.jstor.org/stable/j.ctt7s8xg

[5] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[6] M. U. Arif and S. Haider, "An evolutionary traveling salesman approach for multi-robot task allocation," 01 2017, pp. 567–574.

[7] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149–156, 1991. [Online]. Available: https://doi.org/10.1287/ijoc.3.2.149

[8] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048304001550

[9] T. Vidal, G. Laporte, and P. Matl, "A concise guide to existing and emerging vehicle routing problem variants," *European Journal of Operational Research*, vol. 286, no. 2, pp. 401–416, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221719308422

[10] A. H. Alaa Khamis and A. M. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," 2015. [Online]. Available: https://www.researchgate.net/publication/277075091_Multi-robot_Task_Allocation_A_Review_of_the_State-of-the-Art

[11] S. A. Z. Dong-Hyun Lee and J.-H. Kim, "A resource-oriented, decentralized auction algorithm for multirobot task allocation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 4, pp. 1469–1481, 2015.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis,

"Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *NIPS*, 12 2013.

[14] Y. Hu, Y. Yao, and W. S. Lee, "A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs," *Knowledge-Based Systems*, vol. 204, p. 106244, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705120304445

[15] Y. Kaempfer and L. Wolf, "Learning the multiple traveling salesmen problem with permutation invariant pooling networks," p. 22, 03 2018.

[16] N. Carion, G. Synnaeve, A. Lazaric, and N. Usunier, "A structured prediction approach for generalization in cooperative multi-agent reinforcement learning," *CoRR*, vol. abs/1910.08809, 2019.

[17] M. S. A. Hameed and A. Schwung, "Reinforcement learning on job shop scheduling problems using graph networks," 2020.

[18] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *ArXiv*, vol. abs/2010.12367, 2020.

[19] M. Chang, S. Kaushik, S. M. Weinberg, T. L. Griffiths, and S. Levine, "Decentralized reinforcement learning: Global decision-making via local economic transactions," 2020.

[20] F. Köpf, S. Tesfazgi, M. Flad, and S. Hohmann, "Deep decentralized reinforcement learning for cooperative control," 2019.

[21] P. a. Gao, Z. x. Cai, and L. l. Yu, "Evolutionary computation approach to decentralized multi-robot task allocation," in *2009 Fifth International Conference on Natural Computation*, vol. 5, 2009, pp. 415–419.

[22] R. Patel, E. Rudnick-Cohen, S. Azarm, M. Otte, H. Xu, and J. W. Herrmann, "Decentralized task allocation in multi-agent systems using a decentralized genetic algorithm," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3770–3776.

[23] D. Singh, E. Singh, T. Singh, and R. Prasad, "Genetic algorithm for solving multiple traveling salesmen problem using a new crossover and population generation," *Computación y Sistemas*, vol. 22, 07 2018.

[24] A. Rexhepi, A. Maxhuni, and A. Dika, "Analysis of the impact of parameters values on the genetic algorithm for tsp," *International Journal of Computer Science Issues*, vol. Volume 10, pp. pp 158–164, 01 2013.

[25] D. Singh, M. Singh, T. Singh, and R. Prasad, "Genetic algorithm for solving multiple traveling salesmen problem using a new crossover and population generation," *Computación y Sistemas*, vol. 22, 07 2018.

[26] A. Singh and A. Baghel, "A new grouping genetic algorithm approach to the multiple traveling salesperson problem," *Soft Comput.*, vol. 13, pp. 95–101, 05 2009.

[27] E. K. Burke, J. P. Newall, and R. F. Weare, "Initialization Strategies and Diversity in Evolutionary Timetabling," *Evolutionary Computation*, vol. 6, no. 1, pp. 81–103, 03 1998. [Online]. Available: https://doi.org/10.1162/evco.1998.6.1.81

[28] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[29] E. Kocyigit, O. K. Sahingoz, and B. Diri, "An evolutionary approach to multiple traveling salesman problem for efficient distribution of pharmaceutical products," in *2020 International Conference on Electrical Engineering (ICEE)*, 2020, pp. 1–7.

[30] K. Jebari, "Selection methods for genetic algorithms," *International Journal of Emerging Sciences*, vol. 3, pp. 333–344, 12 2013.

[31] P. Larranaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, pp. 129–170, 01 1999.

[32] M. U. Arif and S. Haider, "An evolutionary traveling salesman approach for multi-robot task allocation," 01 2017, pp. 567–574.

[33] K. El Bouyahyiouy and A. Bellabdaoui, "A new crossover to solve the full truck-load vehicle routing problem using genetic algorithm," in *2016 3rd International Conference on Logistics Operations Management (GOL)*, 2016, pp. 1–6.

[34] M. A. Al-Omeer and Z. H. Ahmed, "Comparative study of crossover operators for the mtsp," in *2019 International Conference on Computer and Information Sciences (ICCIS)*, 2019, pp. 1–6.

[35] H. Ismkhan and K. Zamanifar, "Study of some recent crossovers effects on speed and accuracy of genetic algorithm, using symmetric travelling salesman problem," *International Journal of Computer Applications*, vol. 80, pp. 1–6, 10 2013.

[36] V. Chahar, S. Katoch, and S. Chauhan, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools and Applications*, vol. 80, 02 2021.

[37] J. Dou, C. Chen, and P. Yang, "Genetic scheduling and reinforcement learning in multirobot systems for intelligent warehouses," *Mathematical Problems in Engineering*, vol. 2015, p. 10 pages, 12 2015.

[38] Z. Ahmed, "Adaptive sequential constructive crossover operator in a genetic algorithm for solving the traveling salesman problem," 02 2020.

[39] A. Király and J. Abonyi, *Optimization of Multiple Traveling Salesmen Problem by a Novel Representation Based Genetic Algorithm*, 07 2011, vol. 366, pp. 241–269.

[40] E. Özcan and M. Erenturk, “A brief review of memetic algorithms for solving euclidean 2d traveling salesrep problem,” 01 2004.

[41] L. Biewald, “Experiment tracking with weights and biases,” 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/

List of Figures

List of Tables