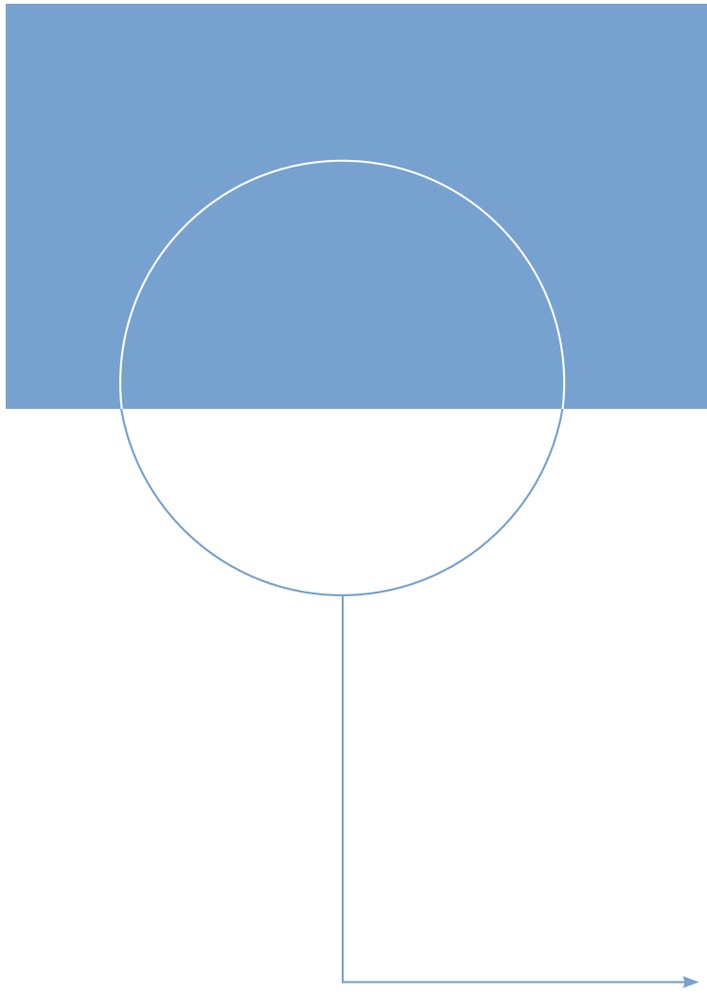


Master's thesis

2020

Henrik Kronborg, Stian Horn Stensli

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Henrik Kronborg
Stian Horn Stensli

Multi-objective Discrete Quantum PSO for dynamic environments

June 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Multi-objective Discrete Quantum PSO for dynamic environments

Henrik Kronborg
Stian Horn Stensli

Datateknologi
Submission date: June 2020
Supervisor: Pauline Catriona Haddow

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Dynamic environments and multiple objectives are prominent in real-world problems. However, a combination of dynamic environments and multi-objectives is not extensively studied. In the thesis a model is proposed that combines the dynamic handling algorithm mQSO and the algorithm MODPSO which operates on discrete multi-objective problems. Further, different types of discrete representations are proposed and evaluated. The model is developed to solve the Vehicle Routing Problem (VRP) and is tested on the Solomon data set and a data set from the Norwegian home care. The model attempts to solve dynamic VRPs where changes to the environment like the appearance of new customers or demands can occur. The dynamic abilities makes the model able to transfer knowledge from before to after a change which can reduce computational cost. The proposed model is able to handle these changes while optimizing multiple objectives and handling constraints such as time windows.

Sammendrag

Dynamiske miljøer og flere objektiver er gjentagende problemer i den virkelige verden. Det er ikke gjort grundig forskning på en kombinasjon av disse problemene. I oppgaven blir det lagt frem en modell som kombinerer den dynamiske algoritmen mQSO og algoritmen MODPSO som operer på diskrete og flere objektiv problemer. Det blir også foreslått og evaluert ulike diskrete representasjoner. Modellen er utviklet for å løse Vehicle Routing Problem (VRP) og er testet på Solomon datasettet og et datasett fra den norske hjemmesykepleien. Modellen forsøker å løse dynamiske VRP'er hvor endringer slik som at nye kunder eller behov kan dukke opp. De dynamiske egenskapene lar modellen overføre kunnskap fra før en endring skjer til etter som kan være med på å redusere nødvendig prosesseringstid. Den foreslåtte modellen er i stand til å håndtere disse dynamiske endringer og samtidig optimalisere flere objektiver og ta hensyn til begrensninger slik som tidsvinduer.

Preface

The thesis is the resulting work of a master thesis in artificial intelligence at the Norwegian University of Science and Technology in Trondheim. The master thesis has been worked on in the period of 15.01.2020 - 10.06.2020.

We would like to thank our supervisor Pauline Catriona Haddow for the guidance on the project. The supervisor had weekly meetings where thorough feedback was given. In addition the supervisor was available with a quick answer to any question asked over email.

Henrik Kronborg and Stian Stensli
Trondheim, June 9, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The chapter elaborates the background for the thesis in section 1.1, followed by the goal and research questions in section 1.2. In section 1.3 the structured literature review process is presented, followed by the preliminary process in section 1.4. Finally, the structure of the thesis is presented in section 1.6.

1.1 Background

The thesis centers around solving the problem of vehicle routing with multiple objectives in a dynamic environment. The Vehicle Routing Problem (VRP) is a problem that aims to find optimal routes for a set of vehicles that have to serve a set of customers. It is often seen in the context of delivering goods from a depot to customers that have placed orders. It is required that the vehicles return to their depot after serving the customers. The objective is often to minimize the route lengths between all vehicles.

Finding optimal solutions to VRPs are considered to be NP-hard. The thesis makes the addition of constraints, multiple objectives and a dynamic setting which will further increase the complexity of the VRP. Bio-inspired algorithms were chosen to handle this problem. These algorithms are inspired by concepts found in nature, such as evolution or social behavior. From these concepts the algorithms has extracted simple, yet elegant, ways of solving complex problems. Bio-inspired algorithms are not guaranteed to find optimal solutions, but may be tuned to find adequate solutions in a reasonable time frame.

The thesis will specifically focus on the bio-inspired algorithm Particle Swarm Optimization (PSO). PSO is inspired by bird flocks and is classified as a swarm intelligence algorithm, a paradigm of bio-inspired algorithms. PSO uses particles which moves around in what is called a search space, where each position of the

search space is related to a solution. The algorithm is therefore attempting to find the position that is related to the optimal solution. Alone each particle exhibits simple behavior, but as a swarm they can find optimal solutions to complex problems.

PSO operates in a continuous search space, while a VRP is a discrete problem. Specifically, the positions of particles in the classical PSO has their positions defined by a set of rational values. Each position is then converted to a solution, which means that the classical PSO allows for an infinite number of solutions. However, discrete problems such as a VRP only allows for a finite number of solutions, even though that number may be extraordinarily large. To solve this, and still be able to utilize the strengths of PSO, some algorithms such as the Set-based PSO (S-PSO) convert the search space to be discrete through a set of integers. When constrained such a search space may allow for easy translation between search space positions and VRP solutions.

Multiple studies have used PSO to solve the Vehicle Routing Problem (VRP). However, few have researched the VRP in a dynamic environment or in other words a VRP that may change over time. These changes can involve new customers or demands that appear in real-time as the algorithm is running. It is often the case in real-world applications that changes can occur, no matter how well planned the original routes may have been. One algorithm designed around dynamic environments is the multiple Quantum Swarm Optimization (mQSO). The algorithm defines a new type of particle for the PSO algorithm to be able to continuously search for changes in the search space caused by a dynamic change. This algorithm is however created for a continuous environment, causing the need for an adaptation when used to solve the VRP.

In addition, a VRP may have several factors that needs to be minimized. Two of the most common are total driving distance and the number of vehicles used. The complexity of multiple objectives lies in the prioritization between objectives e.g. how does one compare time to money. The exact prioritization is completely subjective. The Multi-objective Discrete Particle Swarm Optimization (MODPSO) algorithm attempts to solve multiple objectives in a discrete problem area. The algorithm handles multiple objectives by converting them to several single objective problems. The single objective problems will then have a known prioritization between the objectives allowing them to be easier to solve separately. When the algorithm meets a stop criteria the results from the single objective problems are combined back to a set of solutions.

The thesis proposes a model based on the mentioned algorithms in an effort to solve a discrete, dynamic and multi-objective VRP.

1.2 Goals and Research Questions

Goal *Explore a combination of a Multi-objective Discrete PSO and multi Quantum Swarm Optimization (mQSO) that optimizes performance and lowers computational cost in a dynamic environment.*

Description: The proposed model is from here on referred to as MODQPSO.

Research question 1 *How can modifications to the search space improve performance while decreasing computational cost?*

Research question 2 *How can reference spans improve performance while maintaining the computational cost when handling multiple swarms in multi-objective optimization?*

Description: Reference spans assign a range of possible objective prioritizations for each swarm. This causes all particles within the same swarm to have a similar prioritization of the objectives.

Research question 3 *How does dynamic environments affect MODQPSO in terms of computational cost and performance?*

1.3 Structured Literature Review Protocol

This section elaborates the process of how the literature has been researched and how the goals and research questions of the thesis were identified. It also presents the inclusion criteria for what papers were to be included, as well as quality criteria that further restricts what papers to include.

1.3.1 Literature sources

The findings from literature have mainly been through the use of Google Scholar. The background theory is based on well established work, while the state of the art is an attempt of presenting more recent publications. Work published through the Gecco and IEEE conferences have been preferred. The citation index found on WebOfKnowledge has also been used where one can sort to find highly cited articles.

In an attempt to answer the research questions some keywords, quality criteria and inclusion criteria are defined to asses the articles value for the research. These are presented in table 1.1 and was used to select relevant and credible sources.

Identifying	**Keywords used when searching** • Multi objective, PSO, discrete, VRP, dynamic, dynamic environment, mQSO, quantum, dynamic VRP
Qualifying	**Inclusion Criteria** • The study's main research topic is Swarm Intelligence. • The study focuses on multi-objectives or dynamic environments. • The study appears relevant based on title and abstract.
Evaluating	**Quality Criteria** • The study is released by a recognized publisher. • There is a clear statement of the aim of the research. • The models design decisions are explained. • The model should be compared to state-of-the-art.

Table 1.1: Paper selection criteria.

1.4 Preliminary Process

Figure 1.1 illustrates the path to finding a project. The project provided by NTNU allowed the students to form their own project. Hence, the goal of the preliminary process was to find and explore areas that could be evolved inside of the paradigm of bio-inspired algorithms.

Within bio-inspired algorithms the genetic algorithm is to be considered the most common to solve VRPs. However, the problem area is already extensively researched. During the preliminary process swarm intelligence algorithms appeared to be a trending strategy for solving VRPs. The most published swarm intelligence algorithm is PSO. It is also a extensively studied algorithm, but there are areas which still lack exploration e.g. constrained search spaces. The PSO algorithm is continuous, so there are even less studies that have looked at discrete adaptations. Other alternatives within swarm optimization to solve VRPs was Ant Colony Optimization (ACO). Compared to PSO, ACO has a more natural bond to discrete problems. It was also discovered that ACO has been used

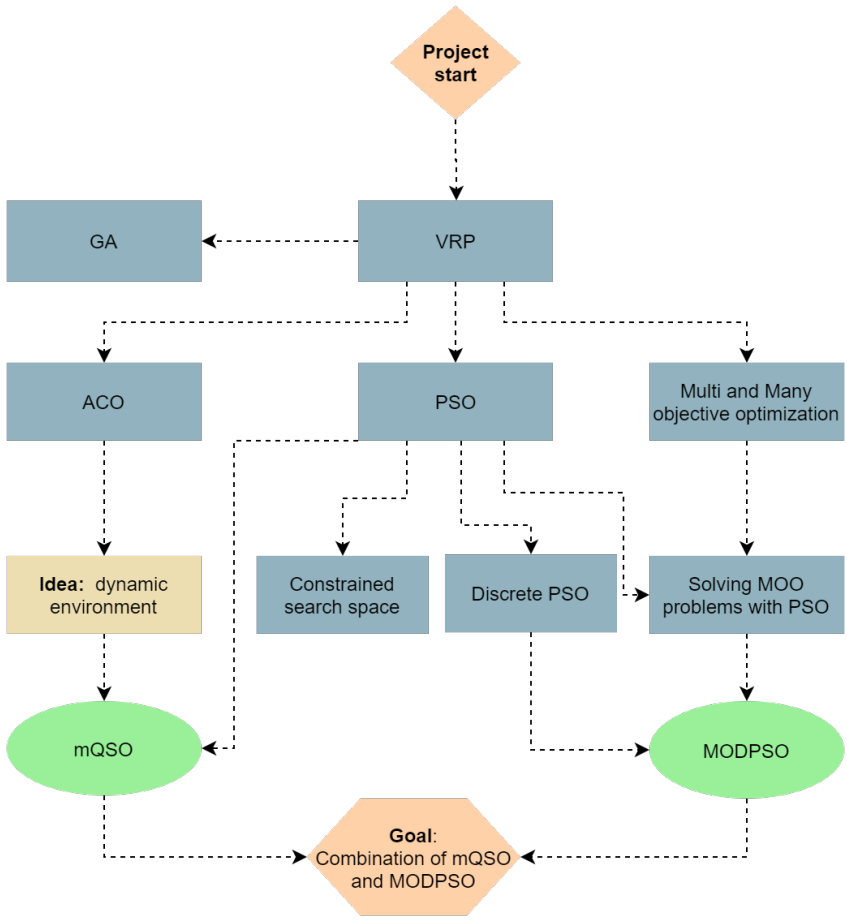


Figure 1.1: Overview of the path to the current research goal.

in dynamic application areas. However, during research there was discovered promising research articles to create a hybrid using PSO. The goal was to find an edge to the research to both contribute to solving VRPs and to evolve PSO.

After deciding to use the PSO algorithm some complexity was added by considering a constrained and multi-objective data set provided by Visma. The data set is a VRP that assigns health care workers to the elderly.

Reading about dynamic environments solved by using ACOs sparked the idea of solving it with PSO. Further research of dynamic environments within PSO suggested that it was a less explored area. The research led to the discovery of Blackwell's work and how it has evolved over the years. Blackwell's algorithm multi-Quantum Swarm Optimization was particularly inspiring and is still relevant as the algorithm is cited and expanded upon in recent publications.

The Visma data set allows for many-objective optimization i.e. handling more than three objectives. Many-objective optimization often require several layers of added complexity compared to multi-objective optimization. It was decided that the scope of the project would be too large when also researching dynamic environments.

The preliminary process resulted in the goal of creating a discrete, dynamic and multi-objective PSO algorithm that would reap the benefits from each component.

1.5 Pre-project

In the fall of 2019 a pre-masters project [Kronborg and Stensli, 2019] with the same topic was performed. Most of the background theory is taken from the pre-project. The state of the art is based on the pre-project, but is is rewritten and evaluates a wider set of articles.

1.6 Thesis Structure

Consecutive chapters are presented as following: Chapter 2 contains the background theory, where fundamental concepts are explained, followed by state of the art for dynamic and multi-objective techniques in PSO and the motivation for the thesis. Chapter 3 uses the state of the art to create a model and presents the proposed model. In chapter 4 the simulator for the model is displayed along with the experimental plan, setup, results and evaluations. Chapter 5 concludes the performed research and elaborates possibilities for future work.

Chapter 2

Background Theory and Motivation

The chapter elaborates the thesis background theory in section 2.1, as well as presenting the state of the art for dynamic and multi-objective techniques in PSO in section 2.2, before the motivation in section 2.3.

2.1 Background Theory

2.1.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a problem in computing where a set of vehicles are utilized to serve a set of customers. In the standard VRP each customer has to be visited once by any of the vehicles. The goal is to design routes that minimizes travel distance of the vehicles.

$$min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} d_{i,j} x_{i,j,k} \tag{2.1}$$

The standard objective of a VRP is defined by equation (2.1) where k is a vehicle from the set of vehicles V, and both i and j are customers from the set of all customers N. The depot is often denoted as index 0 of N. $d_{i,j}$ is the distance between customer i and j and $x_{i,j,k}$ is either 1 or 0 defined by whether or not the solution traverses between customer i and j by vehicle k. In addition some rules usually apply such as that all customers in N must be visited by one of the vehicles in V and that all vehicles must start and end at a depot. Finding optimal solutions that both follow these rules and minimize the objectives of the VRP are considered to be a NP-hard problem.

VRP: Heuristics

A heuristic is a technique that seeks improved solutions at a reasonable computational cost without being able to guarantee that a solution is optimal, to state how close to optimal a particular feasible solution is or in some cases even to guarantee feasibility [El-Sherbeny, 2010]. An important type of heuristics are route-building heuristics that may be used to find good solutions by being provided a set of customers in a particular order. Consequently the algorithm only has to find a fitting order for the customers to appear and then utilize a heuristic to find the best vehicles to assign the customers to. This process may decrease the algorithms ability to produce all possible solutions, but in turn causes an improved fitness in the produced solutions.

A specific example of route-building heuristics is the "parallel tour construction algorithm" from Jurgen Antes and Derigs [1995]. The heuristic connects specific orders of customers to vehicles, where the connection between an order of customers and vehicles are referred to as a vehicle route. In the heuristic there is an auctioning process that generates the fittest solutions. The auctioning process calculates the cost of inserting every available customer in to each vehicle route. After retrieving all available costs of insertion, a single customer is inserted to the vehicle route connected to the lowest cost. The auctioning process is completed when all customers are inserted in to a vehicle route. The result is a set of vehicle routes that represents the solution to the VRP. The parallel tour construction algorithm is deterministic and hence will generate no solution diversity on its own.

2.1.2 Particle Swarm Optimisation

Particle swarm optimisation (PSO) was proposed in 1995 by James Kennedy and Russell Eberhart [Kennedy and Eberhart, 1995]. The idea is inspired by movement of large flocks of animals that as a group display complex behaviors e.g. bird flocks and schools of fish. PSO was originally formed to simulate bird flock behavior. The conversion to an optimization algorithm happened with the implementation of a feature that made the bird flock locate and land at food sources. This evolved to an algorithm where particles are placed in to a search space where they move around searching for optimal positions. Their movement are decided through a combination of wanting to go back to their own previous best position and to the global best position found by the swarm.

$$v_i(t+1) = \omega v_i(t) + c_1 ran_1[p_i(t) - x_i(t)] + c_2 ran_2[g(t) - x_i(t)] \tag{2.2}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.3}$$

Equation (2.2) is used for velocity updating of each dimension of the search space. Equation (2.2) is a refined version of the 1995 algorithm as seen in [Eberhart et al., 2001]. This equation is more commonly referred to as the velocity update function.

The velocity of a particle, seen in equation (2.2), is updated through three parts. The first term represent friction for particle i where the velocity of i at time t marked as $v_i(t)$ will be slowed down by the factor ω. The factor ω is recommended to be in the range of $\langle 0, 1 \rangle$. The second and third term represents how the particle is pulled towards its personal best $p_i(t)$ and global best $g_i(t)$ position by subtracting the current position $x_i(t)$. The constants c_1 and c_2 is set at initiation and represents what degree the particles will favor the personal best and global best positions. It is often recommended that the sum of parameters c_1 and c_2 is equal to 2 at initialization. The variables ran_1 and ran_2 are random values in the range of $\langle 0, 1 \rangle$. After a velocity have been calculated the position of a particle is updated as seen in equation (2.3), here the position is updated by adding the velocity to the position.

In PSO the search space is defined as a multidimensional space for which particles are positioned. Each position in the search space correlates with a solution to a desired problem. To know how well each position solves the desired problem objective functions are used to calculating fitness scores. The score given by each objective function defines where in the objective space a solution is positioned. If a problem only has a single objective then this position is defined by a single number e.g. driving distance. The objective scores found at a position in the search space is utilized when the velocity is calculated. If a found position in the search space correlates to a good objective score it may be selected as a personal best position for a particle or even a global best position for the swarm. As the particles traverse the search space they explore the correlation between the search space and objective space. If the PSO is successful the swarm will center around areas of the search space that is linked to the optimal objective scores for the given problem.

The search space should be manipulated or designed to work well with a given problem. Objective functions may translate positions of the search space directly to objective scores. However a indirect mapping trough the use of a decoding process may improve the objective scores at each position of the search space e.g. the decoding process may utilize heuristics. An indirect mapping of the search space to an objective space does demand more computational power, but is commonly used to be able to handle non-continuous problems or to manipulate the search space to further improve the fitness of each position.

2.1.3 Dynamic environment

Some problems have the trait of changing over time. In PSO this can be observed if the optimums change positions in the search space. These types of problems are often referred to as Dynamic Optimization Problems (DOP).

There are two strategies when dealing with DOPs [Mavrovouniotisa et al., 2017]:

1. *Complete restart:* When a change is detected, the algorithm is restarted with a modified environment. This strategy makes a DOP indistinguishable from a series of static optimization problems. A disadvantage with this approach is that no knowledge is transferred between each change.

2. *Partial restart:* The concept of partial restarting is to maintain knowledge after the occurrence of a change. Several different methods can be used, but the essence is to have sufficient diversity to produce a new solution near the optimum with less computing power than if a complete restart was performed. This way one can say that the algorithms are able to learn how to produce good solutions in an environment.

In a dynamic VRP, swarms may learn beneficial routes that will still be useful after a new event. In a VRP an event could be a new customer added to the route. When the new routes are computed with the new customer as a factor, the swarm will in theory be able to find an optimal solution faster than after a complete restart. Different techniques will change how well an algorithm is able to retain knowledge of the problem area.

There are two major factors in how a dynamic environment operates; the severity of the spatial change and temporal severity. The former is how large effect each change has and the latter is how often each change occurs. There is also two key points to consider of outdated memory and loss of diversity. All this is crucial when discussing what technique to incorporate to an algorithm. Hence, it is important to know how often a new customer is added (temporal) and on average how big of a change that makes in the search space positions of the optimums (spatial).

Different dynamic environments will have different rates of spatial and temporal changes. Because of the importance of spatial and temporal values, a system was proposed in Duhain and Engelbrecht [2012] to label environments in four different classes. The list below explains the different classifications and in figure 2.1 one can observe the relations between the classes.

1. *Quasi-static environments*: both low spatial and temporal changes.

2. *Progressive environments*: low spatial, but frequent temporal changes. Changes occur often, but does not have a large effect on the location of optimums.

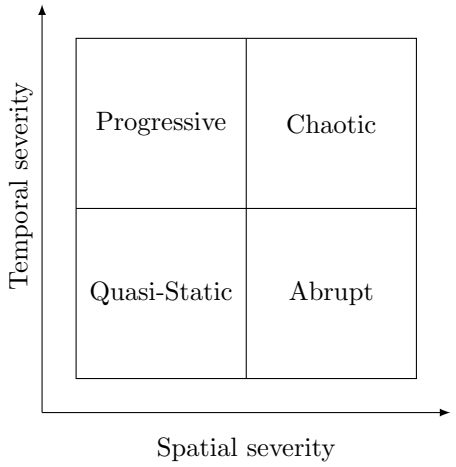


Figure 2.1: Types of classifications for Dynamic environment [Unger et al., 2013].

3. *Abrupt environments*: large spatial, but infrequent changes. The environments is perceived as static for long periods of time before large changes occur.

4. *Chaotic environments*: large spatial and frequent changes. The environments will have numerous changes with large impacts.

2.1.4 Multi-objective

Multi-objective problems are optimization problems that involve more than one objective function. The goal is to find a set of solutions with the best tradeoff between the objectives.

An example on multi-objectives is picking the best possible flight. Making an algorithm that achieves this for a range of customers would be impossible, because of the ambiguities surrounding multiple objectives. It is impossible to know if a customer wants to travel with an expensive plane and arrive early, or a cheaper one and save money.

Figure 2.2 illustrates a *Pareto Front (PF)* as green points on a dotted line that correlates with the set of Pareto optimal solutions. Solution p_1 is nondominated (*Pareto Optimal*), if no other solution is found so that for all objective functions that solution is better than p_1. Considering it the opposite way, p_2 is dominated by p_1 since it has both a lower value for objective 1 and 2.

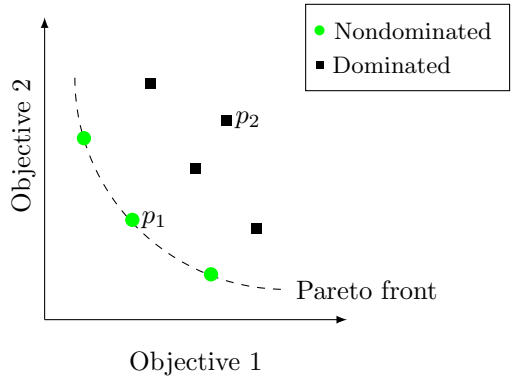


Figure 2.2: Pareto Front

$$\begin{gathered} minimize(f_1(x), ...f_k(x)), \\ x \in S \end{gathered} \tag{2.4}$$

Multi-Objective Optimization Problems (MOOP) can be mathematically defined as in equation (2.4) where $f_k(x)$ represents objective functions, and S represents the feasible region [Miettinen, 1999]. The feasible region is a subset of the search space that contains all solutions that comply with the given constraints. A solution is ranked by all given objective functions. The objective of a MOOP is then to find solutions that optimizes all of the objective functions, where in the case of equation (2.4) optimization is equal to minimization.

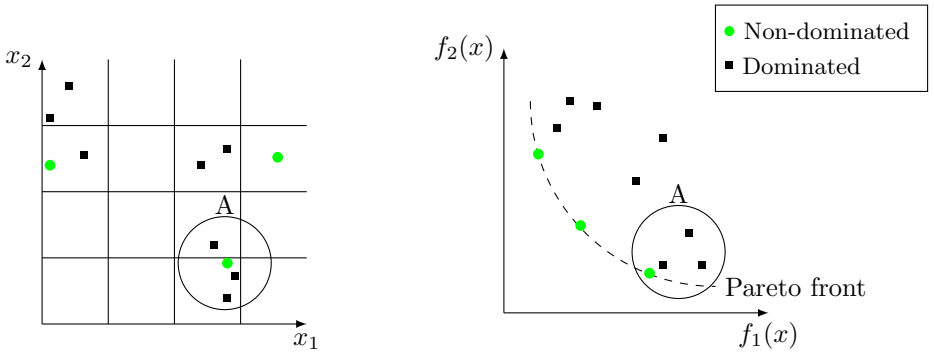
Hypercubes

The Multi-Objective Particle Swarm Optimization (MOPSO) algorithm proposed in Coello Coello and Lechuga [2002] divides the search space into multi-dimensional cubes called hypercubes. With the use of hypercubes the authors assume that particles converge at different positions in the search space and that these positions are linked to different parts of the Pareto Front. This is called a geographically-based approach of maintaining diversity.

The algorithm uses an archive containing solutions that are nondominated. Whenever a particle finds a new solution it is compared to the archive of non-dominated solutions and if the new solution is nondominated it is stored in the archive. Solutions that already exist in the archive that are dominated by the new solution are removed.

Hypercubes are represented in figure 2.3a as squares between the axis. Particles only communicate with other particles within the same hypercube. This

communication consists of sharing their local best and agreeing on a global best. The global best is chosen through Pareto dominance and in the case where there are two or more nondominated particles in a single hypercube one is chosen at random. Circle A represents the same particles in the search space and objective space, illustrated in figure 2.3a and figure 2.3b respectively.



(a) Search space with Hypercubes (b) The resulting objective space

Figure 2.3: MOPSO: Hypercube representation

Hyper Volume Indicator

The Hyper Volume Indicator (HVI) is a measure of the area that a Pareto front covers in the objective space [Beume et al., 2009]. HVI by itself is not enough to determine if one MOOP algorithm is superior to another as a large HVI may represent an ability to find superior solutions in a single part of the Pareto front, while another part may not have been explored. However, when an algorithm shows a superior HVI compared to another algorithm it reveals that the superior algorithm is able to at least produce one solution that dominates the solutions from the other algorithm. Hence, HVI can be an indication of how well algorithms are doing, but does not provide a guarantee.

HVI is a calculated with the use of a reference point. When the Pareto front is known this reference point can be set to the nadir point. The nadir point is found by using the worst objective values in each dimension from the true Pareto front i.e. finding the point that is dominated by all other points. If the true Pareto front is unknown an approximation of the nadir point can be calculated using the the worst found objective value for all solutions that are used when calculating the HVI. For instance, if the HVI is calculated when comparing three different Pareto fronts, all three fronts are used when finding the nadir point.

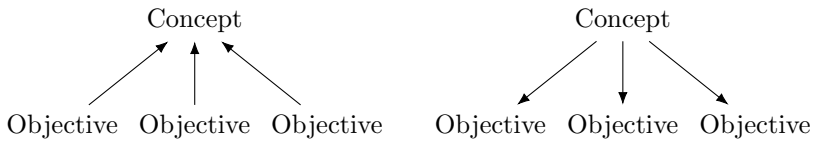
The HVI is then calculated as the volume found between the Pareto front and the reference point.

HVI is usually represented as a number but can also be presented as a percentage of the ideal volume [Zitzler et al., 2007]. The ideal volume is calculated by the use of the ideal point which is a point that dominates all other points in the true Pareto front i.e. the opposite of a nadir point. When the true Pareto front is unknown an approximation of the ideal point can be calculated the same way as for the nadir point. The ideal HVI is calculated by finding the volume between the ideal and nadir point. When using HVI a simpler comparison can be made by diving the HVI by the ideal value.

Decomposition and Aggregation

Decomposing multi-objective algorithms converts a MOOP to a set of single objective sub problems. Each sub problem have a function that determines a varying prioritization of the MOOP. The different strategies of determining prioritization are called aggregation functions, where the most common ones are the Tchebycheff approach and the weighted sum method.

Both the Tchebycheff approach and the weighted sum method relies on the possibility of generating a system where the sub problems can individually be assigned to solve their own single objective task. After convergence the sub problems must also be able to be aggregated to form a Pareto front. The relation between decomposition and aggregation can be seen in figure 2.4.



(a) Aggregation (b) Decomposition

Figure 2.4: Difference between aggregation and decomposition.

$$\begin{aligned} &\text{minimize } g^{ws}(x|\lambda) = \sum_{i=1}^{m} \lambda_i f_i(x) \\ &\text{subject to } \ x \in \Omega \end{aligned} \tag{2.5}$$

The weighted sum method generates a set of random uniform weights for each sub problem, where each weight is tied to a single objective in a MOOP. This is seen in equation (2.5), where λ represent a set of weights and f_i an objective function. Having different sets of weights result in finding solutions to different

parts of the pareto front. The weighted sum method does not perform well in complex objective spaces, but it is less computational heavy than Tchebycheff approach [Zhang and Li, 2007].

Each sub problem in the Tchebycheff approach is not only dependent on a set of weights as in weighted sum, but also on the best found value in each objective found by the swarm. This distinction makes the Tchebycheff approach able to operate in both complex and non-complex search spaces.

$$\begin{gathered} \text{minimize } g^{te}(x|\lambda, z^*) = \max_{1 \leq i \leq m} \{\lambda_i |f_i(x) - z_i^*|\} \\ \text{subject to } \ x \in \Omega \\ z^* = (z_1^*, \ldots, z_m^*)^T, \text{ where } z_i^* = \max\{f_i(x) | x \in \Omega\} \end{gathered} \tag{2.6}$$

The Tchebycheff approach is represented in equation (2.6). The randomly distributed set of weights is symbolized by λ and z^* which is a set of reference values. The reference values are the best objective values found in each objective.

Each sub problem forms its own objective function represented as $g^{te}(x|\lambda, z^*)$, as seen in (2.6). This function is what makes the search for solutions to each sub problem dependent on both the reference values (generated from the whole population) and a weight vector λ, which is the prioritization of the objectives. The dependency of z^* makes the swarm share information about the search space.

2.1.5 Constraints

A constraint set limits on available choices. Typical constraints in a VRP are that the vehicles will have limitations such as driving distance and load limits. These limitations are commonly referred to as constraints.

The list below presents some of the commonly found constraints within VRPs.

- *Vehicle types:* Vehicles have properties making them unique, making them fit for different tasks.
- *Time windows:* A vehicle or customer is only available during specific time intervals.
- *Visit restrictions:* Some vehicles can not visit certain customers.
- *Synchronized tasks:* Some customers must be visited by two or more vehicles at the same time.
- *Load limits:* A vehicle may have a load capacity restricting the number of customers visited.

These constraints can be of the following two types:

1. *Soft constraints* are preferred to be met, but a breach of one will not invalidate a solution.

2. *Hard constraints* have to be met for a solution to be valid.

Soft constraints can be solved through the use of multiple objectives. Implementing soft constraints as objectives allows the algorithm a larger set of feasible solutions, as the algorithm is allowed to create solutions that tries to minimize the soft constraint violations. For instance if the time window constraint is regarded as soft the goal is to minimize the deviation from breaching the time windows. This particular method is known as an indirect constraint handling technique [Eiben and Smith, 2015].

2.2 State of the art

2.2.1 Dynamic PSO

The optimal method of solving problems in dynamic environments will differ depending on the dynamic classification of the given environment. The dynamic classifications are explained in section 2.1.3. However, methods made for one classification may also work well in another.

A model proposed in Demirtaş et al. [2015] is designed to operate on abrupt or quasi-static environments, due to how the algorithm converges after a static problem is generated. Each static problem is generated after the occurrence of a change and is a representation of the current environment. The search space allows the insertion of customers dynamically anywhere in the position vector. However, transfer of knowledge between the static problems is not discussed. If the swarm converges in every static problem some method of maintaining diversity will be required to continue the search for a better optimum. This indicates that the model is designed around an environment of low temporal severity where convergence is possible between each change. The model presumes that there is no need to track multiple optimums, as each static problem should only contain a single best optimum. This is not assumed in most other dynamic work.

A common issue in dynamic environments are how a local optimum may change to be the global optimum after a dynamic change. A common strategy to handle this issue is to utilize multiple swarms. The specific implementations of which may vary.

One such model is the Multi-Adaptive PSO (MAPSO) proposed in Khouadjia et al. [2010]. The idea is to place a MAPSO swarm on each local optimum. Ideally these swarms will maintain enough diversity to track dynamic changes even after a convergence. The swarms evolve in a cooperative island model where the

population is decomposed into sub-populations to be able to track different optimums. These sub-populations, also called islands, exchange information about found optimums through particle exchange. An elitist replacement strategy is used to replace the worst particles of a swarm after each exchange. Since the algorithm is composed of several different techniques, further improvement may prove to be difficult as it already requires tuning of a large number of parameters for optimal results.

Another multi swarm method is the Species-PSO Parrott and Xiaodong Li [2006], where the different swarms are referred to as species. Each species is based on proximity so that particles may change what species it belongs to based on its location in the search space. A maximum number of particles per species prevents all particles from converging to a single location and simultaneously force the particles to search different areas. Species-PSO has been used to solve dynamic problems and presented good results. However, if the changes happen infrequently then the different swarms will converge which causes the algorithm to struggle finding the new optimums. This issue is addressed in Bu et al. [2017] where memory techniques are utilized to insert diversity. Particles from before a convergence will retain its velocity and be able to continue the search for optimums and even make a whole species re-initiate its search. This particular memory technique works best in environments where changes cyclically occur, allowing the stored particles to find old optimums that once again are superior.

The multi Quantum Swarm Optimization algorithm (mQSO) introduced in Blackwell and Branke [2004] is designed around tracking a range of different optimums. The algorithm is based on the ideas of quantum optimization, that is a meta-heuristic primarily used to optimize the local search abilities of an algorithm [Karmakar et al., 2017]. This makes the local search in mQSO preferable to the one in standard PSO.

The meta-heuristics of quantum optimization is inspired by real world quantum mechanics. In quantum mechanics one can at no point be certain of a particles position or velocity due to the uncertainty principle. Early work such as Sun et al. [2004] introduced algorithms that where heavily inspired by this principle and paid close attention to the actual rules of quantum mechanics. The early quantum algorithms have more complex calculations than the standard PSO, but in turn presents better local search abilities without a loss of global search. These quantum algorithms were not optimized for dynamic environments, but rather focused on optimizing the results of the standard PSO. Quantum PSO increases the diversity which also increase the search ability of PSO [Karmakar et al., 2017].

The mQSO algorithm was designed to utilize the quantum heuristic to help restart the algorithm when a change occurs, with the added benefit of optimized search abilities. The multi-swarm part of mQSO is more strict than algorithms

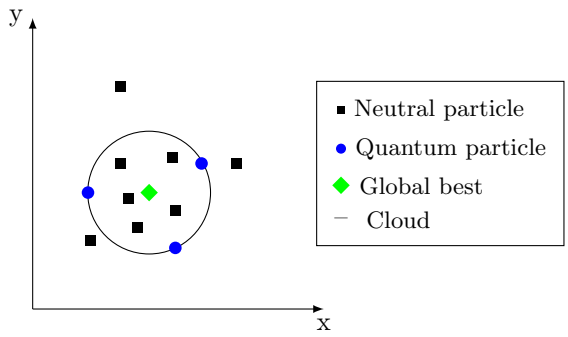


Figure 2.5: QSO: Neutral and Quantum particles

like Species-PSO. No information is transferred between the swarms, besides the competition that occurs when two swarms collide. In the competition, the winner is chosen at random and allowed to converge, while the loser is re-initiated at a random position. Later publications on mQSO increase the focus on finding new optimums [Blackwell and Branke, 2006]. When all swarms have converged, the worst performing one is re-initiated at a new location in the search space. This will have the effect of creating continuous search for new and better optimums.

Making mQSO work with a discrete search space requires modifications as the algorithm is designed to work in a continuous one. However, other quantum techniques have been used to optimize the search abilities in discrete problems with promising results seen in [Li et al., 2017]. However, these have not focused on the promising dynamic traits of quantum.

Quantum Inspired PSO

Compared to other quantum inspired algorithms, mQSO simplifies the complex calculations needed for finding a quantum particles position. The proposed method uses two types of particles: neutral and quantum particles.

1. *Neutral particles* are the particles used in the standard PSO.

2. *Quantum particles* have no velocity function and the position is simply updated with a function that has a radius of r and with the number of dimensions d, equal to that in the search space.

   The function will make quantum particles appear at random locations with a distance r away from the global best position. This introduces a random search around the convergence point of a swarm.

The relation between neutral particles, quantum particles and the global optimum is illustrated in figure 2.5. The global best particle at time *t+1* can be a position found by both a neutral and a quantum particle at time *t*. When a new global best position is found, the radius is moved to be centered around it at time *t+1*.

The behavior of quantum particles makes their search independent of velocity, which is utilized in mQSO to avoid the problem seen in Species-PSO. In the particular problem all particles velocity are reduced to 0 causing them to get stuck at a singular point. In experiments done on mQSO the quantum particles are able to reintroduce velocity to a swarm after a dynamic change through their behavior of continuous search around the optimum.

2.2.2 Discrete PSO

In PSO all particles have to traverse a continuous search space in the search for an optimal solution, as discussed in section 2.1.2. However, in problems such as the knapsack problem, traveling salesman and VRP there is no direct benefit of using continuous values. Solutions to these problems are classified as discrete, and the PSO algorithm require extensions or direct manipulation to be able to efficiently find solutions to discrete problems.

An early proposed method is to round each value in a particles position to an integer, as utilized in Improved PSO (I-PSO) [Qing Zhu et al., 2006]. I-PSO is specifically designed around the VRP. It requires the dimension of the position vector to be *n+k-1*, where n is the number of customers and k is the number of vehicles. The position represents the sequence of customers each vehicle traverses where a depot is positioned in the sequence to mark the use of a new vehicle. The computational complexity of PSO decreases with the number of dimensions in the position vectors. However, a larger vector may be efficient if it can remove complexity related to the objective functions, in tasks such as decoding the position or maintaining valid solutions. The position vector of I-PSO makes it able to find all valid solutions, even the most inefficient. Since I-PSO is designed for the VRP, some constraints are solved by the particle itself. An example is to satisfy all customer demands, where I-PSO ensures that all particle positions visit every customer in the VRP.

Another method is to modify the search space directly, removing the need of converting position values from continuous value to discrete, as observed in the Multi Objective Discret PSO algorithm (MODPSO) [Gong et al., 2013]. Both a particles position and velocity are defined to discrete values, this is achieved by modifying the velocity and position update function. The proposed discrete velocity and position update is similar to the original PSO formulas, but the modifications lessens the computational complexity of converting a continuous

search space to a discrete one.

There are no published articles that use MODPSO to solve a VRP. Other methods such as I-PSO have the benefit of being proven to work at solving the VRP and to have been later expanded to increase the efficiency using techniques such as Tabu search [Zeng, 2019]. Modifying MODPSO to be able to handle constraints that other algorithms have solved in this search space may prove to be both a complex task and to lessen the benefits from the discrete search space.

An algorithm that has a discrete search space, is designed for discrete problems and proven to be able to solve VRP is the Set-based PSO (S-PSO) introduced in Chen et al. [2010]. The velocity of S-PSO is based on continuous values that are all linked to discrete actions. In a VRP a natural link between these probabilities and actions is whether or not a vehicle drives from one customer to another. The position consist of discrete values, where S-PSO is able to use the probabilities in the velocity when updating the position. In the position update the probabilities of the velocity are manipulated into what is called a set, hence set-based PSO. The concept of S-PSO is expanded in Gong et al. [2012], where S-PSO is altered to operate on a VRP with time windows. In the paper it is shown that the structure of S-PSO can be utilized to solve some VRP constraints by design. In turn the results from S-PSO are compared directly with I-PSO, showing an ability to retrieve solutions with better travel distance in S-PSO.

Set-based PSO

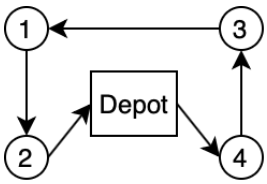
The position of a particle is represented by a 2d array, where the index of each array symbolizes a customer, as seen in figure 2.6a. The first element inside the array represents the previous customer and the second element represents the next. By following all the connections in the representation, a sequence of customers is found creating a Hamiltonian path. The Hamiltonian path of figure 2.6a is drawn as the path 0-4-3-1-2-0 in figure 2.6b, where index zero represents the depot.

The position is in other words defined as an array with the dimensions $2 * N$. The velocity is an array of N^2, containing all possible connections between all customers and the depot. S-PSO keeps track of both the next and previous customer as in some VRPs a route from the depot to customer A and then to B will be equal to the route from the depot to B then to A. In this case marking both the path from A to B and from B to A as optimal, will be favourable when generating a new route. However, when confronted with strict time windows it may be the case that A must be visited before B, where as this behavior is not considered by S-PSO.

Since S-PSO for VRP has a Hamiltonian path as its position, utilizing multiple vehicles in S-PSO requires the use of a decoder. The decoders primary goal is to minimize the number of vehicles used, where travel distance is a secondary

index:	from	to
0:	2	4
1:	3	2
2:	1	0
3:	4	1
4:	0	3

(a) S-Particle representation.



(b) S-Particle Hamiltonian path.

Figure 2.6: S-Particle relation between representation and Hamiltonian path.

prioritization. S-PSO documents positive results in the goal of using as few vehicles as possible and in comparison to I-PSO it achieves a better travel distance. With the inclusions of the decoder S-PSO introduce a new factor to diversity, as the decoder is responsible for being able to generate all optimal solutions, which in algorithms such as I-PSO lays solely on the algorithm itself.

2.2.3 Multi-objective PSO

Multi-objective optimization involves optimizing more than one objective function simultaneously. To make optimal decisions one need to consider trade-offs between two or more conflicting objectives.

One such method is the Hypervolume-based Multi-Objective Particle Swarm Optimizer (MOPSOhv) that uses an archive to store non-dominated solutions. MOPSOhv extends the concept of hypercubes discussed in section 2.1.4. Instead of hypercubes which are multi-dimensional squares, MOPSOhv has hypervolumes which in this case are multi-dimensional rectangles. From each nondominated solution a hypervolume is formed in the objective space, where each volume is restricted based on the closest nondominated solution in each objective. A large volume indicates a less explored area of the Pareto front. This behavior is utilized as MOPSOhv directs particles towards areas with large hypervolumes. More specific the particle with the largest volume will be chosen as a leader for other particles. In addition, MOPSOhv stores all nondominated particles in an archive. The method does not present groundbreaking results, although it adds another layer of complexity. The computational cost is high and increases with the number of objectives. However, it does become viable at 6 or more objectives where it may be regarded as the state of the art.

There are a large number of different strategies to multi-objective PSO, where as hypercubes and decomposition are two of the most prominent. In Zapote-

cas Martínez and Coello Coello [2011] a decomposition based PSO algorithm is introduced. The interesting aspect of this algorithm is that it requires no form of archive. The decomposition-based Multi-Objective PSO (dMOPSO) is based on the Tchebycheff approach, and performs a search without any mutation or particle re-initializing. This means that the final position of all particles will be as close to the Pareto front as the algorithm ever got. This strategy may work well in a simple search space, but a more complex one will likely be challenging. It does however introduce an interesting viewpoint in how decomposition guides particles towards the Pareto front.

Multi-Objective Discrete Particle Swarm Optimization (MODPSO) was introduced in Gong et al. [2013]. MODPSO uses the Tchebycheff approach of decomposition, but does require an archive as it includes a random mutation through what is called a turbulence factor. The turbulence is implemented to increase the swarms diversity. It may change a particles position at random to be more similar to one of its neighbors. Each particle has a set of neighbors that is made up of the particles with the most similar weight vectors compared to its own. A weight vector is a specific prioritization of the objectives. The turbulence factor is reported to work well in the given problem, where local optimums were prevalent. MODPSO performs decomposition as explained in section 2.1.4, with the exception of the leader selection strategy and turbulence factor.

The neighborhoods of MODPSO allows particles to communicate with other particles with a different prioritization of the objectives. Communication between particles with similar weight vectors guides particles to converge at optimums. Communication between particles with different weight vectors introduces diversity that can help particles escape local optimums. Each trait is tuned by the size of the neighborhood where a large neighborhood gives more diversity while a small one makes for faster convergence.

Leader selection

Different leader selection strategies can be used in PSO. The leader selection is highly relevant to the current state of the art of multi objective PSO. In PSO a leader can take the place as the global best in the velocity update function, as there may be several candidates for a global best in multi-objective problems. In MODPSO the selection of a global leader was performed by randomly picking the local best position from a set of particles with similar decomposition priorities i.e. picking from a neighborhood. A more computational heavy strategy was considered, but discarded due to high run time and nearly indistinguishable improvements of performance.

In Nebro et al. [2013] a study was performed around the difference of picking random leaders against picking a leader based on hypervolume. The study found it preferable to pick based on hypervolume. However, randomly deciding a leader

that was in the Pareto archive performed surprisingly well. The results varied from problem to problem, but indicated that a random selection of a leader may be the most efficient and a valid strategy for other algorithms which can be observed in MODPSO. More importantly the results indicates that a selection strategy should not be taken for granted as random decisions, hypervolumes or Pareto based strategies all have their place with a given algorithm or problem.

MODPSO retrieved efficient solutions with the use of decomposition and a random leader selection. The authors endorse this strategy as it is less computational heavy than a hypervolume strategy. With the random strategy the algorithm manages to run more iterations in the same time frame as it would if using the hypervolume strategy. The random strategy also produces fitter solutions in the same time frame as the hypervolume strategy.

2.3 Motivation

Through the state of the art some similarities are discovered. The MODPSO algorithm from section 2.2.3 includes a turbulence factor that performs a search that will help the algorithm escape local optimums. Though not identical, the quantum traits of mQSO does have similarities to the turbulence factor and may work as an alternative when using decomposition in future work.

Importantly, the use of multiple swarms in mQSO is already optimized to find and track different optimums. The proposed algorithm will attempt to utilize this trait when faced with multiple-objectives and to optimize the solutions in the case of dynamic changes.

Chapter 3

The Model

The models topology is presented in section 3.1.1, followed by an explanation of the model flow in section 3.1.2. Further in section 3.2 the design choices made to create the model from combining and restructuring other algorithms are presented.

3.1 Model overview

The proposed model is the Multi-Objective Discrete Quantum PSO (MODQPSO). The model combines different techniques such as multiple swarms, decomposition of multiple objectives and quantum particles in an effort to solve a discrete and dynamic VRP.

3.1.1 Topology

MODQPSO is implemented using four different components where the relationship between them can be observed in figure 3.1.

The algorithm is initiated from the MODQPSO component. The component utilizes a set of swarms to search different regions of the Pareto front.

The MODQPSO component ensures that the entire Pareto front is searched by decomposing the main problem through the use of reference spans. Reference spans restricts the possible prioritizations of objectives. Each swarm has its own reference span, so when the model is set to use three swarms it will generate three reference spans which can be observed in figure 3.2a. The spans marked as *S1*, *S2* and *S3* are contained using vector pairs that define the edges of a span e.g. *S1* has a and b while *S2* has b and c. The two outlying vectors a and d mark the edges of all possible span values. This is because a and d represents a prioritization

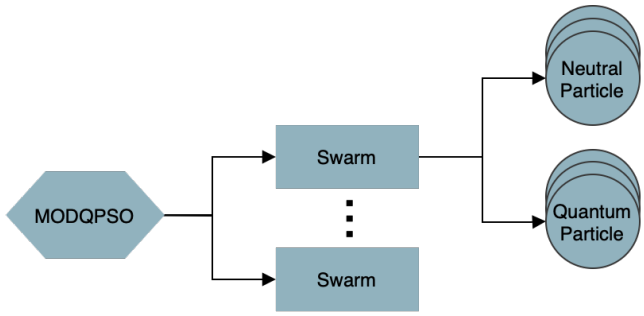
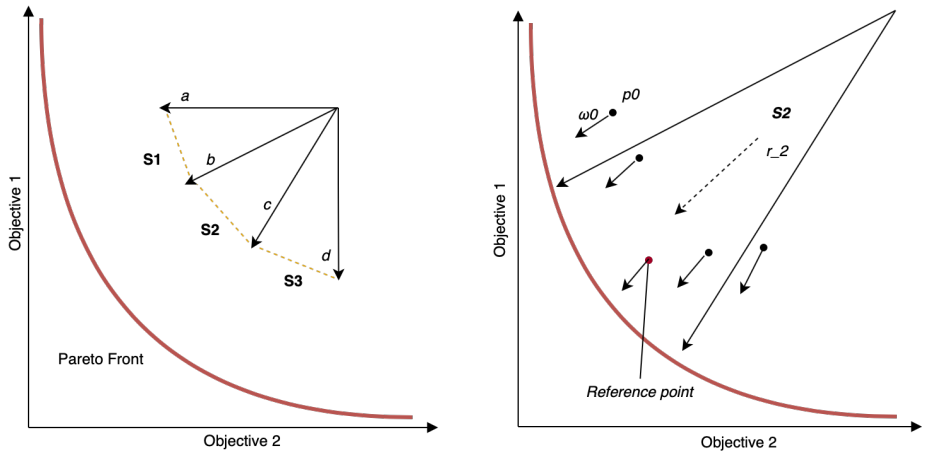


Figure 3.1: Components of the model.

of the objectives that solely focus on a single objective and ignores the other i.e. ⟨0,1⟩ or ⟨1,0⟩. These two vectors are always represented, independent of the amount of swarms used.



(a) The problem decomposed to three spans.

(b) The weight vectors of particles defined by a span.

Figure 3.2: The use of reference spans in relation to decomposition and weight vectors.

Each swarm control both a defined number of neutral and quantum particles. Figure 3.2b shows an initiated swarm where each point such as *p0* represent a neutral particle and the arrows represent weight vectors. Each neutral particle is

connected to a specific weight vector. For instance the weight vector $\omega 0$ marks the direction that the particle $p0$ deems as optimal towards the Pareto front. In the figure span **S2** represents the upper and lower angle for the weight vectors. The angle each weight vector has towards the Pareto front is defined by the reference span when the swarm is initiated.

In figure 3.2b the reference vector is marked as *r_2*. The reference vector is the middle value of the prioritization as defined by the span **S2**. Using the reference vector a reference point is calculated using the Tchebycheff method explained in section 2.1.4.

Quantum particles searches randomly around the position of the reference point. This method replaces the use of the global best that is used in the original mQSO as explained in section 2.2.1. The use of the reference spans for neutral particles and reference position for quantum particles makes all particles in the swarm have a similar prioritization of the objectives. A similar prioritization of objectives will guide particles towards the same part of the Pareto front.

As explained in section 2.1.2 the velocity of particles directly affect the particle position in the search space. This will in theory make each particle traverse to the desired part of objective space following the direction of the weight vector. However, such behavior is not guaranteed.

In figure 3.2b each particle of the swarm has a weight vector aiming at the middle of the Pareto front, but one particle $p0$ is exploring the top of the Pareto front. In an unknown number of iterations $p0$ will likely return to the middle of the Pareto front as the weight vector $\omega 0$ and the span prioritize the middle position of the Pareto front.

The exploration seen from particles such as $p0$ introduces positions which are outliers in terms of the span prioritization. An archive is kept inside each swarm to store all Pareto optimal solutions found throughout all iterations which includes the outliers. When the algorithm has completed the outliers may prove beneficial when aggregating the archives from the separate swarms. Otherwise the outliers are ignored as another swarm has found solutions that dominates the outliers.

The swarms update each neutral particles velocity, position and fitness in every iteration. The fitness function is problem specific e.g. the number of vehicles or driving distance. The swarms also maintain particle communication, such as creating neighborhoods. All particles are made aware of the most similar neutral particles in terms of their weight vectors. A defined number of the most similar neutral particles is used to form a neighborhood, further explained in section 2.2.2. In addition the swarm maintains Tchebycheff specific variables such as the $Z*$ archive of best found values in each objective. The final role of a swarm is to maintain the reference point.

Neutral particles need two variables from the swarm; the weight vectors and

the neighborhood. With each velocity update the reference point is received and the particle will perform a leader selection operation. The leader selection will select any of the neighbors, the reference point or its personal best at random with equal probability. In the fitness update the weight vector combined with $Z*$ will determine the particles personal best position, as seen in Tchebycheff's method. The fitness update creates a solution to the VRP through decoding a particles position which is explained in section 3.1.2.

Quantum particles are affected by a single constant called *radius* which has the same value for every quantum particle in all of the swarms. The role of quantum particles is to perform random searches around a given position. This *radius* constant defines how many random mutations that should be performed around the positions. The new found position is then decoded to a solution.

In each iteration the swarms generate solutions from both neutral and quantum particles. This causes the total number of solutions from each iteration to be *(number of neutral + number of quantum) * number of swarms.*

3.1.2 Algorithm Flow

An illustration of the algorithm flow is shown in figure 3.3, where the following sub sections explain the different parts of the algorithm in detail.

Initialization

The initialization starts by generating a set of reference spans. A reference span is generated for every swarm, where the span itself is a range of possible values for each objective. Each swarm then generates the reference vector and a weight vector for each neutral particle. The swarm forms neighborhoods for each neutral particle that is defined by the similarity of the weight vectors. Each neutral particle is then placed in to a discrete search space at random positions.

Velocity and Position update

For each iteration all swarms receive a request to update the velocity of its contained neutral particles. The velocity update is performed using the position retrieved from the leader selection. When the velocity update has been performed a request is sent to all swarms to update the particle positions. The new position is found using the calculated velocity and the current position which forms a Hamiltonian path.

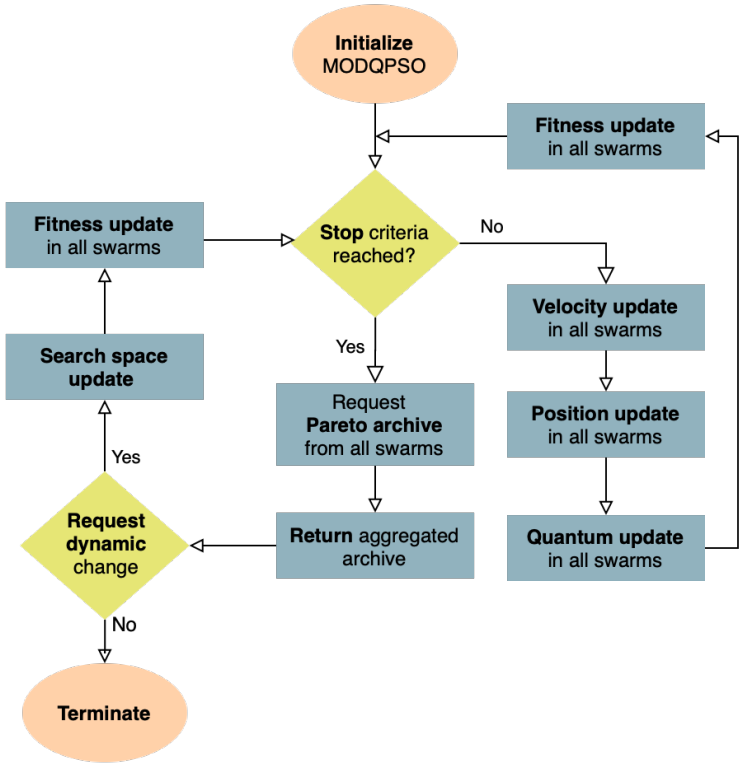
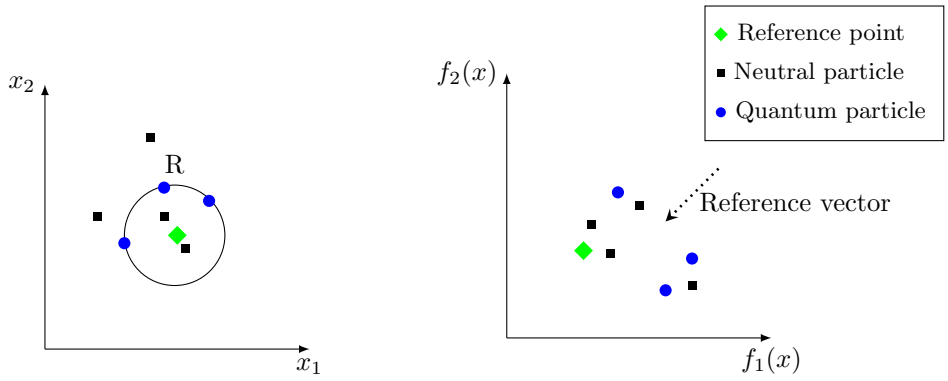


Figure 3.3: The flow of the MODQPSO algorithm.

Quantum update

The quantum update in figure 3.3 is an element easier to understand through a detailed figure. Figure 3.4 shows the relation between the search space and objective space. The figure shows the particles from a single swarm, where none of the particles are currently positioned on the reference point. The reference point is the best found position from all the previous iterations and particles may move away from it from one iteration to another.

The quantum update makes the quantum particles find a random position centered around the reference point. In figure 3.4a quantum particles have been placed randomly on a circle R that is centered on the reference point. Figure 3.4b shows that the reference point remains the best position in relation to the reference vector, where in this scenario the reference vector prioritizes both objective 1 and 2 equally. An equal prioritization of the objectives make the position closest to the graphs origin the superior position. Since no better position was found in figure 3.4 the quantum update for the next iteration will center around the same position. It is first when a superior position is found that the quantum particles will center around a new position. In the first iteration no reference point has been calculated making the quantum particles do a random search unconstrained by a reference point.



(a) Search space. (b) Objective space.

Figure 3.4: A swarms position in both the search space and objective space.

Fitness update

A fitness update is both performed after the velocity, position and quantum updates, but also after a dynamic change. Its purpose is to update the fitness

(the objective values) for all particles. Figure 3.3 does not include a detailed view of the fitness update, but the the update is a complex task. The fitness update is performed in each swarm where all particles decode their position to generate a solution to the VRP where all customers are linked to a vehicle in a specific order.

As a part of the fitness update all generated solutions are kept to be processed at a later stage, but before that the neutral particles update their decomposed fitness. The Tchebycheff method use both $Z*$ and the weight vector to calculate the decomposed fitness of a particle. If this value is lower than the best found decomposed fitness the current position is set as the personal best. In the case of an equal decomposed fitness to the previous found best, the actual sum of objective values is used to choose which position is the new personal best. The reference point is also updated using the same logic. This is possible since the swarm contains a reference vector which can be used as a weight vector in the Tchebycheff method.

As a last step of the fitness update all solutions found by the different swarms are compared. This is done using the Pareto optimal archive inside of each swarm. The archive is updated after all particles have completed their fitness update. At the same time the $Z*$ archive is updated to store any new improved objective values.

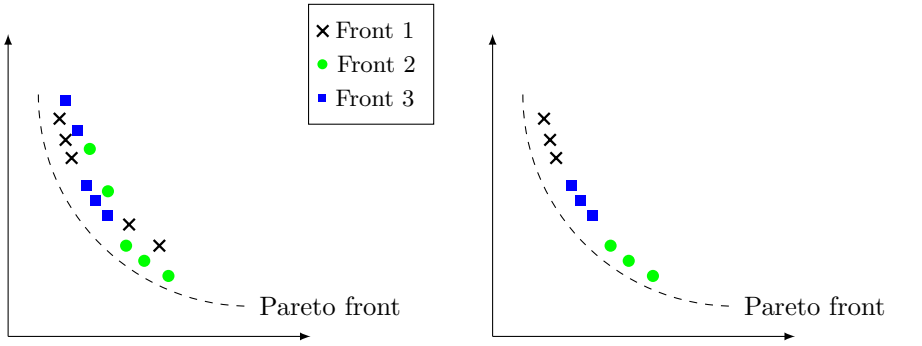
Return aggregated archive

When the stop criteria is reached the swarms return their archives of non-dominated solutions. In figure 3.5a there are three fronts found by three separate swarms. Some of the found solutions are dominated by solutions within other swarms e.g. parts of *Front 3* are dominated by solutions in *Front 1.* After aggregation the resulting front can be observed in figure 3.5b where only non-dominated solutions remain from each of the fronts. The Pareto front seen in the figure is stored in an archive which is used as the return value of the algorithm.

Search space update

After aggregation and return of the found Pareto front, a dynamic change may occur. A dynamic change will require the model to perform a search space update which adds new customers to the VRP. This does not change the order of the Hamiltonian path that is already generated. The new customers are added in a random sequence to the end of the existing path.

As seen in figure 3.3 a fitness update is performed after a search space update since the change may have impacted the objective values. The model will then be able to search for new optimums presented by the dynamic change.



(a) Fronts found by three swarms. (b) Result from aggregating the fronts.

Figure 3.5: Aggregating fronts to a main archive.

3.2 Design decisions

3.2.1 Particles

The model uses the two particle types quantum and neutral, where the distinction between them are derived from mQSO. The logic seen in the leader selection, velocity update, position update and fitness update is composed of a mix of different sources.

Each particle traverses a search space which is formed by the representation. The model allows this representation to take different forms. However, every representation must in some way represent a Hamiltonian path where all customers of the VRP is included in a specific order. The method of representing the Hamiltonian path may vary such as the method used in either S-PSO or I-PSO. In the model both the representations of S-PSO and I-PSO are implemented, both further discussed in section 2.2.2.

Leader selection

The leader selection is invoked from the velocity update. All particles involved in the leader selection have an equal probability of being selected. This function is inspired from both MODPSO, explained in section 2.2.3 and the leader selection of S-PSO. The leader selection of S-PSO is identical to the one in MODPSO if the neighborhood size in MODPSO is set to the *number of particles - 1*. The model allows for any number of particles contained in the swarm to be a part of the neighborhood where any neighbor, the personal best position of the particle or the reference point can be selected as a leader.

The addition of the reference point is included so that quantum particles are able to affect the velocity and position of neutral particles, as quantum particle are not a part of any neighborhoods. Quantum particles have to be able to affect the velocity of neutral particles in the case of a dynamic change that are discovered by the quantum particles. The neutral particles may have converged, but quantum particles can still discover new optimums and guide neutral particles towards new optimums introduced by the change which is further discussed in section 2.2.1.

The model utilizes a random leader selection as it requires less computations and adds less complexity. A random leader selection is found to have the same performance as a selection strategy for most problem types which is further discussed in section 2.2.3.

Velocity and Position update

The velocity structure is based on S-PSO. The velocity does not represent direct velocity in each dimension, but probabilities linked to every action. An action is defined as a route from one specific customer to another in the VRP. Further discussion of the functions of S-PSO is discussed in section 2.2.2.

$$v_i(t+1) = \omega v_i(t) + c_1 ran_1[\text{leaderSelection}(x_n, p_i, \text{refPoint}) - x_i(t)] \quad (3.1)$$

In MODQPSO the velocity update function includes the selection of the reference point as leader, seen in equation (3.1). The velocity is only affected by the particles previous velocity and the selected leader. The leader is found using the leader selection as explained in the previous section.

The position update is dependent on the representation. For both the S-PSO and I-PSO representations the velocity is used to modify the current position. In the case of S-PSO the position update is not modified from implemented as in the original S-PSO algorithm. However, it should be noted that even though the position update of I-PSO is not modified some modification is performed on I-PSO. As explained in section 2.2.2 I-PSO does not represents a Hamiltonian path, but each position of I-PSO includes stops at the depot which means the change of a vehicle. In MODQPSO when referring to the I-PSO representation the use of multiple vehicles are ignored and a single Hamiltonian path is what is stored at each position for the particles. This means that both S-PSO and I-PSO positions has to be decoded to represent a solution to the VRP.

Quantum update

The quantum particles used in the model are inspired by those in mQSO. However, the mQSO quantum particles are designed to operate in a continuous search

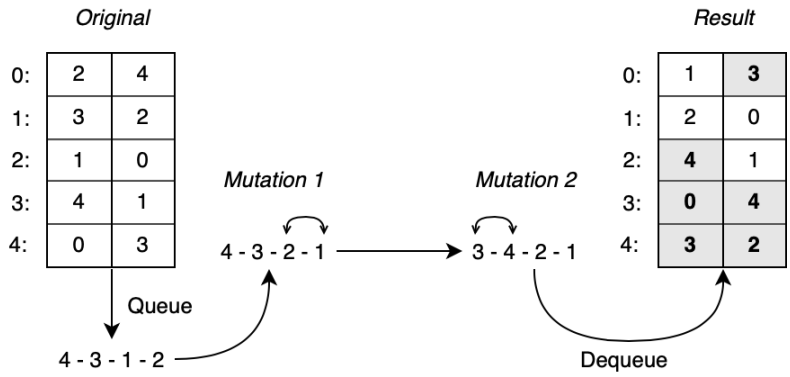


Figure 3.6: Quantum update

space. The mQSO quantum particles are placed at a set distance away from the global optimum using a continuous ball function to place them in the search space, further discussed in 2.2.1. As the representations of both I-PSO and S-PSO are discrete the placement of quantum particles must be calculated using a discrete interval. The discrete interval, also called the radius, is equivalent to the number of random mutations.

These mutations consist of switching two and two values to simulate the random distance which quantum particles are placed away from a position in mQSO. The process of switching a series of two and two values are denoted as performing a quantum update. At each iteration a swarm will send the reference point to all of its quantum particles, where the reference point serves as the center of the quantum update.

Figure 2.5 performs a quantum update with a radius of 2 on a given position. The details of the S-PSO representation are explained further in section 2.2.2. The *Original* position in figure 2.5 is first transformed to a queue before performing the mutations. In the figure *Mutation 1* and *Mutation 2* both switches the position of two customers on the Hamiltonian path. The queue is then transformed back to a S-PSO position marked as *Result*. If this switch was to be performed directly on a S-PSO position rather than the proposed queue, a domino effect would occur where the values for both the previous and next customer had to be updated on each of the switched positions. In addition the values of the next and previous customers after and before the switched customers on the Hamilton path had to be updated. Hence it is less complex to transform the position to a queue, as only the values that are switched are affected as seen with both *Mutation 1* and *Mutation 2*. When using the I-PSO representation the

step of creating a queue is skipped as all positions of I-PSO represents a queue of customers.

In MODQPSO quantum particles have two main functions. The particles introduce diversity, where they replace the need for other techniques such as turbulence seen in MODPSO. Where turbulence is a method which preforms similar mutations on positions, but on random neutral particles further explained in section 2.2.2. In addition, and as previous stated, quantum particles may guide neutral particles towards new optimums introduced by dynamic changes.

Fitness update

The fitness update start with the initiation of the decoding process. As mentioned the position is stored in each particle as a Hamiltonian path. However, solutions may utilize more than one vehicle. To be able to use multiple vehicles parts of the Hamiltonian path are separated into different routes, where each route is linked to a specific vehicle.

As in S-PSO, all particle positions in MODQPSO are decoded. However, S-PSO initiates a thorough use of heuristics to optimize the found solutions after the search is completed. The MODQPSO model does not end with such heuristics, but instead perform a heuristic as a part of the decoding process. MODQPSO uses a slightly modified version of the heuristic "parallel tour construction algorithm" [Jurgen Antes and Derigs, 1995]. The heuristic uses an auctioning process that calculates the increase of objective values for each customer to each vehicle, denoted as the cost. The heuristic will then place the customer to the vehicle route that has the smallest cost, further explained in section 2.1.1. The modification of the heuristic when implemented in MODQPSO is to constrain the available customers used in the auctioning process. The heuristic is not able to choose from all customers when inserting customers to vehicles, but is restricted to place the customer consecutively from a queue of customers.

The heuristic as implemented in MODQPSO is illustrated in figure 3.7. In the figure the heuristic has already inserted three of the six customers. Customer *c_8* and *c_2* is inserted to vehicle *v_1* and customer *c_5* to *v_2*. Customer *c_4* is the next in the queue and is used as input for the auctioning process. When a new customer is evaluated all available vehicles will return the minimum cost of inserting the customer and the index of where it should be placed within the given vehicle. In the figure all vehicles has returned a cost where vehicle *v_2* has returned the lowest, and recommends it to be inserted at index 1. Consequently, the next step of the heuristic will be for customer *c_4* to be inserted to vehicle *v_2*. As vehicle *v_2* already contained another customer *c_5*, the updated vehicle route will be *c_5* followed by *c_4*.

When using a queue of customers as input, MODQPSO is able to produce a population diversity by having different sequences of customers in the input

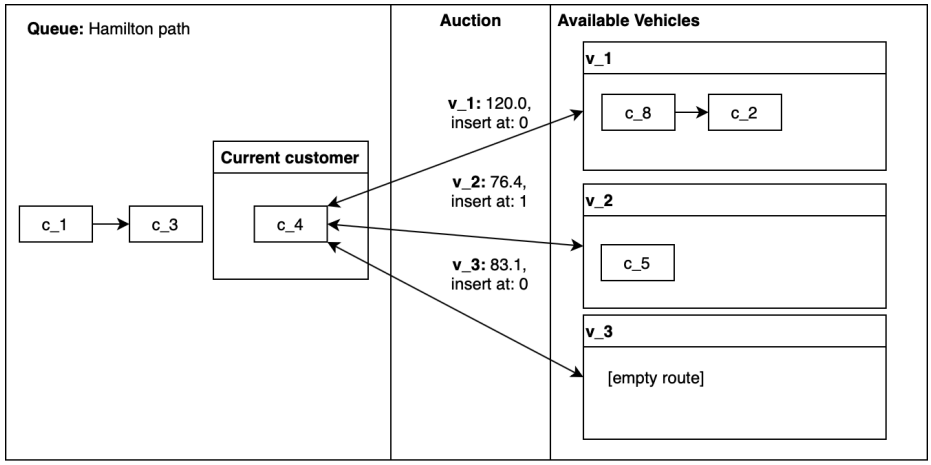


Figure 3.7: Decoding Hamiltonian path with heuristic.

queue for the heuristic as particles finds positions which correlates to different sequences of customers. The heuristic will produce different solutions based on the input queue, as the placement of one customer restricts the possibilities of the next customer in the queue.

The constraint handling of MODQPSO is implemented on top of the auctioning process. Before calculating the cost of inserting a customer into a vehicle route, it is first calculated if the vehicle is able to insert the customer in terms of the problem specific constraints. Vehicles where the customer may be inserted are referred to as an available vehicle in the auction process. Making vehicles that are unable to satisfy constraints unavailable removes the possibility of generating infeasible solutions. In the case of infeasible solutions where all vehicles are unavailable or one or more customers have not been inserted, the particle is immediately reinitialized to a random position.

The cost used in the auctioning process is originally the increase of a objective in a single objective problem. Since MODQSPO operates on multi dimensional problems the increase of each objective are combined to a single cost value. This process is performed in figure 3.8 where the cost of inserting vehicle *v_2* from figure 3.7 is calculated. In figure 3.8 the increase of two objectives are calculated and combined using the reference vector from the swarm to calculate a single value cost. The cost of insertion is calculated for all possible index positions of insertion. Vehicle *v_2* already contained a customer *c_5* which means there are two possible index positions of insertion i.e. before and after *c_5*.

Using the reference vector inside the heuristic lets the auctioning prioritize

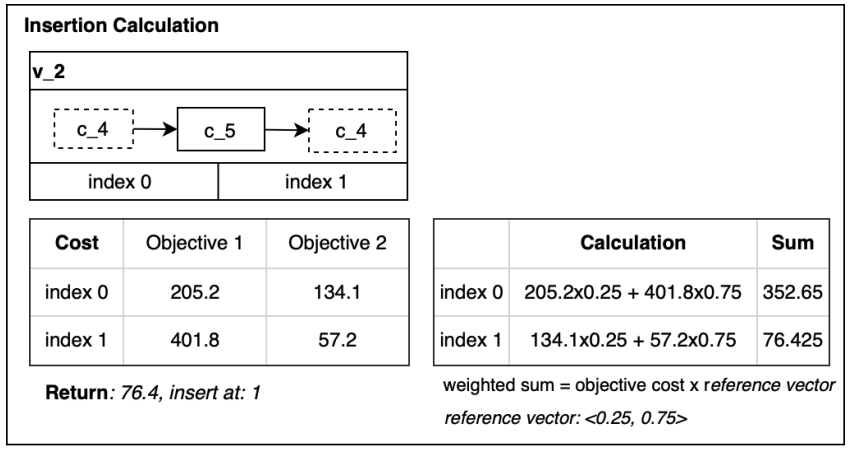


Figure 3.8: Calculating heuristic insertion cost.

the objectives like that of the swarm itself. If the reference span is set to favour objective 2 more than objective 1 the heuristic will also do so. This is achieved by using the reference vector to prioritize the objectives when calculating the insertion cost, as the reference vector is the middle value of the reference span. This decision makes the search space of each swarm correlate to different areas of the objective space, since the decoding from search space positions to solutions are based on variables that are dependent of the swarm. This is not an issue as there is no communication between the swarms, and the correlation between the search space and objective space is the same for all particles within the same swarm.

When the heuristic is completed all customers will have been placed inside a vehicle route, where the set of these vehicle routes are referred to as a solution. The objective fitness of each solution have also been calculated along with the insertion of customers.

3.2.2 Swarm structure

Multiple swarms are utilized in MODQPSO as a dynamic handling technique as seen in mQSO. However, MODQPSO does further exploit the multiple swarms as it decompose the multi-objective problem into different swarms. This is done by sending a reference span to each swarm, so that each swarm can focus their search on a specific part of the Pareto front. The reference spans make the swarms search for solutions that are favorable of the spans prioritization of the objectives. This may still return a front of non-dominated solutions. Decomposition is therefore

used once more to make each particle of the swarms have a specific prioritization of the objectives i.e. have a defined weight vector. Each particle of a swarm will have a similar weight vector, as all particles within a swarm has their weight vector set within the boundaries defined by the reference span. Weight vectors allows particles to choose a single position as their personal best. Similar weight vectors within the same swarm will guide the particles to search for solutions in the same area of the Pareto front.

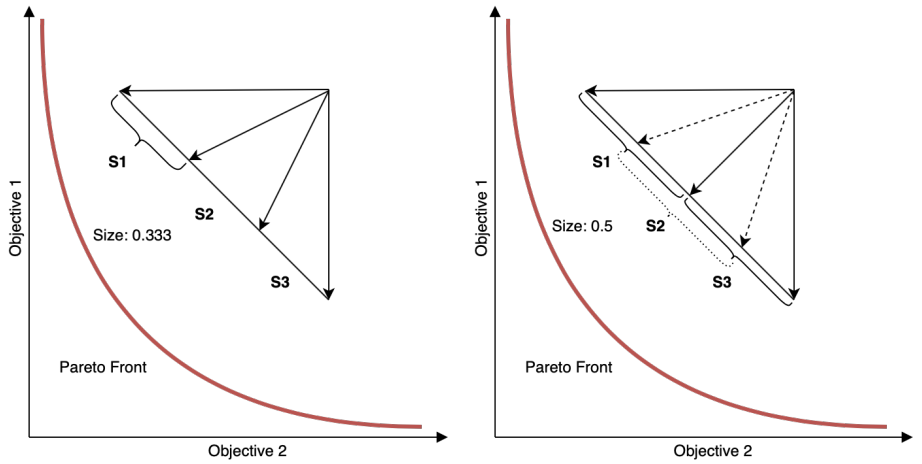
The decomposition technique of MODQPSO is inspired by other decomposition based multiple objective techniques, such as MODPSO. However, MODQPSO uses decomposition on two levels as both particles and swarms are affected by it. This concept is proposed to increase the efficiency of quantum particles with multiple objectives. The increase of efficiency is theorised based on the assumptions of hyper-cubes, explained in section 2.1.4. Algorithms based on hyper-cubes such as hyper volume based approaches explained in section 2.2.3, all depends on the concept of geographically-based diversity. This concept means that the search space and objective space are coupled. All Pareto optimal solutions are not centred around the same position of the search space, but instead spread out. Still, positions that are close in the objective space are also close in the search space. When creating the reference spans MODQPSO guides each swarm to a specific part of the Pareto front as it also assumes geographically-based diversity. As particles searches for a specific part of the Pareto front they will benefit from each other in communicating their personal best position. Since all particles of a swarm has similar weight vectors the optimal positions for the particles will be located in a "similar" position in the search space.

If all swarms were to search the entire Pareto front, the findings of the quantum particles may only be beneficial for a small number of neutral particles contained in a swarm. All particles within a swarm has similar weight vectors and the reference vector is the middle value of those weight vectors. The reference vector is then used to find the reference point. Quantum particles will search randomly around the reference point which may prove to be beneficial to many of the particles because of the concepts of geographically-based diversity, as the reference point will be located in proximity to the optimums of all the neutral particles of the same swarm.

Reference span size

The default configuration for the reference spans are to allocate an equal size per span so that there are no overlap between the reference spans. It was decided to make this the default configuration as it was desired that each swarm focused on a completely separate part of the Pareto front to be able to generate as much diversity as possible.

However, theory from MODPSO surrounding neighborhoods, further explained



(a) Three swarms with default reference span size.

(b) Three swarms with the reference span size of 0.5.

Figure 3.9: Visualization of two reference span sizes.

in section 2.2.2 conclude that it is important for particles to communicate with particles with similar, but different weight vectors, as this adds diversity to the swarm. This may mean that the default setting for reference spans may make the spans too small when the number of swarms are increased. Hence the model allow for the reference span size to be tuned.

In figure 3.9a the reference spans at a default configuration are shown. In figure 3.9b an example of overlapping spans are shown. These figures highlight that an increased size from the default value will cause an overlap between the spans.

Chapter 4

Experiments and Results

The chapter presents the simulator used for the experiments in section 4.1, followed by an explanation of the data sets in section 4.2. The preliminary tests and preliminary results are presented in section 4.3. Further, the experimental plan is presented in section 4.4, while the experimental setup is presented in section 4.5. Finally, the experimental results are presented in section 4.6.

4.1 Simulator

A simulator is implemented to run the MODQPSO model and illustrate the produced solutions. The simulator is designed for VRP problems and is able of handling different data sets. For the thesis the simulator is set up to handle a home care data set provided by Visma and the Solomon VRP data set. The visualisation of the solutions display both the path each vehicle traverses and the time schedule each vehicle has.

In square 1 of figure 4.1 the depot is drawn as a red circle and customers as black squares. The vehicle routes are drawn as lines with an unique color for each vehicle. The drawing of the routes allows for a graphical view of the achieved fitness. The simulator is designed for multiple objectives, hence a Pareto front is displayed as a graph in square 2 of the figure. In the graph each found solution is drawn as a point with an arch connecting it to the closest point. Also in square 2 a next and a previous button allows the user to browse through the Pareto front and choose which solution to visualize. In the figure the solution furthest to the right in the Pareto front is selected which can be observed by the orange circle.

From square 3 the algorithm can be initiated. The Run button initiates a single run of the algorithm, while Calculate SD runs the algorithm a set number of times and presents meta information such as standard deviation, run time and

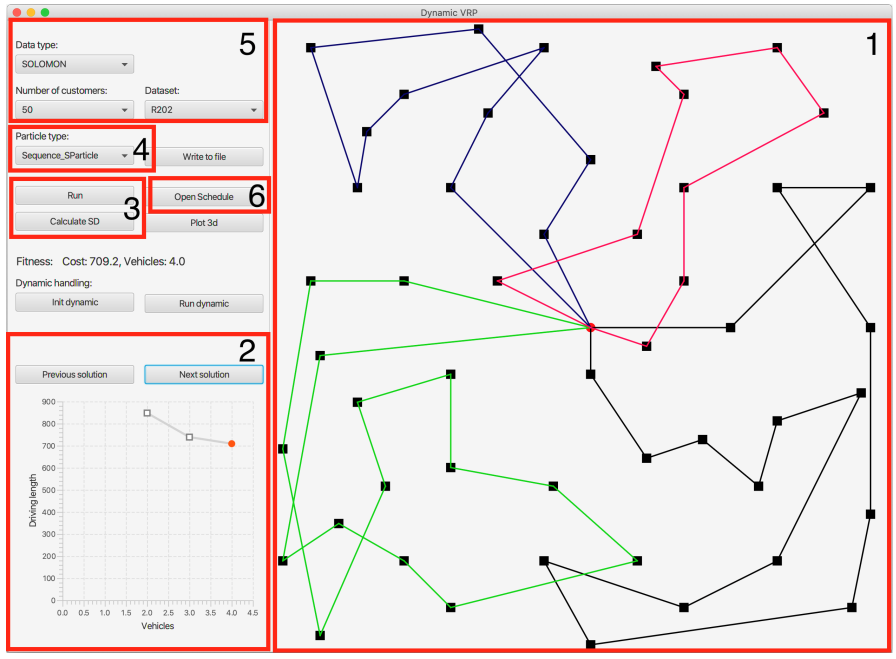


Figure 4.1: The graphical user interface of the simulator.

a complete log of the achieved fitness from each of the runs.

Square 4 contains the option to change the selected algorithm between S-PSO or MODQPSO with a specific discrete representation. From the dropdown menus in square 5 the user can choose the desired data set, amount of customers for the problem and the specific problem.

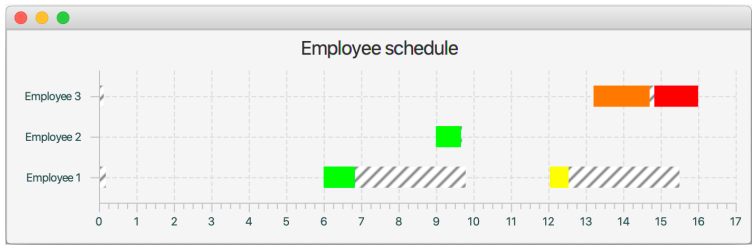


Figure 4.2: Graphical view of the time windows of the vehicles.

In square 6 there is button that opens a new window that shows the schedule assigned to each employee, as seen in figure 4.2. Green squares symbolizes that the time window of the job is not breached, while yellow and orange means that the time windows are breached with less than 2 and 4 hours respectively. Time windows that are breached with more than 4 hours are symbolized by red squares. The striped lines symbolize that the employee must use time to drive from one location to another.

4.1.1 Parameters

The parameters of the model are presented and explained in table 4.1.

Iterations	The number of iterations the algorithm is run.
Swarms	The number of swarms.
Quantum particles	The number of quantum particles used in each swarm.
Neutral particles	The number of neutral particles used in each swarm.
Radius	The size of the radius a search is done around the best position of a swarm.
Neighborhood size	A neighborhood is formed with neutral particles based on the similarity of weight vectors. The number of particles that is included in the leader selection.
Inertia	The slowdown factor from the traditional PSO.
C1	Importance of the selected leader in the velocity update.
Reference span size	The size of the reference spans used by the swarms.

Table 4.1: Parameters for the MODQPSO algorithm

4.2 Data set

4.2.1 Solomon

The Solomon data set, retrieved from Sintef [2008], contains a variation of test problems that are widely used as a VRP benchmark. It contains 56 test problems for 25, 50 and 100 customers. The test problems are made of six different types (C1, C2, R1, R2, RC1, RC2). In the C sets the customers are clustered, while in the R sets the customers are spread uniformly. The RC sets are a combination of randomly placed and clustered customers. In the thesis time windows are defined

as wide, narrow or a mix of the former two. The Solomon sets of type 1 (C1, R1, RC1) contains narrow or mixed time windows and a low vehicle capacity, only allowing a few customers per route. Sets of type 2 have wide time windows and a high vehicle capacity allowing more customers to be serviced by the same vehicle.

Each problem contain constraints for vehicles, customers and the depot. Regarding vehicles there are limitations on the maximum amount of vehicles that can be used and the maximum load they each can carry. Customers have coordinates, a demand of load capacity, a time window and a service duration they require. The depot have coordinates and a time window that all customers have to be serviced within. Time windows in the Solomon data set are handled as a hard constraint. The distance between customers and depot are calculated using the Euclidean metric, where travel time is equal to the Euclidean distances.

The Solomon data set can be both single and multi-objective, where the number of objectives must be chosen before using the data set. Each problem is solved and ranked based on the total driving distance and optionally the number of vehicles used. Both available objectives are to be minimized.

Problems from the Solomon data set will be presented in a shorthand version e.g. R111.50. The *R* is the problem type where the customers are spread uniformly, *111* is the problem identifier and *50* is the amount of customers in the problem.

The Solomon data set is not a dynamic data set, nor does it provide a widely known way of converting it to one. However, a method seen in Necula et al. [2017] is used where a percentage of the customers are removed from the problem at initialization. These customers are introduced at a later iteration to create the effect of a dynamic change. There are no official benchmarks for a dynamic version of Solomon.

The benchmarks of the Solomon problems are gathered from the Solomon website [Sintef, 2008]. A series of publications contains benchmark solutions to Solomon problems such as R111.50 and will be used throughout the experiment chapter. The specific publications are from the following authors: Kohl et al. [1999] denoted as KDMSS, Kallehauge et al. [2001] denoted as KLM and Nazif and Lee [2010] denoted as OCGA.

4.2.2 Visma

Visma has provided anonymous data from a real-world home care scheduling problem. The data set consist of seven problems, which correlates with the consecutive days of a single week. In this data set vehicles are denoted as employees and customers as patients.

Each employee have an unique identifier and a time window for there work hours, meaning a employee can only perform tasks within its work hours. Each patient have an unique identifier, a longitude, a latitude and the travel time to every other patient. In home care a patient can be visited more than once per day. Instead of connecting vehicles to customers for the Visma data set there are used tasks instead. A task is connected to a customer, allowing a customer to have several tasks per day. Each task have a patient id, a time window and a service duration. It is desired that each task is to be started within the given time window (soft constraint) and must be served for the entire service duration (hard constraint). Lastly, the depot have a longitude, a latitude and the travel times to each of the patients.

$$\begin{cases} f_1 = \sum_{e=0}^{X} \sum_{c=0}^{Y_e-1} D(P(c,e), P(c+1,e)), & \text{Travel distance} \\ f_2 = \sum_{e=0}^{X} \sum_{c=0}^{Y_e} T(e, t(c)), & \text{Deviation from time windows} \\ f_3 = \sum_{e=0}^{X} max(W(Y)) - W(Y_e), & \text{Equal work load} \end{cases} \tag{4.1}$$

The seven problems vary in employees, patients and tasks. Following, the different data sets have employees with varying time windows, location of patients and tasks with varying duration and time windows. The problems are solved and ranked based on the three separate objectives functions described in equation (4.1). Hence, each solution is dependent on the total driving distance in objective function f_1, the total deviation time from all time windows in objective function f_2 and the rate of task distribution between the employees in objective function f_3. Objective function f_1 is also used in the Solomon data set when discussing "total driving distance". The abbreviations and functions used in declaring the objective functions are explained in table 4.2. In the thesis all objective functions are to be minimized.

The Visma data set is a dynamic data set which operates similar to that of dynamic Solomon. Around 5% of the total tasks of each problem are dynamic, which means that they are introduced at a certain iteration where a dynamic change occur.

Term	Description
X	The number of vehicles.
e	A specific vehicle.
Y	A set of all routes.
Y_e	The set of customers for vehicle e i.e. a specific route.
c	A specific customer.
p_c	The position of a customer c.
$D(p_c, p_{c+1})$	Returns the travel time between a customer c and another.
$t(c)$	Returns the time window of customer c. The time window is defined by two values, a start and end time.
$T(e, t_c)$	Returns the deviation between the arrival time of e at c and the time window of c. Returns 0 if vehicle e is inside the time window of c.
$W(Y_e)$	The number of tasks in a specific route.

Table 4.2: Visma abbreviations and functions.

4.3 Preliminary tests

This section establish the experiments conducted while developing the proposed algorithm. The tests are performed to ensure that the components of the model benefit the algorithm and finding the optimal parameters for the components. The tests have been performed on the Solomon data set. All parameters except the reference span size is explored in the preliminary tests. In the preliminary tests the reference span size is set to the default configuration of 1/number of swarms.

The preliminary test phase use the single benchmark problem R111.50. This particular problem was chosen as a Pareto front of several solutions can be found. The Pareto front consist of solutions with 6 and 7 vehicles, as there is no driving distance of 6 vehicles that is found to dominate the driving distance of 7 vehicles. The best possible front based on {link solomon?} and S-PSO [Gong et al., 2012] is: {⟨v: 6, dd: 756.3⟩, ⟨v: 7, dd: 707.2⟩}, where v equals vehicles and dd equals driving distance.

The preliminary testing consist of three phases that each does a parameter sweep. In each phase a set of parameters is defined that attempts to bring the best parameter values from the previous phase. Each parameter value is tested through 30 runs. The results present the standard deviation, the average driving distance and the average run time in milliseconds. When running the tests the model returns a flag marking the number of failures where the Pareto front only include solutions with 8 or more vehicles. In addition it has a separate flag for the number of Pareto fronts that includes solutions with 6 vehicles.

All tests are ran on a Macbook Pro 2018 with a i7 6-core 2.2 GHz processor that boosts up to 4.1 GHz and has 16GB of RAM.

4.3.1 Phase one: Sweep around parameters from literature

The first phase has the parameters set to the recommended parameter values from literature, as seen in table 4.3.

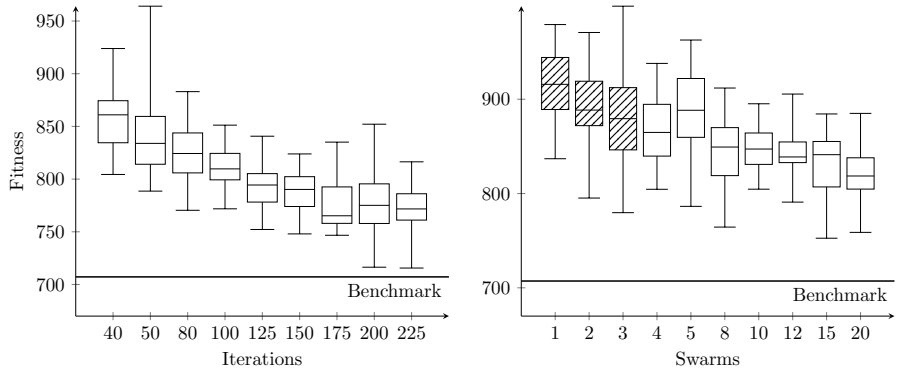
The first phase has the parameters set to the recommended parameter values for the algorithms used as inspiration for the model, as seen in table 4.3. The MODPQSO algorithm does both combinations and changes to those algorithms which can impact whether these parameters are optimal. Hence, the first phase makes a thorough search centered around the parameter values from literature as a starting point.

Parameter	**Value**
Iterations	50
Swarms	10
Quantum particles	5
Neutral particles	5
Radius	3
Neighborhood size	3
Inertia	0.3
C1	2

Table 4.3: Phase one algorithm parameters

After testing different values for each parameter from the literature some trends were observed. The model benefit by an increase in the number of iterations as seen in figure 4.3a. The parameter value for iterations was set to 150 for phase two to give the model more time to converge, but still keep the computational cost and run time at a reasonable level.

The number of swarms does affect both the performance and efficiency of the algorithm. Increasing the number of swarms rapidly increases the computational cost required, but in return increase the likelihood of finding fit solutions as seen in figure 4.3b. The figure illustrates the improved fitness related to the number of swarms, where the boxes tend to become smaller indicating more stable results as the number of swarms increase. The testes revealed that when using between 1 and 3 swarms the algorithm was unable of consistently retrieving solutions that used 7 vehicles. In the figure the parameters that failed doing so are marked with dotted lines. The swarm behavior was improved greatly by using a higher amount of swarms. The number of swarms was set to 15 for phase two.



(a) Iterations (b) Swarms

Figure 4.3: Selected results from preliminary phase one testing.

All of the remaining parameters were also tweaked from the standard value. Parameters such as quantum had a tendency of affecting the fitness, but did not show the same clear tendencies as observed with both iterations and swarms.

4.3.2 Phase two: Improve performance

A new set of parameters were defined for phase two based on observations made in phase one. All parameters except C1 has been updated with new values from the previous phase. The resulting parameters can be seen in table 4.4. The sweep around the new parameters were narrowed down, as some values tested in phase one were regarded as unsuitable for the model e.g. not testing less than eight swarms. The main focus in this phase was to improve fitness, with little regard to efficiency in terms of computational cost.

Parameter	**Previous value**	**Value**
Iterations	50	150
Swarms	10	15
Quantum particles	5	7
Neutral particles	5	7
Radius	3	2
Neighborhood size	3	4
Inertia	0.3	0.25
C1	2	2

Table 4.4: Phase two algorithm parameters.

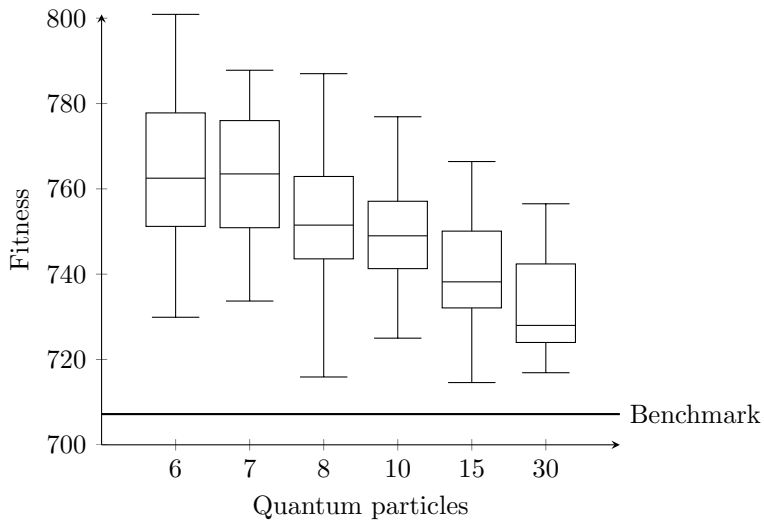


Figure 4.4: Results from testing the number of quantum particles in phase two.

In phase 2 increasing the number of quantum particles was observed to improve fitness. In figure 4.4 it can be observed how the median of found driving distance is lowered from 762.5 with 6 quantum particles per swarm to 727.99 with 30 quantum particles. In addition, 15 and 30 quantum particles was the first preliminary tests that where able of finding Pareto fronts that contained both 6 and 7 vehicles.

The number of swarms, which had a great impact in phase one, made less of a difference when testing a higher amount such as 25 swarms. This may correlate to the model structure, where a great of number of swarms will cause each swarm to be more similar to a single weighted objective algorithm. With many swarms the objective span will be set to an insignificant range, where the possibilities for different objective prioritization is indistinguishable to single objective prioritization. This could make the algorithm have problems escaping local optimums. This seems to be related to theory from MODPSO which tries to solve the issue by creating neighborhoods made up of particles with similar, but not equal weight vectors. Communication between particles of different weight vectors increases swarm diversity further discussed in section 2.2.2.

Neutral particles did not present interesting results when increased, both in phase one and two. However using less than 5 showed a significant worsening of fitness, indicating that both quantum particles and the model itself need some neutral particles to guide the search.

4.3.3 Phase three: Improve efficiency

Phase three focused on decreasing the computational cost from phase two, while attempting to maintain the improved fitness. A new set of parameters are defined as seen in table 4.5. The changed values are swarms, quantum and neutral particles, neighborhood size, inertia and C1. The sweep around the new parameters were again narrowed down, where some new parameter values were also tested.

Parameter	**Previous value**	**Value**
Iterations	150	150
Swarms	15	10
Quantum particles	7	15
Neutral particles	7	6
Radius	2	2
Neighborhood size	4	3
Inertia	0.25	0.3
C1	2	1.5

Table 4.5: Phase three algorithm parameters

Running the algorithm with the new parameters revealed surprising results. The goal of the phase was to lessen computational cost at the cost of fitness, but the results showed both less computations and improved fitness. This may be an indication of the use of too many swarms in phase two, where the reference spans assigned to each swarm are getting too restricted as the reference spans are not allowed to overlap in the preliminary tests. Another factor may be that the increase of quantum particles made the swarms behave a bit different than what was observed in phase one. In figure 4.3b it is observed that increasing the number of swarms always improved the resulting fitness, but this is contradicted in phase three. It may be that an increase in both the diversity and convergence abilities of each independent swarm makes the number of swarms redundant above a specific number.

4.3.4 Parameter selection

To conclude the preliminary tests a final phase is conducted for both evaluation and optimization of the model. Each of the introduced phases improved the fitness which can be observed in figure 4.5. In the figure phase one shows how unsuitable the parameters from the literature was for the model and also that the parameters from phase three are dominant to those found in phase two. The figure also shows how close the model was to find the problems benchmark solution. In separate runs of both candidate 1 and 5 from table 4.6 the algorithm

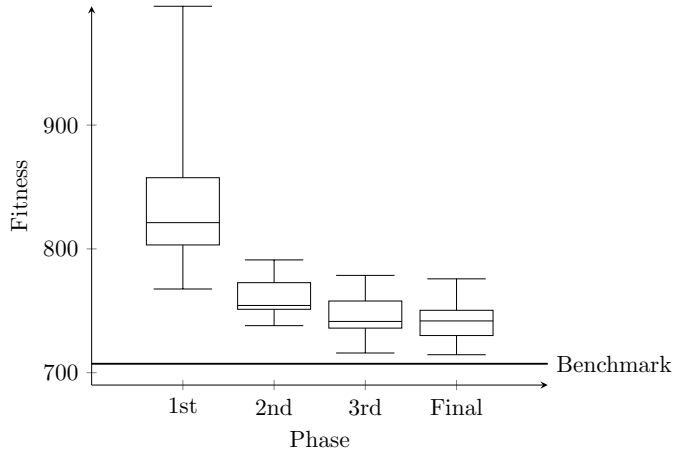


Figure 4.5: Comparing the parameters for all phases.

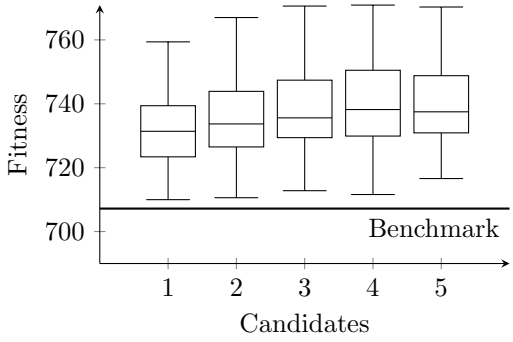
was 0.1 away in driving distance from finding the benchmark driving distance of 707.2. The route of 707.3 in driving distance was found by both candidate 1 and 5 and is shown in figure 4.7b. The difference in driving distance found by both candidate 1 and 5, and the parameters from phase one is visualized in figure 4.7.

	Candidates				
Parameter	**1**	**2**	**3**	**4**	**5**
Iterations	200	200	175	150	150
Swarms	11	11	11	11	11
Quantum particles	20	20	20	20	20
Neutral particles	7	6	6	6	7
Radius	2	2	2	2	2
Neighborhood size	3	3	3	3	3
Inertia	0.4	0.35	0.35	0.35	0.35
C1	1.75	1.5	1.5	1.5	1.5

Table 4.6: Phase three algorithm parameters

The evaluation phase is intended to learn from the tested parameter values used in the three earlier phases to be able to select a set of final parameters to use for the experiments. Five different combinations of parameters was chosen based on observed behavior of each separate parameter, where these five combinations are referred to as candidates. The selected parameter values for each candidate are presented in table 4.6.

The most promising candidate based solely on fitness from the results shown in figure 4.6a is candidate 1. However, candidate 5 decreases the computational cost by approximately 26%, with an average fitness loss going from 736.59 in candidate 1 to 741.8 in candidate 5. Based on the average difference of 5.21 on lost driving distance compared to the 26% decrease in computational cost, candidate 5 was chosen as the model parameters.

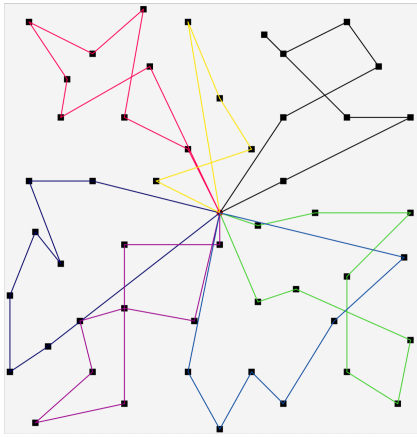


(a) Comparison of candidates from phase three.

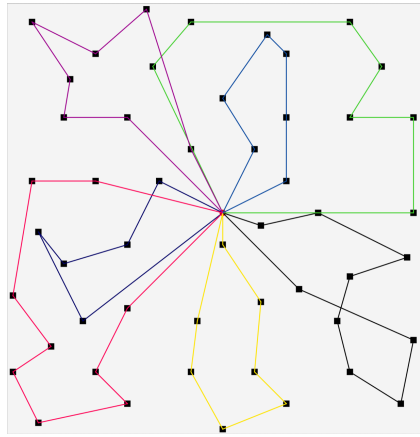
	CPU time
1	3.74s
2	3.57s
3	3.18s
4	2.71s
5	2.79s

(b) Average runtime of candidates.

Figure 4.6: The preliminary parameter candidates and their average runtimes.



(a) Best found solution with parameters from literature, fitness: 782.7.



(b) Best found solution with parameters from preliminary tests, fitness: 707.3.

Figure 4.7: Comparison of travel distance from before and after preliminary testing.

4.4 Experimental Plan

4.4.1 RQ1 - Experimental Plan

The goal of the tests are to explore the possibility of modifying the search space to both improve performance and decreasing computational cost. As explained in section 3.1 the MODQPSO model is implemented such that the search space representation can be changed as long as each particle position is able to produce a Hamiltonian path. The first representation implemented was the S-PSO, explained in section 2.2.2. This representation seemed promising when explained by the original author as many VRP specific issues is solved in the representation itself. In the S-PSO paper Gong et al. [2012] the author referred directly to I-PSO, explained in section 2.2.2. The author claims that I-PSO perform worse than S-PSO due to it being based on the methodology of the original continuous PSO, and that it is not well fit for the discrete VRPTW. This statement sparked an interest in modifying the representation of S-PSO.

The details of how both I-PSO and S-PSO are integrated in to the model are further explained in the model chapter 3. S-PSO and I-PSO uses the velocity update differently, but in both cases the velocity represents probabilities of certain actions. However, what these actions are depends on the use of either S-PSO or I-PSO which is further explained in section 3.2.1. This leads to the possibility of creating hybrids of S-PSO and I-PSO as both the representation of velocity and position may interchange in the configurations of the model.

The following list presents the created representations to be able to perform the tests in RQ1.

Representation 1 The representation as explained in S-PSO with no modifications.

Representation 2 S-PSO velocity and position, but the knowledge of the previous customer is removed.

Representation 3 The position of I-PSO combined with the S-PSO velocity.

Representation 4 I-PSO velocity and position.

To be able to evaluate the representations behavior in convergence abilities, single objective Solomon problems are used where driving distance are to be optimized. In turn, to be able to observe the representations abilities to both cover and converge on a large Pareto front the Visma data set is used. Hence, the following two tests are declared to explore RQ1:

Test 1.1 Test the different representations on the Solomon data set.

Test 1.2 Test the different representations on the Visma data set.

Hypothesis

The hypothesis of representation 2 is that the same performance of representation 1 can be achieved while also decreasing computational cost. Both test **1.1** and test **1.2** will show this behavior. It is also expected that the behavior of representation 1 and 2 correlates with the size of the time windows. Representation 1 will in problems with wide time windows benefit from the original S-PSO representation and have an improved performance compared to representation 2. The opposite is expected for problems that contains smaller time windows, where representation 2 will have an improved performance compared to representation 1. This hypothesis is based on the behavior explained in section 2.2.2 where S-PSO is able to learn from both the next and previous customer on a Hamiltonian path. It is expected that when the time windows are small learning from the previous customers on a path will only cause noise as time windows may not allow for a previous customer to be visited after what was established.

Representation 3 and 4 are expected to decrease performance, compared to representation 1, as they are based on I-PSO. I-PSO is shown to be inferior to S-PSO in the case of VRP, as seen in Gong et al. [2012]. However, the behavior when used in the MODQPSO model may be different and it is expected that representations based on I-PSO will have a decrease of computational cost. This behavior will be seen in both test **1.1** and test **1.2**.

4.4.2 RQ2 - Experimental Plan

The goal of the following tests are to explore different strategies for reference spans in order to improve performance in multi-objective optimization.

Reference spans are proposed in chapter 3 as a method of combining mQSO with decomposition. Each reference span allocates a specific prioritization of the Pareto front to each swarm. The default setting for spans is a size that causes no overlap in objective prioritization between the swarms. However, it is possible to set a size so that an overlap between the spans occur as discussed in section 3.2.2.

From preliminary testing in section 4.3, the number of swarms is set to 11. When using 11 swarms the reference spans may have a size of 1/11 or approximately 0.091 without causing an overlap between the spans. This is used as the default value for reference spans. When using spans with any size between 0.091 and 1 it will cause an overlap between the spans. Using the largest reference span size of 1 will cause the model to only use one span i.e. as if spans were not implemented.

The three different configurations for reference spans are listed below:

Configuration 1 Default span value.

Configuration 2 No spans.

Configuration 3 Overlapping spans.

As the behavior of overlapping spans is unknown a parameter sweep should be performed to explore and optimize the performance of the model around this parameter. The value found as optimal will then be used when comparing the three different configurations of reference spans.

To be able to evaluate the spans convergence abilities, single objective Solomon problems are used where driving distance are to be optimized. In turn, to be able to observe the spans abilities to both cover and converge on a large Pareto front the Visma data set is used. Hence, the following three tests are declared to explore RQ2:

Test 2.1 Find the optimal reference span overlap on the Solomon data set.

Test 2.2 Test reference span configurations on the Solomon data set.

Test 2.3 Test reference span configurations on the Visma data set.

Hypothesis

It is expected that in test **2.1** a degree of reference span overlap will be discovered that increases diversity and in turn optimize performance.

It is expected that when comparing the configurations of no spans, overlapping spans and default that overlapping spans will have the optimal performance. This is because of the diversity introduced when increasing the span sizes, further discussed in section 3.2.2. It is expected that using no spans will be considerably worse in terms of performance. This is especially the case for the Visma data set since it includes three objectives and the use of reference spans will help it find fit solutions.

4.4.3 RQ3 - Experimental Plan

The following tests explore MODQPSO in a dynamic environment. The dynamic environment can be tuned in the simulator, as it allows for the spatial severity of the dynamic version of the Solomon problems to be tuned between 0 and 1. The value of the spatial severity represents the percentage of customers removed from the Solomon problem which are to be later introduced as a dynamic change. Spatial severity is further discussed in section 2.1.3. Figure 4.8 shows a dynamic problem initiated with a spatial severity of 0.5 before and after the dynamic change. Hence, in figure 4.8a 50% of the customers are removed from the Solomon

(a) R111.50 initiated to introduce a dynamic change of 50% spatial severity.

(b) R111.50 after the dynamic change.

Figure 4.8: Visualisation of 50% spatial severity on a R111.50.

problem and are later introduced back to the problem as a dynamic change, seen in figure 4.8b.

As spatial severity is an important factor of a dynamic environment, it is crucial to explore the behaviors of MODQPSO when faced with different degrees of spatial severity. Exploring this will show if MODQPSO manages to show similar behavior to other dynamic algorithms such as mQSO. MODQPSO should be able to transfer knowledge about good routes from before to after the occurrence of dynamic changes. As it is based on mQSO, MODQPSO should be able to converge faster than if the algorithm was to be completely restarted after the occurrence of a dynamic change. In order to declare a point of convergence the algorithm is considered to have converged when the fitness remains unchanged for 10 consecutive iterations.

It is of interest to know how a dynamic change affects the performance of MODQPSO. It is unknown whether or not a dynamic change can affect the performance of MODQPSO if introduced before the end condition of the search in MODQPSO is reached.

It is unknown how the dynamic handling of MODQPSO behaves when faced with three objectives. The theory of the use of multiple swarms in MODQPSO is inspired by mQSO. The mQSO algorithm uses multiple swarms to track a range of different local optimums of a single objective problem to enhance the dynamic handling abilities, further explained in section 2.2.1. When introducing multiple objectives there is also an increase of local optimums. For MODQPSO to handle both multiple objectives and a dynamic environment it should be able to track both the current positions related to the Pareto front and positions which have the potential to become a part of a new Pareto front after a dynamic change.

To explore the dynamic abilities of MODQPSO in the topics explained above the following tests are introduced:

Test 3.1 Test how the degree of spatial severity affect the models convergence abilities after dynamic changes.

Test 3.2 Test if introducing some customers while the algorithm is converging improves the resulting fitness on the Solomon data set.

Test 3.3 Test the computational cost required to converge after dynamic changes on the Visma data set for S-PSO and MODQPSO.

Hypothesis

It is expected that dynamic handling of MODQPSO will succeed in transferring knowledge of good routes from before to after the occurrence of a dynamic change. This will be observed if MODQPSO is able to converge faster than it would if it was completely restarted. This is tested in **3.1**.

The MODQPSO will be able to improve the performance when initiated on a VRP with less customer before a dynamic change, as it will be less complex to find optimal solutions on a smaller VRP. When the remaining customers are introduced the algorithm will be able to place those customers at optimal positions within routes that are already generated. This will create solutions with improved performance compared to when all customers are known from the beginning. This hypothesis is tested in test **3.2**.

The dynamic version of MODQPSO will be able to converge faster after a dynamic change compared to a complete restart version of MODQPSO or plain S-PSO. Additionally the algorithm will improve the fitness in test **3.3**.

4.5 Experimental Setup

4.5.1 Parameters

The parameter values of table 4.7 is used for all runs if no other information is declared.

Parameter	**Value**
Iterations	150
Swarms	11
Quantum particles	20
Neutral particles	7
Radius	2
Neighborhood size	3
Inertia	0.35
C1	1.5
Reference span size	1/11

Table 4.7: Parameters used for experimental testing of the MODQPSO algorithm

4.5.2 RQ1 - Search space representation

Four different representations are tested to measure the performance and computational cost of MODQPSO. For test **1.1** the model is tested on 10 problems from the Solomon data set. Every representation is ran 30 times on all of the problems. The problems vary in the amount of customers and how strict the constraints are to ensure that the model is optimized to work on a variety of problems. The problems used in test **1.1** are shown in table 4.8

Problem name	**Time Window**
R111.25	Mixed
R211.25	Wide
R101.50	Narrow
R106.50	Mixed
R111.50	Mixed
RC102.50	Mixed
RC207.50	Wide
R202.100	Wide
RC207.100	Wide
C106.100	Narrow

Table 4.8: The Solomon problems used in test 1.1 in RQ1.

For test **1.2** the model is tested on 3 consecutive problems from the Visma data set. Every representation is ran 10 times on all of the problems.

4.5.3 RQ2 - Reference span strategies

For test **2.1** the model will be tested on Solomon problem R111.50. The test consist of doing a parameter sweep to find optimal parameter values for overlaps and will follow the same structure as in the preliminary tests. The goal is to tune the parameter value for the reference span size. A range of 12 different spans sizes between 0.091 and 1 will be tested on the Solomon problem, each for 30 runs of the model.

For test **2.2** the three span configurations are tested to measure the performance and computational cost. The configurations are tested on 7 problems from the Solomon data set. Every representation is ran 30 times on all of the problems. The problems used in test **2.2** are shown in table 4.9

For test **2.3** the model is tested on 3 consecutive problems from the Visma data set. Every representation is ran 10 times on all of the problems.

Problem name	Time Window
R111.25	Mixed
R211.25	Wide
R111.50	Mixed
RC102.50	Mixed
RC207.50	Wide
R202.100	Wide
RC207.100	Wide

Table 4.9: The Solomon problems used in test 2.2 in RQ2.

4.5.4 RQ3 - Dynamic optimization

Test **3.1** is conducted on Solomon problem R111.50 and R202.100. The spatial severity values of 5%, 10%, 15%, 30%, 50% and 80% are tested on both of the problems. Each of the spatial severity values are ran 30 times on the model.

Test **3.2** uses a total of 150 iterations per run where a dynamic change is introduced after a set number of iterations of 0, 10, 15, 30, 40, 75, 100, 110, 120, 135 or 140. For each of these the model is ran 30 times on the Solomon problem RC202.100. The test is run using two separate values of spatial severity at 5% and 80%.

For test **3.3** S-PSO and four configurations of MODQPSO are tested. These four configurations are MODQPSO using representation 1 and 4 and MODQPSO using representation 1 and 4 where the model is completely restarted after the occurrence of a dynamic change. Three consecutive problems from the Visma data set are used where the model is ran 10 times on all of the problems.

4.6 Experimental Results

4.6.1 RQ1 - Results

The experimental plan for research question 1 introduced four representations that will be used to explore how search space representations affects both the performance and efficiency of the model. In both test **1.1** and test **1.2** representation 1 is utilized to be able to evaluate the performance of the other representations.

Test 1.1

Test **1.1** uses the Solomon data set. To get an overview of the average found fitness of all representations figure 4.9 illustrates the resulting fitness compared to the benchmark on 8 of the 10 chosen Solomon problems. Figure 4.9 uses the

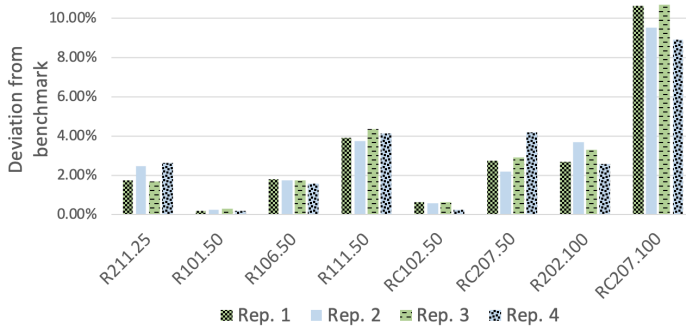


Figure 4.9: Average fitness found RQ1 representations, compared to the benchmark on 8 Solomon problems.

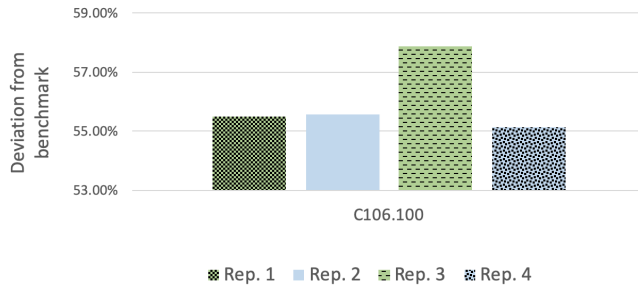


Figure 4.10: Average fitness found RQ1 representations, compared to the benchmark on C106.100.

benchmark solution in each Solomon problem as a comparison i.e. *found fitness divided by benchmark fitness - 1*. This means that a score of 0 represents that the benchmark is met, while a score n above 0 means that the found solution has a driving distance n% longer then the benchmark. In the figure it can be observed that for every problem the average fitness found by all the representations are fairly similar. Information such as standard deviation, run time and the fitness of each individual run made it possible to perform further analysis.

The results of Solomon problem R111.25 is not included in figure 4.9 as all four representations found the benchmark solution. The results of Solomon problem RC106.100 are in a separate figure 4.10. For this particular problem the average fitness was considerable worse as the driving distance was around 55% - 58% worse than the benchmark. Including the results of RC106.100 in figure 4.9 would make the difference between the representations unreadable. Solomon problem

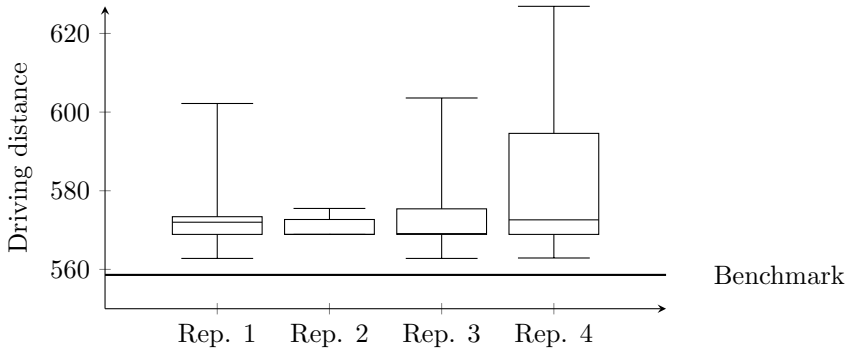


Figure 4.11: Comparing RQ1 representations on Solomon RC207.50.

C106.100 does unveil a weakness of the model as the found fitness is considerably worse then the benchmark. The reason of the issue does not seem to be because of any representation, as each representation retrieve a fairly similar fitness. Hence, the issue may be lay in another part of the model.

When analysing the standard deviation, representation 2 shows a noteworthy difference compared to the other representations. Solomon problem R207.50 highlights this behavior as seen in figure 4.11. The resulting standard deviation of representation 2 on R207.50 is 2.4 in found driving distance, while representation 1, 3 and 4 is at 9.38, 11.03, 16.73 respectively. This trend is also seen when comparing the found standard deviation from all problems in test **1.1**. Overall, representation 3 has a 27.4% higher standard deviation than representation 2, while representation 4 is 24.1% higher and representation 1 is 17.8% higher.

		X			
	Run time Y/X	Rep. 1	Rep. 2	Rep. 3	Rep. 4
Y	Rep. 1	1	1.038	0.855	1.012
	Rep. 2	0.964	1	0.824	0.976
	Rep. 3	1.169	1.213	1	1.184
	Rep. 4	0.988	1.025	0.845	1

Table 4.10: Comparing total computational cost of RQ1 representations using the total run time on 30 runs on 9 different Solomon problems.

Representation 3 does not manage to find the best average driving distance on any of the Solomon problems with 50 or more customers which can be seen seen in figure 4.9 and figure 4.10. In figure 4.10 the deviation from the benchmark of representation 3 is 57.9% compared to representation 4 at 55.1%. Represen-

tation 3 also deviates from the rest in terms of computational cost. Table 4.10 shows a direct comparison between the run times of the different representations. Representation 2 and 4 decreases the run time between 1% and 4% compared to representation 1. However, representation 3 increases the run time with between 16.6% compared to representation 1.

Table 4.11 illustrates how many problems each representation managed to find the average best driving distance compared to the other representations. An equally good driving distance was found by both representation 1 and 4 on problem R101.50. In this table it can be observed that representation 4 manages to find the average best solutions to a majority of the problems. Representation 4 does however have an issue with R207.50 seen in figure 4.11 where it performs worse than the other representations.

	Nr. of mean best
Rep. 1	1
Rep. 2	2
Rep. 3	1
Rep. 4	6

Table 4.11: The performance of RQ1 representations on 9 Solomon problems.

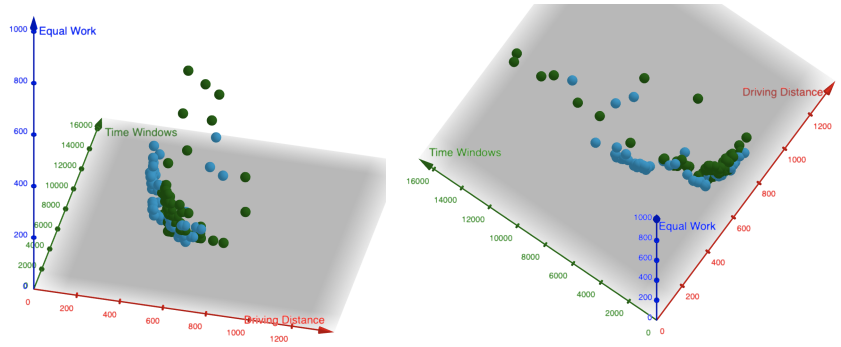
Test 1.2

After running test **1.2** the results from all runs on each problem are used to calculate both the ideal point and nadir point. The nadir point is then used as the reference point to calculate the HVI for each run and the ideal point is used to calculate the ideal volume which can be used to help contextualize the produced data. HVI is further discussed in section 2.1.4.

The results from two runs on Visma 1 are presented in figure 4.12 where one can observe that a front is dominating the other. The HVI for the front closest to the origin is 66.517% of the ideal volume, where the other front has a HVI of 55.397%. A perfect HVI has the value of 100%, as HVI scores closer to the ideal HVI makes for a Pareto front that is closer to the origin and the true Pareto front. This can be observed as the front closest to the origin in figure 4.12 is also the one with the largest HVI.

A solution from the Visma 1 problem is shown in figure 4.13. This solution has a minimized time window deviation on the Pareto front of figure 4.12. Figure 4.13 shows both how the model is able to find solutions which minimizes the deviation of time window and how tasks are distributed amongst the employees.

Table 4.12 shows all HVI in relation to the ideal HVI of the given Visma problem. The table shows that both representation 2 and 3 scores well on two



(a) View 1. (b) View 2.

Figure 4.12: Visualisation of two Pareto fronts from the Visma data set.

		Rep. 1	Rep. 2	Rep. 3	Rep. 4
Visma: 1	Avg.	58.897%	**59.793%**	59.531%	60.143%
	Best	62.189%	63.080%	**66.517%**	63.273%
Visma: 2	Avg.	76.887%	78.806%	**78.955%**	77.780%
	Best	83.985%	**86.063%**	81.660%	84.980%
Visma: 3	Avg.	52.851%	51.491%	53.284%	**54.047%**
	Best	56.180%	53.529%	57.314%	**59.978%**

Table 4.12: The HVI of RQ1, found Pareto fronts of the representations in relation to the ideal volume on the Visma data set.

out of three Visma problems where the best score for each of the problems is highlighted in bold. Representation 4 manages to find both the best and the best average volume on the third problem.

Hypothesis evaluation

The hypothesis for representation 2 was that it would be able to have the same performance as representation 1 while decreasing computational cost. In addition the time windows are a deciding factor of this behavior. Test **1.1** included a variety of problem types where some had strict time windows, notably R101.50 and C106.100. The results revealed that there was no noticeable improvement in fitness between the two representations in relation to strict time windows, as both representations achieved the same fitness. Problems such as RC207.50 and R202.100 had large time windows. For those problems the results also indicated that there was no clear relation between how wide or strict the time windows are

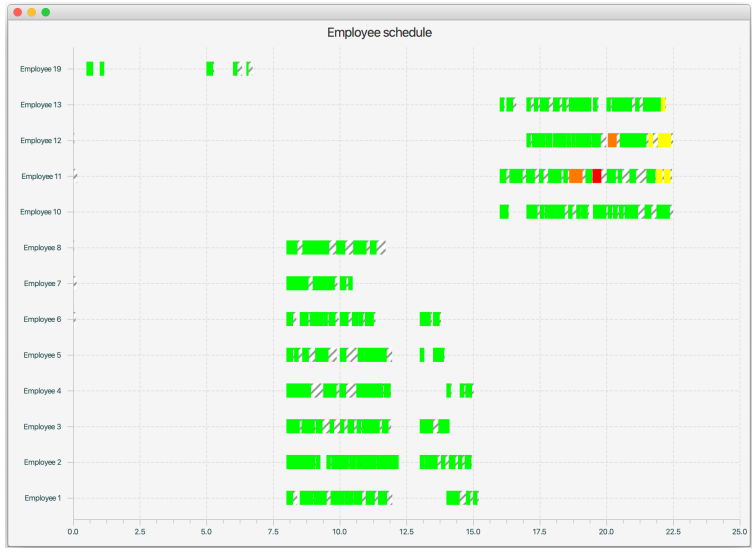


Figure 4.13: A schedule for Visma 1, from a solution which has a low deviation of time windows.

and the fitness scores of the two representations.

However, beyond the time window hypothesis representation 2 was able to lessen the standard deviation of the found solutions, lessen the computational cost by 3.8% and in most cases find a better average fitness and HVI than representation 1. In conclusion storing both the next and previous customer in the representations seems to be redundant information. The concept of storing both the next and previous introduce unnecessary complexity and noise that in the end does not benefit the algorithm in terms of fitness.

The hypothesis surrounding the hybrids of S-PSO and I-PSO was based on the literature of the S-PSO algorithm. The hypothesis was that simple representations such as representation 3 and 4 would have a decreased performance compared to representation 1. However, representation 3 and 4 would decrease computational cost. Representation 3 does not align with the hypothesis as it increased the run time, but did have a decreased performance compared to representation 1. Representation 4 managed to decrease the computational cost with 1.3% and also improved the average fitness on most problems in test **1.1**. It seems that differences between MODQPSO and the original S-PSO and I-PSO, such as decomposition and quantum particles, makes a more complex discrete representation unnecessary and even counterproductive.

		MODQPSO: Rep. 4		Benchmark		
		DD	NV	DD	NV	Author
R101.50	Avg.	1046.2	12.3	N/A	N/A	
	Best	1044	12	1044	12	KDMSS
R106.50	Avg.	805.5	8	N/A	N/A	
	Best	793	8	793	5	KDMSS
R111.50	Avg.	736.6	7.1	N/A	N/A	
	Best	710.5	7	707.2	7	KLM
C106.100	Avg.	1286	12	N/A	N/A	
	Best	994.2	11	828.9	10	OCGA
R202.100	Avg.	1107.5	5.3	N/A	N/A	
	Best	1075.9	5	1079.4	6	OCGA
RC207.100	Avg.	1147.4	4.9	N/A	N/A	
	Best	1076.5	4	1053.6	6	OCGA

Table 4.13: Solomon results from MODQPSO representation 4 compared to the best known solutions.

Table 4.13 shows a direct comparison between some benchmark solutions of Solomon problems and the found solutions of representation 4. It can be observed that in general the MODQPSO algorithm is able to compete with the benchmarks.

Representation 4 improved the average found fitness compared to the other representations, and is viable in terms of the resulting HVI. Hence, it is considered the most versatile of the proposed representations. Due to this representation 4 is used in the following experiments if no other information says otherwise.

4.6.2 RQ2 - Results

The experimental plan for research question 2 introduced three different span configurations that explores the possibilities of the proposed reference spans.

Test 2.1

After running the series of parameter values against the Solomon problem it was discovered that a specific range of reference span sizes improved the fitness of the produced solutions. A more specific search was performed around this area where the findings are presented in figure 4.14. It can be observed that the size of 0.106 was able to find the benchmark solution of R111.50.

The configurations of the MODQPSO model was previously not able to retrieve the benchmark solution on R111.50. Even with 3000 separate runs the

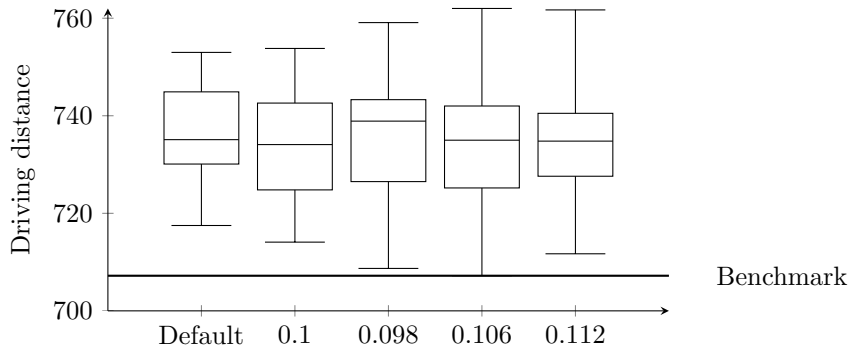


Figure 4.14: Result from different reference span sizes, Solomon R111.50.

algorithm was unable of finding the benchmark solution when using the default configuration for reference spans.

Being able to find the benchmark solution indicates a strong ability of local search. It was concluded that a span value of 0.106 was optimal and is the value used in test **2.2** and test **2.3** as the default reference span size used for the overlapping configuration.

Test 2.2

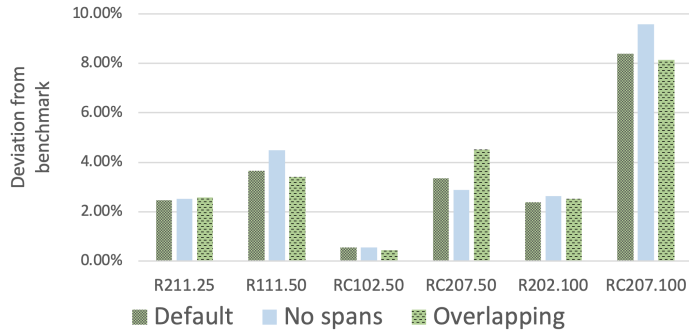


Figure 4.15: Average fitness found by RQ2 span configurations, compared to the benchmark on 6 Solomon problems.

Test **2.2** compares the default span representation of MODQPSO with the use of no spans and overlapping spans. The test results of 6 out of 7 Solomon problems are shown in figure 4.15. The figure shows the average deviation in driving

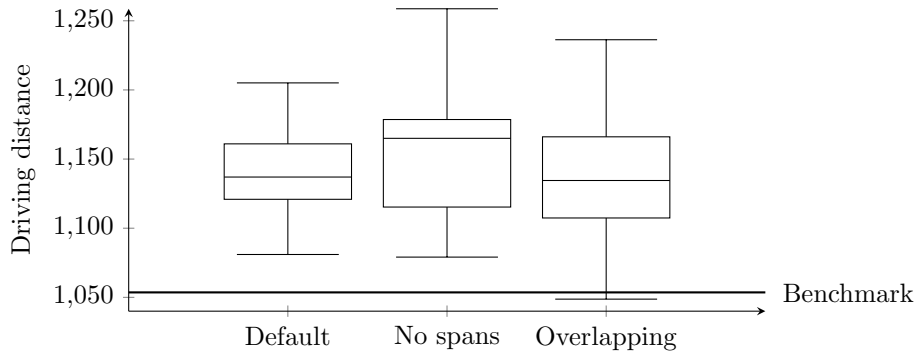


Figure 4.16: Comparing RQ2 span configurations on Solomon RC207.100.

distance compared to the benchmark of each problem for each configuration i.e. the average found driving distance divided by driving distance of the benchmark. As in RQ1 the results of Solomon problem R111.25 is not included in figure 4.15 as all runs found the benchmark solution.

The resulting fitness of Solomon problem RC207.100 is shown in figure 4.16. This figure illustrates the trend of the difference in standard deviation. Default spans are able to reliably find the same solutions after each run. The overall standard deviation increases by 10% between default spans and overlapping spans, and by 20% between default and no spans. The figure marks this trend as the sizes of the boxes for no spans and overlapping spans are larger than that of default spans.

The results of test **2.1** indicates a strong ability of local search using overlapping spans. This ability is further affirmed in figure 4.16 where overlapping spans are able to find a better solution than the benchmark on the particular problem. The found solution has a driving distance of 1048.7 compared to the benchmark of 1053.58.

Test 2.3

Test **2.3** also compared the three different span configurations. Test **2.3** focuses on the abilities for reference spans to converge and find a wide range of solutions when faced with multiple objectives.

The HVI are presented as the percent of similarity to the ideal volumes. This process is further explained in section 2.1.4. The resulting HVI's of test **2.2** are shown in table 4.14. On the problem "Visma: 1" overlapping spans retrieves both the best found and the best average volume. However, the average HVI is only 0.434% larger than that of default spans and the best HVI is 0.142% larger

		No spans	Default	Overlapping
Visma: 1	Avg.	76.983%	80.206%	**80.348%**
	Best	80.180%	85.138%	**85.572%**
Visma: 2	Avg.	75.753%	**76.867%**	76.428%
	Best	**81.808%**	80.169%	81.762%
Visma: 3	Avg.	79.113%	**80.180%**	80.166%
	Best	82.542%	**88.910%**	86.004%

Table 4.14: The HVI of RQ2 found Pareto front of span configurations in relation to the ideal volume.

than the best of default. These differences are low compared to the ones found in problem "Visma: 3". In the "Visma: 3" problem the default spans manages to retrieve a best volume 2.9% larger than that of overlapping spans while the average remains similar to that of overlapping spans.

Hypothesis evaluation

The hypothesis around reference spans was that they should increase performance. In other words both default and overlapping spans should have an improved performance over the no spans configuration. In addition the overlapping spans should improve performance over default configurations.

However, the tests on the Solomon data set did not align with the hypothesis as strongly as expected. Removing reference spans does on average weaken performance. However, Solomon problems RC207.50 in figure 4.15 show that a configurations of MODQSO without reference spans may also be able to find promising results.

The results from the Visma data set does however indicate that reference spans are able to improve performance on multiple objectives and hence being able to cover a wider area of the Pareto front. In summary it was found that using no reference spans may still allow the algorithm to find good solutions on single objective problems, but it negatively impacts the performance in multi-objective problems.

The hypothesis of overlapping spans showed to be somewhat correct. If the focus was solely on the Solomon data set the hypothesis would align. The resulting HVI from the Visma data set does however imply that the default configuration is the superior configuration. Hence, it is used in further experiments as multiple objectives are a key point of the thesis.

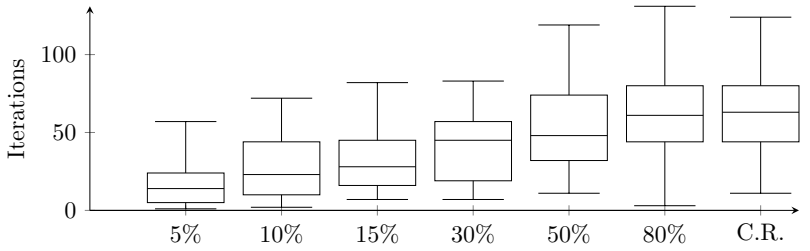


Figure 4.17: Comparing the number of iterations to converge after a dynamic change on Solomon R202.100.

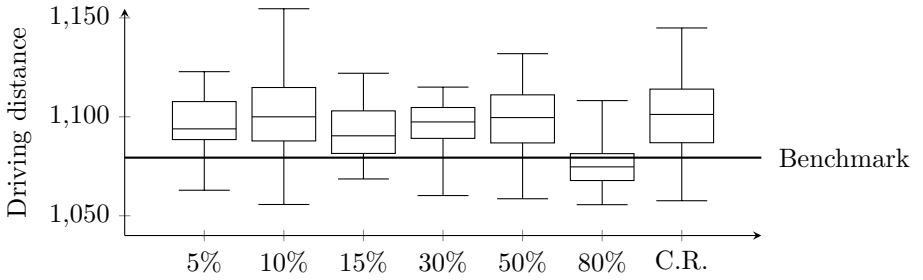


Figure 4.18: Comparing RQ3.1 on Solomon R202.100.

4.6.3 RQ3 - Results

Test 3.1

To verify how well the algorithm was able to handle dynamic changes, and in other words able to transfer knowledge of the environment before a change to after, a complete restart (C.R.) version was used as explained in section 2.1.3.

Figure 4.17 shows the number of iterations required for the model to converge after a dynamic change. The figure indicate that a low spatial change will require a low number of iterations for the model to converge at a new optimum. The larger the spatial severity, the higher the number of iterations required to converge.

Figure 4.18 shows how the model was able to converge in terms of the resulting fitness depending on the degrees of spatial severity. The same behavior was observed when running the model on problem R111.50. Figure 4.19 shows the found driving distance iteration by iteration and shows the same behavior as in figure 4.17 where a low spatial change requires few iterations to converge. It also shows that when the model has already converged before a dynamic change it

is able to find solutions on par or even superior to that of the complete restart version. This is also shown in figure 4.18 on the problem R202.100 where there is a tendency of finding superior results on average after a change of 80% spatial severity, which is further explored in test **3.2**.

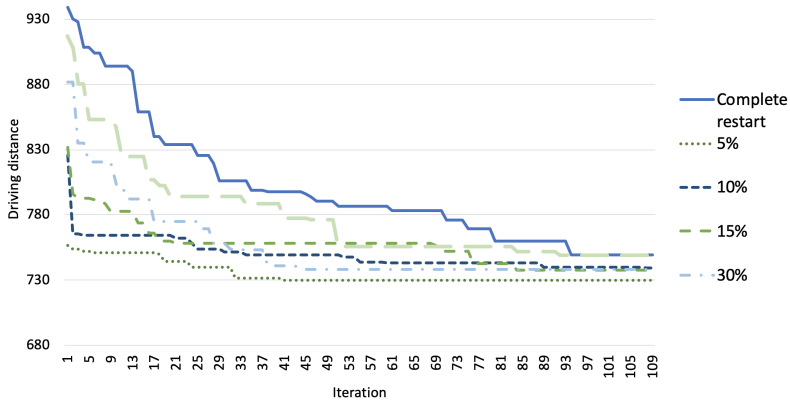


Figure 4.19: Convergence after a dynamic change with different values of spatial severity. On Solomon problem R111.50.

Test 3.2

Test **3.2** uses a total of 150 iterations where a dynamic change is introduced after a set number of iterations. The Solomon problem RC202.100 is used after test **3.1** showed that this particular problem had a high tendency of improvement when faced with a low spatial dynamic change.

The results of **3.2** on a 5% spatial change, is shown in figure 4.20. The figure shows no strong pattern in changed fitness based on when the change is introduced. This introduces a new test to conclude why 5% spatial had a superior performance compared to complete restart in test **3.1**. The major difference between the set up of test **3.1** and **3.2** is that test **3.2** uses a set amount of iterations before and after the introduction of a change. In test **3.1** the dynamic versions are able to first converge using 150 iterations on a sub-set of the problem before another 150 iterations is ran on the full problem. The complete restart version only uses 150 iterations in total.

The hypothesis is that the complete restart is not able to properly converge on the tested Solomon problems in only 150 iterations and that another 150 iterations will make it converge properly. In the test two separate complete

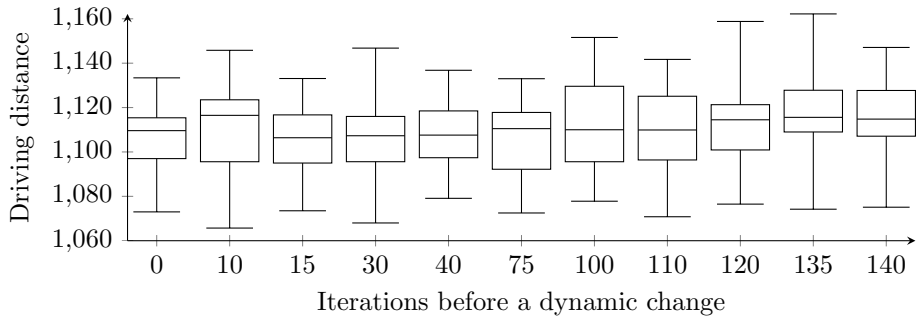


Figure 4.20: Comparing RQ3.2 on Solomon RC202.100 with 5% spatial severity.

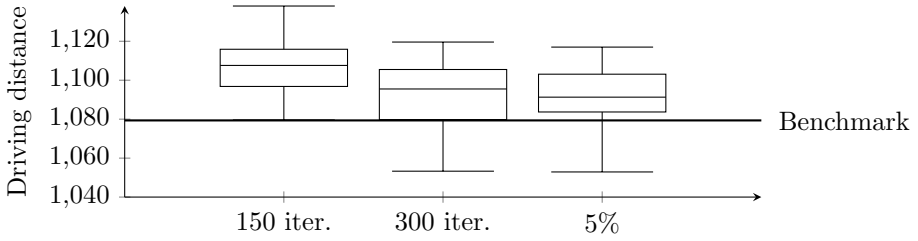


Figure 4.21: Comparing complete restart using both 150 and 300 iterations with a dynamic run on 5% spatial severity.

restart versions were given 150 and 300 iterations to converge. Figure 4.21 show the resulting fitness, where it is shown that 300 iterations makes for similar results as to that of a dynamic run on 5% spatial severity. The hypothesis seems to stand correct. The results also indicates that knowledge is transferred from before to after a dynamic change, as 150 iterations before and after a change gives similar results to 300 iterations on the full problem. The complete restart version with 300 iterations is however a lot slower than MODQPSO.

The results of dynamic runs with 80% spatial severity showed interesting results. Figure 4.22 shows these results and when the model is ran for 30-40 iterations on a a sub-problem before introducing the rest of the customers it will find superior solutions compared to a run that has knowledge about all customers for all iterations. However, when the change is introduced after 100 iterations the remaining iterations are not sufficient enough to converge and the fitness is worsened.

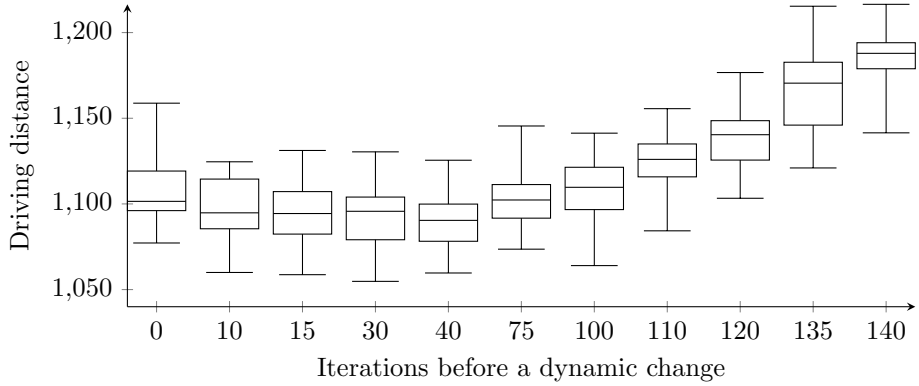


Figure 4.22: Comparing RQ3.2 on Solomon RC2020.100 80% spatial severity.

Test 3.3

Test **3.3** was run on three Visma problems where the simulator saved the Pareto front for each iteration of each run. This made it possible to calculate the HVI for each iteration and observe the change iteration by iteration. Figure 4.23 is the result of this process on the problem Visma 1. For this test the S-PSO is as implemented in the paper Gong et al. [2012]. S-PSO is not designed for multiple objectives, but is included as a comparison to be able to observe the multi-objective abilities of MODQPSO. S-PSO is compared to both dynamic runs of MODQPSO marked as D.Rep. 4 and D.rep. 1, referring to the discrete representation 1 and 4 from research question 1, and complete restart runs of MODQPSO marked as C.R.Rep. 1 and C.R.Rep. 1.

		S-PSO	**D.Rep.4**	**D.Rep.1**	**C.R.Rep.4**	**C.R.Rep.1**
Visma 1	Avg.	54.278%	81.737%	77.447%	81.955%	80.382%
	Best	57.026%	90.944%	84.851%	85.968%	84.701%
Visma 2	Avg.	53.131%	78.517%	79.835%	77.592%	76.697%
	Best	56.631%	84.944%	89.264%	77.592%	85.653%
Visma 3	Avg.	50.693%	72.015%	73.370%	75.114%	73.466%
	Best	54.253%	78.200%	80.289%	83.149%	78.015%

Table 4.15: The hypervolume indicators of RQ3 in relation to the ideal volume.

The resulting fitness of runs against the Visma set shows a similar trend to that of the results of test **3.1**. The resulting fitness does not seem to be affected by whether or not the model is set to dynamic or complete restart at a 5% dynamic

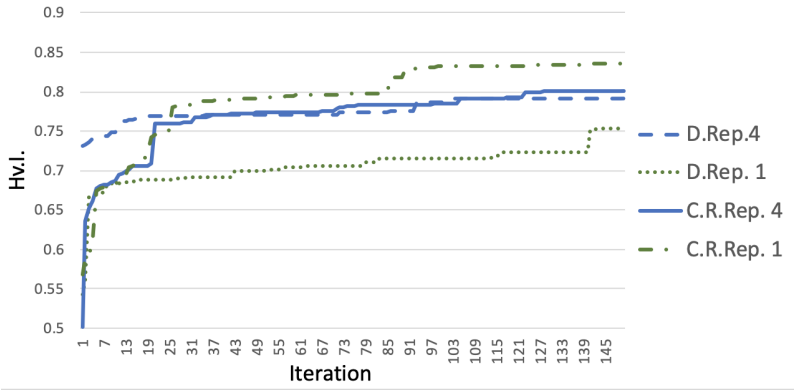


Figure 4.23: Convergence after a dynamic change on the VRP Visma 1.

change independent of the data set. Table 4.15 shows this as the resulting HVI of dynamic versions are about the same as their corresponding complete restart version on three separate Visma problems.

Figure 4.23 also shows results similar to that of test **3.1**. Dynamic runs of representation 4 manages to start with a superior fitness to that of complete restart versions. The dynamic run of representation 4 also converge faster than the others. However, the expected result was that the dynamic version of representation 1 should behave similarly to that of dynamic representation 4. This leads to the addition of a separate test to show if representation 1 is able to show the same dynamic abilities as representation 4 on the Solomon data set.

In this extra test representation 1 and 4 is first able to run 150 iterations on the Solomon problem R111.50 before a dynamic change of 5% occurs. Figure 4.24 shows three separate runs of both representations where it shows a clear trend. Representation 1 is not able to learn from the VRP before the dynamic change as its fitness starts much worse than that of representation 4. The reason is that representation 1 uses the velocity update function of S-PSO which seems to not be able to handle an updated search space. This is due to the fact that a dynamic version based on the S-PSO velocity update is almost indistinguishable to a complete restart version.

Hypothesis evaluation

The dynamic handling of MODQPSO succeeded in transferring knowledge of vehicle routes from before to after the occurrence of a dynamic change, as seen in test **3.1**. In test **3.1** on Solomon problem R111.50 the dynamic version required

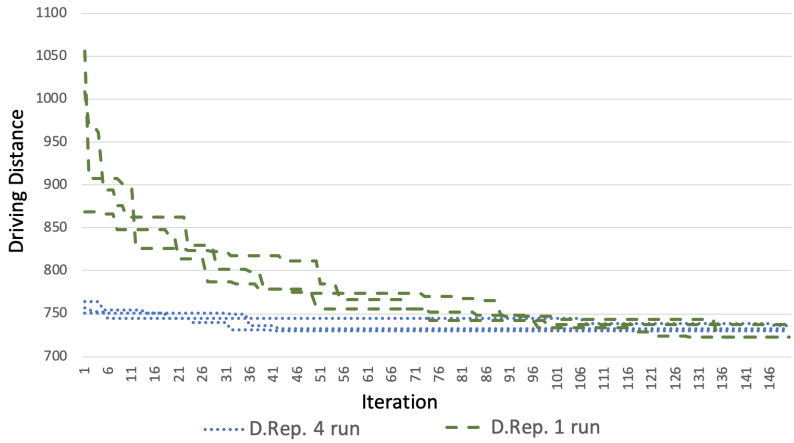


Figure 4.24: Convergence of representation 1 and 4 on Solomon R111.50.

only 8.3 iterations to converge on average, compared to 39.27 iterations for a complete restart version. This is a 78.9% decrease in iterations required to converge. On two other problems the decrease in required iterations was at 89.3% and 71.9% on Solomon problem RC207.50 and R202.100 respectively. This can be linked directly to a decrease of computational cost as each iteration has an equal cost.

The hypothesis stated that the dynamic version of MODQPSO would find fitter solutions than with a complete restart. In cases with a high spatial severity this was shown to be true. With lower values of spatial severity the fitness did neither improve nor decrease. Figure 4.24 shows that if the complete restart version gets extra time to converge the same results are possible in the case of low spatial dynamic changes. Otherwise the dynamic version finds superior results.

Interestingly a phenomenon occurred when the spatial severity was at high values. At a 80% spatial change the standard deviation lowered significantly and the fitness of the solutions improved. The hypothesis stated that a dynamic version of MODQPSO are able to improve the fitness when initiated on a sub-problem of a VRP before introducing the full amount of customers. This is valid when the spatial change is around 80% i.e. learning on only 20% of the VRP before introducing the remaining 80% of customers. This is an interesting aspect of the model and may work as an optimization technique. This trend was not observed on lower spatial changes in the range between 5% and 50%.

The dynamic version was able to maintain, but not increase performance, in multiple objectives on the Visma data set. However, representation 4 has a much faster convergence rate. This means that representation 4 has a potential of lowering computational cost as the number of iterations required to converge may be lowered depending on the spatial severity of the dynamic change.

		MODQPSO: Rep. 4		**S-PSO**	
		DD	NV	DD	NV
R101.50	Avg.	1070.59	11.74	1060.1	11.8
	Best	1098.7	11.0	1100.7	11
R106.50	Avg.	850.88	7.26	851.2	7.5
	Best	830.2	7	865.9	7
R111.50	Avg.	736.97	6.98	722.6	6.9
	Best	769.6	6	756.4	6
C106.100	Avg.	1316.07	11.14	828.94	10
	Best	1166.0	11	828.94	10
R202.100	Avg.	1145.95	4	1259.84	3.5
	Best	1097.7	4	1247.03	3
RC207.100	Avg.	1190.06	3.96	1192.55	3.7
	Best	1279.9	3.0	1130.37	3

Table 4.16: Solomon results from MODQPSO representation 4 compared to S-PSO.

In addition, MODQPSO with representation 4 is compared directly with S-PSO in table 4.16. The main objective of S-PSO is to lower the number of vehicles where driving distance is considered as a secondary objective. MODQPSO always return a front of solutions on the Solomon problems where it prioritizes both objectives equally through decomposition. In other figures and tables throughout this chapter the driving distance has been the priority when presenting Solomon results. However in this table the solutions with the lowest number of vehicles from the Pareto front are used to make a fair comparison with S-PSO. It can be observed that MODQPSO is able to retrieve solutions with a low number of vehicle and a lower driving distance on most problems. It does however struggle with finding the solutions with the lowest number of vehicles on C106.100 and R202.100.

Chapter 5

Evaluation and Conclusion

The chapter presents an overall discussion of the model based on results from experimental tests and goal evaluation in section 5.1, followed by the contributions made to the field in section 5.2. Finally the future work is presented by suggesting improvements and possibilities that can be explored in section 5.3.

5.1 Discussion and Goal Evaluation

The goal was to explore a combination of a Multi-objective Discrete PSO and multi Quantum Swarm Optimization that optimizes performance and lowers computational cost in a dynamic environment. These research questions are answered through the use of a series of tests.

Research question 1 *How can modifications to the search space improve performance while decreasing computational cost?*

Exploring the search space proved to be more important for the decrease in computational cost than for the performance. The tests related to RQ1 unveiled a representation that both decreased computational cost and improved performance on a non-dynamic VRP. However, if no exploration had been performed around the search space the dynamic abilities of the proposed algorithm would not have been able to improve computational cost.

An important additional test related to RQ3 was performed when the different representations were used with the dynamic Visma data set. This test showed that the S-PSO representation was not able to use the information gathered from before a change and required as many iterations to converge as a newly initiated algorithm. Because of RQ1, another representation based on I-PSO was implemented and referred to as representation 4. The representation was able to drastically reduce the required iterations to converge and even improve the

found fitness after dynamic changes. This shows that such fundamental concepts as the search space of swarm algorithms should not be overlooked, but rather thoroughly explored. Even when larger meta-heuristics and concepts such as multi quantum swarms and decomposition are used.

Research question 2 *How can reference spans improve performance while maintaining the computational cost when handling multiple swarms in multi-objective optimization?* Research question 2 was conducted to explore the concepts believed to be fundamental for quantum particles to work with decomposition. The research question uncovered some weaknesses in the model design.

A significant part of the model is the use of reference spans. Even though RQ2 benefit from the spans in terms of optimization performance in multiple objectives their importance may have been overvalued. The use of spans prevent communication across swarms as the heuristic used to decode each particle position is coupled with the reference span of each swarm. Cross swarm communication such as the island model has proven to work in other publications. The consequence of utilizing reference spans is that such cross swarm communication can not be implemented without modifications to the concept of the spans.

Research question 3 *How does dynamic environments affect MODQPSO in terms of computational cost and performance?* Research question 3 unveiled a strong ability for the model to lower computational cost when faced with low spatial dynamic changes. In the related tests MODQPSO is able to find and maintain the same performance using 10%-30% of the computational cost when faced with a low spatial dynamic change. Interestingly when the dynamic change is large at, 80% spatial, the performance of the MODQPSO algorithm improves. This seems to indicate that an optimization technique can be based on the concept of dynamic changes. The way MODQPSO is able to learn on a small set of customers before the remaining is introduced seem similar to methods that decomposes a VRP problem to several sub problems of customer clusters as seen in Lian and Castelain [2010]. However, further exploration of this area is not aligned with the research goal as such changes does not lessen computational cost.

In general tests on both RQ1 and RQ3 revealed that the model struggled with the Solomon problem R202.100 and C106.100. The problem lies in retrieving the solutions with the minimum number of vehicles. Both the issue with C106.100 and R202.100 seems to be similar in nature. A theory of why this happens is due to the chosen heuristic. The heuristic is not designed to retrieve solutions with a substantially worse driving distance which seems to be necessary to allow the use of the minimum number of vehicles for problem R202.100. Another reason may be that the swarms converges around solutions that uses more than the minimum number of vehicles and does not have enough diversity to locate solutions using the minimum number of vehicles. However, an ability to compete with the state

of the art in solving every Solomon problem was not the main goal of the model. Future work could be made in relation to this issue.

In conclusion, the results from the three research question has established a base of how the proposed model can combine multi-objective Discrete PSO with multi Quantum Swarm Optimization. However, there are still more to be researched related to all of the fields where a dynamic PSO remains relatively untouched compared to the other fields. The proposed model takes a small step in unveiling knowledge of a specific and niche part of PSO as a technique, but in an area that represents a lot of potential.

5.2 Contributions

In the thesis a model has been proposed that is able to utilize established discrete representations to solve discrete problems such as the VRP. More so the model is able to find solutions to VRP's that are on par with the state of the art. When faced with the dynamic VRP the model improves fitness while simultaneously reducing the required computational cost.

The model extends the I-PSO representation with heuristics as used in S-PSO with promising results in both dynamic and non-dynamic environments. In the thesis it was proposed to modify S-PSO to not store both the previous and next customer in the representation. Through experimental tests it was discovered that storing the previous customer did not notably change the performance of the model. Storing this information seems to be redundant and could be removed in S-PSO to lower computational cost.

The model has continued upon Blackwell's work on mQSO, where it is extended to support both discrete environments and multiple objectives. This is achieved using decomposition and the proposed reference spans.

5.3 Future Work

Through the exploration of the research goal interesting concepts and ideas where uncovered. These concepts and ideas where either not strictly aligned with the research goal or too large to be handled by the scope of the thesis. However, these may be of interest if the work in the thesis is to be expanded upon.

- **Additional MOO techniques:** The focus on multi-objective optimization can be taken further through the use of techniques such as swarm communication.
- **Many objectives:** The use of four or more objectives will introduce new challenges for the model that are not currently explored. The effect of

handling many objectives related to dynamic changes are also not explored. This opens a potential to see if there are computational benefits of handling dynamic changes when faced with many objectives.

- **Constraint handling:** The proposed model struggles when faced with a large set of constraints. For instance in the Visma data set there is a "skill level" constraint that makes each employee able to perform a only a specific set of tasks. This is a rather strict constraint that made the model unable of creating feasible solutions for some problems. This is one of the constraints that must be included to truly solve the Visma problem.

- **User centered approach for Visma:** The Visma data set is taken directly from a real "home care" problem. In order to help solve the home care problem a Visma representative informed that the users must be in the focus. A set of adjustments must be designed to accommodate for the users such as breaks or time windows that may be of more importance than the other. For instance if a patient needs immediate attention an employees break can be rescheduled i.e. prioritizing the most important task.

- **Focus on large scale VRP:** The thesis has focused on optimizing problems with customers upwards to 100. There exists a Solomon data set with 1000 customers that can be further explored and optimized towards.

- **Adaptive parameters:** The swarms and the particles could be able to change their parameters while running. In the thesis a manual parameter sweep is performed to find parameters that works well on most problems, but there will always be exceptions. Utilizing adaptive parameters could be beneficial when running the algorithm on a new problem.

- **Dynamic as an optimization technique:** During the experimental tests it was revealed a potential for increased performance and lower standard deviation if the model learned on a sub set of the VRP before introducing the rest of the problem. Further exploration of this behavior as an optimization technique could be performed.

- **Exploring diversity:** Introducing more diversity to the model through the use of another heuristic or swarm communication can increase performance for model. The model solved most Solomon problems well, but some proposed a challenge. Further methods to escape local optimums can aid the model.

Bibliography

Beume, N., Fonseca, C. M., López-Ibáñez, M., Paquete, L., and Vahrenhold, J. (2009). On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082.

Blackwell, T. and Branke, J. (2004). Multi-swarm optimization in dynamic environments. *Applications of Evolutionary Computing. EvoWorkshops 2004. Lecture Notes in Computer Science*, 3005:489–500.

Blackwell, T. and Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE*.

Bu, C., Luo, W., and Yue, L. (2017). Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. *IEEE Transactions on Evolutionary Computation*, 21(1):14–33.

Chen, W., Zhang, J., Chung, H. S. H., Zhong, W., Wu, W., and Shi, Y. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation*, 14(2):278–300.

Coello Coello, C. A. and Lechuga, M. S. (2002). Mopso: a proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1051–1056 vol.2.

Demirtaş, Y. E., Özdemir, E., and Demirtaş, U. (2015). A particle swarm optimization for the dynamic vehicle routing problem. In *2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, pages 1–5.

Duhain, J. G. O. L. and Engelbrecht, A. P. (2012). Towards a more complete classification system for dynamically changing environments. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8.

Eberhart, R. C., Shi, Y., and Kennedy, J. (2001). *Swarm intelligence*. Elsevier.

Eiben, G. and Smith, J. (2015). *Introduction to Evolutionary Computing 2nd edition*. Springer-Verlag Berlin Heidelberg.

El-Sherbeny, N. A. (2010). Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University-Science*, 22(3):123–131.

Gong, M., Cai, Q., Chen, X., and Ma, L. (2013). Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition. *IEEE*.

Gong, Y., Zhang, J., Liu, O., Huang, R., Chung, H. S., and Shi, Y. (2012). Optimizing the vehicle routing problem with time windows: A discrete particle swarm optimization approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):254–267.

Jurgen Antes, J. and Derigs, U. (1995). A new parallel tour construction algorithm for the vehicle routing problem with time windows. Technical report, Technical report, Lehrstuhl f ur Wirtschaftsinformatik und Operations.

Kallehauge, B., Larsen, J., and Madsen, O. B. (2001). Lagrangean duality applied on vehicle routing with time windows-experimental results. *Internal report IMM-REP-2000-8, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.*

Karmakar, S., Dey, A., and Saha, I. (2017). Use of quantum-inspired metaheuristics during last two decades. In *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 272–278.

Kennedy, J. and Eberhart, R. (1995). *Particle Swarm Optimization*. IEEE.

Khouadjia, M. R., Jourdan, L., and Talbi, E. (2010). Adaptive particle swarm for solving the dynamic vehicle routing problem. In *ACS/IEEE International Conference on Computer Systems and Applications - AICCSA 2010*, pages 1–8.

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Kronborg, H. and Stensli, S. (2019). Dynamic multi-objective pso in a discrete search space. Project report in TDT4501, Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology.

Li, L., Jiao, L., Zhao, J., Shang, R., and Gong, M. (2017). Quantum-behaved discrete multi-objective particle swarm optimization for complex network clustering. *Pattern Recognition*, 63:1–14.

Lian, L. and Castelain, E. (2010). A decomposition approach to solve a general delivery problem. *Engineering Letters*, 18(1).

Mavrovouniotisa, M., Lib, C., and Yang, S. (2017). *A survey of swarm intelligence for dynamic optimization: Algorithms and applications*. Elsevier.

Miettinen, K. (1999). *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media.

Nazif, H. and Lee, L. S. (2010). Optimized crossover genetic algorithm for vehicle routing problem with time windows. *American journal of applied sciences*, 7(1):95.

Nebro, A. J., Durillo, J. J., and Coello, C. A. C. (2013). Analysis of leader selection strategies in a multi-objective particle swarm optimizer. In *2013 IEEE Congress on Evolutionary Computation*, pages 3153–3160. IEEE.

Necula, R., Breaban, M., and Raschip, M. (2017). Tackling dynamic vehicle routing problem with time windows by means of ant colony system. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2480–2487.

Parrott, D. and Xiaodong Li (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10(4):440–458.

Qing Zhu, Limin Qian, Yingchun Li, and Shanjun Zhu (2006). An improved particle swarm optimization algorithm for vehicle routing problem with time windows. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1386–1390.

Sintef (2008). Solomon benchmark. `https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/`. Accessed: 2020-06-07.

Sun, J., Feng, B., and Xu, W. (2004). Particle swarm optimization with particles having quantum behavior. *IEEE*.

Unger, N. J., Ombuki-Berman, B. M., and Engelbrecht, A. P. (2013). Cooperative particle swarm optimization in dynamic environments. In *2013 IEEE Symposium on Swarm Intelligence (SIS)*, pages 172–179. IEEE.

Zapotecas Martínez, S. and Coello Coello, C. A. (2011). A multi-objective particle swarm optimizer based on decomposition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 69–76.

Zeng, H. Y. (2019). Improved particle swarm optimization based on tabu search for vrp. *Journal of Applied Science and Engineering Innovation*, 6(2):99–103.

Zhang, Q. and Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731.

Zitzler, E., Brockhoff, D., and Thiele, L. (2007). The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 862–876. Springer.