Kevin Kellermann

Multi-Objective Optimization of Multi-Agent Path Planning using a Co-evolutionary Genetic Algorithm

FAKULTÄT FÜR
INFORMATIK

Multi-Objective Optimization of Multi-Agent Path Planning using a Co-evolutionary Genetic Algorithm

Master Thesis

Author
- Kevin Kellermann -

December 12, 2020

Professor: Prof. Dr. Sanaz Mostaghim, Chair of Computational Intelligence
Advisor: Sebastian Mai
Advisor: Jens Weise

Kevin Kellermann: Multi-Objective Optimization of Multi-Agent Path Planning using a Co-evolutionary Genetic Algorithm
Otto-von-Guericke-Universität
Magdeburg, 2020.

Abstract

Multi Agent Pathfinding (MAPF) is used in many real world and virtual applications. In recent papers, approaches for solving the MAPF problem multi-objectively (MOMAPF) with genetic algorithms were formulated. This work follows up on this idea and presents a co-evolutionary approach for MOMAPF. The subpopulations are optimized by their own objectives, which are linked to the objectives of the solution for the whole problem. Two variants of the algorithm are implemented: One which optimizes the objectives of the subpopulations multi-objectively (MO SACCGA) and one which optimizes the objectives of the subpopulations with a weighted sum approach (SO SACCGA). These variants are compared to each other to get empirical information about how good they work for MOMAPF and MAPF. The metrics used for comparison are the GD and IGD and the calculation of the weighted sum by using the same weights as for optimizing the subpopulations in the SO SACCGA. The MO SACCGA solves nearly every problem better than SO SACCGA in terms of GD and IGD. The MO SACCGA solves most problems better in terms of the calculation of the weighted sum. The reasons for these results are most likely that the SO SACCGA converges away from pareto optimal solutions, which do not fit the weighted sum and the SO SACCGA has the tendency to fall into local optima, which the MO SACCGA can avoid. These results gives insight on how MOMAPF can be solved using co-evolution and how well weighted sum approaches do compared to multi-objective approaches in evolving the subpopulation in MAPF with co-evolutionary algorithms.

Contents

List of Figures

List of Tables

Acronyms

MAPF	Multi-agent Path Finding
MOMAPF	Multi-objective Multi-agent Path Finding
CCEA	Cooperative Co-evolutionary Algorithm
SA	Slave algorithm
MO	Master Algorithm
SO SA	Single-objective Slave algorithm
MO SA	Multi-objective Slave algorithm
SO SACCGA	Cooperative Co-evolutionary Genetic Algortihm with Single-objective Slave algorithm
MO SACCGA	Cooperative Co-evolutionary Genetic Algortihm with Multi-objective Slave algorithm
GA	Genetic Algorithm

1 Introduction

The Multi agent pathfinding (MAPF) problem is an important research topic. In MAPF path plans for multiple agents are created, which the agents can follow to get to their target destination free of conflict. The usage of Multi agent pathfinding takes place in robot [38] and vehicle coordination [9] automated warehouses [22] and in other applications.

MAPF belongs to the NP-hard problems [46]. This means that it makes sense to apply heuristics and metaheuristics to this problem, to create good but non-optimal solutions in reasonable time. Many cooperative co-evolutionary approaches can be found among these metaheuristics in literature, which apparently lead to good results [5,18,32,33]. Cooperative co-evolution intended that individuals from subpopulations should be evaluated on the basis of the best solution to the whole problem [31]. However, in some of the approaches the individuals of the subpopulations were evaluated by how well they solve their part of the problem according some objectives [32,33].

In literature, several objectives for MAPF can be found. The most used objectives, makespan and sum of costs, are in conflict with each other [38,40]. Therefore, it can be useful to solve the problem multi-objectively to create a set of pareto optimal solutions of which the decision maker can pick the one best suiting their preferences. In Weise et al. [42] a genetic algorithm was implemented and tested, which solved several MAPF problems multi-objectively by optimizing the objectives makespan, sum of costs and overlaps. Optimizing the overlaps objective is supposed to minimize the collisions. While in the classic MAPF collisions are forbidden, Weise et al. argued that, since it was shown in Oliveira et al. [27] that conflict free multi-agent plans in robotics can lead to conflicts anyways, which have to be solved during the execution, conflicts in the multi-agent plan should be accepted and should be weighed against the other objectives by the decision maker [42].

In this thesis, a cooperative co-evolutionary algorithm is used to solve the MAPF problem multi-objectively. The objectives the algorithm optimizes are makespan, sum of costs and overlaps. The algorithm uses different objectives for the subpopulations, which are linked to the objectives of the whole solution. Two variants of the algorithm are implemented: One, which solves the subpopulations with weighted sums and one, which solves the subpopulations multi-objectively. This is done in order to examine the following theses:

- Thesis 1: MAPF multi-objective optimization with co-evolution works better if the subpopulations of the agents are optimized multi-objectively than if the subpopulations are optimized single-objectively with a weighted sum approach.
- Thesis 2: Using a co-evolutionary approach, if the decision maker weighs the objectives of the Multi-objective MAPF problem with the same weights the objectives of the

subpopulations of the agents are weighted using a weighted sum approach, then this weighted sum approach works better than optimizing the objectives of the subpopulations of the agents multi-objectively.

The findings of the first thesis shall help developers and researchers to conceptualize co-evolutionary approaches for Multi-agent MAPF better.

The second thesis examines if weighted sum approaches are suitable to evolve subpopulations in MAPF. If the weights used to evaluate the subpopulations do not lead to optimize the objectives of the whole solution by the same weights as well as other approaches, then the other approaches should be used.

The goal of this thesis is to provide concepts for two cooperative co-evolutionary multi-objective MAPF solver: One which solves the subpopulations multi-objectively and one which solves the subpopulations with a weighted sum approach. Additionally, these algorithms should be implemented and compared against each other, using the metrics GD, IGD and by calculating the weighted sums. The weights for the weighted sums used should be the same as the ones by which the objectives of the subpopulations of the weighted sum approach are optimized. Based on the test results a statement should be given whether the experiments approve the contents of the thesis or not. Since there are infinitely many ways to conceptualize the algorithms, the thesis can not be proven or disproven for every concept of a co-evolutionary algorithm.

In chapter 2, the basics to cooperative co-evolution are explained. In chapter 3, the State of the art is described. In chapter 3.1, MAPF solvers are classified and a few of them are explained as examples. In chapter 3.2, all co-evolutionary approaches and in chapter 3.3 all multi-objective MAPF solver are summarized. In chapter 3.4, the findings of the comparison of multi-objectively and single-objectively solving single-agent pathfinding in Ahmed and Deb [1] are described. In chapter 4, the two variants of the cooperative co-evolutionary multi-objective MAPF solver are described and explained. In chapter 5, the two variants of the co-evolutionary approach are compared to each other in regards to the thesis described above. The results are analyzed to explore the limitations of the co-evolutionary approach and to find the reasons for the results. With the findings in the experiments a statement about the thesis is given. Lastly, the findings of this thesis are summarized and recommendation for future work is given in chapter 6.

1.1 Problem-description

Multiple variations of the MAPF problem exist. The MAPF problem, which is supposed to be solved by the co-evolutionary algorithm of this thesis, is similar to the definition of the classical MAPF problem explained in chapter 2.1.

The MAPF problem with *k* agents has the tuple *(G, s, t)* as input. *G = (V, E)* is an undirected graph, *s:[1, 2,. . . k-1, k] → V* assigns an agent to a starting vertex and *t : [1,2 . . . k-1, k] → V* assigns an agent to a target vertex. As simplification, time is discrete. At each time step, an agent is in a vertex and takes an action. The action *a(v)*, an agent can take is to switch to an adjacent vertex *a (v) = v'*- with $v \in V$. π_i is a single agent plan and includes all actions that move an agent *i* from the starting point *s (i)* to the end point *t (i)*. π_i *[x]* is the vertex the agent *i* is in after *x* time steps/ *x* actions. One solution $\pi = \{\pi_1 \ldots \pi_k\}$ to the entire MAPF problem is a set of *k* single-agent plans. Each single agent plan relates to one agent. The vertices are assumed to be either freely passable terrain or obstacles. Agents are not allowed to move into an obstacle. Additionally, a common simplification is used: Agents disappear after reaching the target vertex. A vertex is considered to have eight neighbors, with the simplification that each one of the eight neighbors is one time step away [cf. 38].

As stated in the introduction, the MAPF problem is usually solved single-objectively. The two most common objectives are makespan and sum of costs. Makespan is defined as the maximum number of time steps required for all agents to reach their target and sum of costs is defined as the sum of time steps each agent needs to reach its target. In this thesis, the MAPF problem is solved multi-objectively. The chosen objectives are makespan, sum of costs and the overlaps objective. All three objectives are minimized.

Usually MAPF solvers treat the conflicts as hard constraints. This solver minimizes the number of conflicts by treating them as the overlaps objective. The penalized conflicts are:

- Vertex conflict: Two agents plan to enter the same vertex at the same time step: π_i *[x]* = π_j *[x]*
- Swapping conflict: Two agents plan to swap their vertices with their action: π_i *[x + 1]* = π_j *[x]* and π_j *[x + 1]* = π_i *[x]* [38]

The objectives makespan, sum of costs and overlaps for every solution π for k agents are calculated as follows:

Makespan:

$$\text{Makespan} = max_{i=1}^{k}|\pi_i| \qquad (1.1)$$

The || operator expresses the path length

Sum of Costs:

$$\text{Sum of Costs} = \sum_{i=1}^{k}|\pi_i| \qquad (1.2)$$

The || operator expresses the path length

Overlaps:*

$C_{Swapping}(x)$ being the number of swapping conflicts at the time step *x*.

$N_{\text{Agents}}(x, v)$ being the number of agents in vertex *v* at time step *x*.

$N_{Vertices}$ being the number of vertices in V.

$$Overlaps = \sum_{x=0}^{Makespan}(C_{\text{Swapping}}(x) + \sum_{v=1}^{v=N_{\text{Vertices}}}(\frac{N_{\text{Agents}}(x,v)}{2} \bullet \left(N_{\text{Agents}}(x, v) - 1\right))) \qquad (1.3)$$

With makespan being used as maximal time step value since time and path length are unitless, discrete, and increase by the same value at each step.

This way, the calculation of the overlaps objective penalizes the clash between more than two agents exponentially to the number of agents. While a vertex conflict between two agents is penalized by a value of one, a vertex conflict between three agents is penalized by a value of three. A conflict is always penalized for each agent, while the penalty increases by a value of 0.5 for each agent one agent collides with.

2 Theoretical Background

In this chapter, the theoretical foundations for the classical multi-agent pathfinding problem and cooperative co-evolution are explained. The description of the classical MAPF in chapter 2.1 problem highlights the differences between classical MAPF and the problem description in this thesis. In chapter 2.2 cooperative co-evolution is explained, since the algorithm used in this work is based on these theoretical foundations.

2.1 Multi Agent Pathfinding (MAPF)

The classical MAPF problem described in Stern et al. [38], is similar to the MAPF problem described in chapter 1.1. In this chapter, the differences between the classic MAPF described in Stern et al. and used in this work are highlighted. In the classical MAPF problem, agents have two action types: The move action $a(v) = v'$ and the wait action $a(v) = v$. Another important difference is that a conflict between two agents in the plans makes a MAPF solution invalid.

Other than the two conflicts used in this thesis, there are other conflicts used in literature. The most commonly used ones are:

- The Vertex conflict: Two agents plan to enter the same vertex at the same time step.
- The Edge conflict: At the same time step, two agents plan to traverse the same edge into the same direction.
- The Following conflict: One agent occupies a vertex, which was occupied by an agent exactly one time step before.
- The Cycle conflict: A set of agents form a rotating cycle pattern. This occurs, if every agent enters the vertex the next agent was occupying the time step before, whereby the last agent occupies the vertex the first agent was occupying at the previous time step.
- The Swapping conflict: Two Agents plan to swap their vertices with their action.

Figure 1 illustrates the described conflicts.

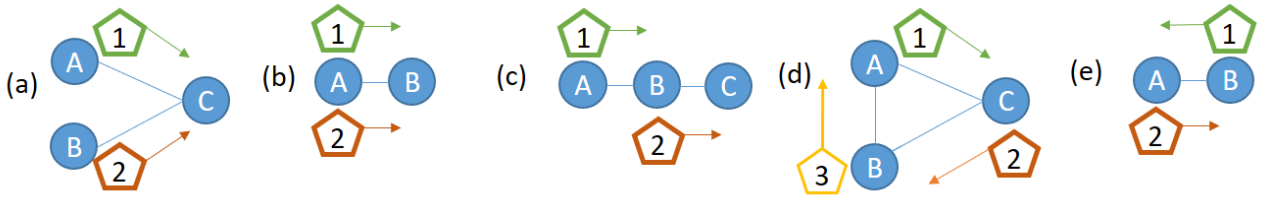

Figure 1: MAPF conflicts

(a) Vertex conflict, (b) Edge conflict, (c) Following conflict, (d) Cycle conflict, (e) Swapping conflict. [cf. 38]

There are two options for agent behavior at the target vertex. Either the agent is considered to have disappeared, or the agents waits in its target vertex until all agents have reached their designated target vertex.

More specialized MAPF problem formulations differentiate from the classical MAPF:

- In the weighting of the graph: Either agents move from one grid to a neighboring one, whereby there can be distinctions between the number of neighboring vertices, or they move in Euclidean space.
- In the applications of Feasibility Rules: Solutions are only feasible if they fulfill some Robustness Rules or if the agents move in formations.
- In transforming the classical problem into a motion planning problem, where agents can have volumes, shapes or move at changing speed.
- And in assigning the task of agents: In some MAPF problems the target vertices have also to be assigned to the agents [38].

2.2 Cooperative Coevolution

Co-evolutional algorithms try to exploit the compositional nature of a problem by using a genetic algorithm for each of their components. Co-evolutional genetic algorithms can be categorized into competitive and cooperative co-evolutional algorithms [45]. In this thesis, a cooperative co-evolutionary algorithm (CCEA) according to Potter and Jong [31] was implemented. In Potter and Jong five ideas how such a cooperative co-evolution approach has to look like:

- A species (subpopulation) is a subcomponent of a solution.
- The complete solutions are obtained by combining representative individuals of each subpopulations.

- The fitness at the species level depend on the fitness of the complete solution it participates in.
- The subpopulations evolve by themselves if necessary.
- The evolution of a species is handled by a genetic algorithm (GA) [31].

Since co-evolutionary algorithms have been around for a long time, further analyzes about them were done. In Wiegand [44] it was shown that normal CCEAs have problems finding global optima. This is because they tend to test the fitness of individuals of subpopulations, which might lead to an optimal solution, with individuals of other subpopulations they do not fit to. On the other hand, co-evolutionary algorithms tend to lead to more robust solutions since they are evaluated on how well they fit together with other solutions [3,45].

In Panait et al. [29] and in Bucci and Pollack [3] approaches were formulated, which help the CCEA to find the global optimum. In Panait et al. [29] it was suggested that an estimation of the optimal values should be mixed into the evaluation. In Bucci and Pollack [3] three different setups were compared against each other to find out which CCEA versions get near the optima. The first version is a classic CCEA, which evaluates an individual by the best solution of the previous generation. The second version the pCCEA considers every solution of the previous generation as an objective. Every individual is evaluated multi-objectively by these objectives. The third version the cCCEA was also tested on every solution but numerical fitness values are used. The experiments showed that the pCCEA and the cCCEA works far better than the standard CCEA version. Ultimately, the pCCEA turned out to work best out of the three version.

3 State of the Art

In this chapter, the state of the art of MAPF solver will be discussed in more detail. In chapter 3.1, a small overview and classification of MAPF solver is given. In chapter 3.2, all cooperative co-evolutionary algorithms found in literature are described and discussed. A few features of these concept are adopted by the approach presented in this thesis. In chapter 3.3, all multi-objective MAPF approaches found in literature are presented and the differences between these approaches and the one used in this thesis are explained. In chapter 3.4, the findings in Ahmed and Deb [1] are summarised. These findings give insight why multi-objective optimization might be better than single-objective optimization in single-agent pathfinding and are helpful for the theses of this work. In chapter 3.5, the state of the art chapter is summarized.

3.1 Overview classic MAPF Solver

MAPF solver can be categorized into distributed [16] and centralized settings. In distributed settings, the agents make their decisions on their own. The algorithm of this thesis falls into the centralized setting category. In this setting, the movements of the agents are controlled by a single decision maker [10]. Additionally MAPF solver can be divided into the categories traditional methods and intelligent methods [32]. The traditional MAPF solvers can further be categorized in whether they solve the problem optimal or suboptimal [10].

To the optimal solvers category belong:

- Reduction-based Optimal Solvers: Reduction-based Optimal Solvers reduce the MAPF problem to a well-known problem, for which optimal solvers exist already. In the approach in Surynek [39] for example the MAPF problem is translated into Boolean variables. This way a SAT formula is created, which SAT solvers can solve.
- A*-based Optimal Solvers: A*-based Optimal Solvers solve the MAPF like a normal A* problem using a joint state space, where the agents are viewed as a single entity. How fast the algorithm converges depends on the used heuristics [6].
- Increasing Cost Tree Search: In Increasing Cost Tree Search, the MAPF problem is solved using the Increasing Cost Tree. The root of the tree represents all the possible solutions where the agents' routes are the fastest. A child will increase the length of the route of an agent by one. Starting with the root once a node holds a valid solution the search is finished [10].
- Conflict based search (CBS): Conflict based search (CBS) uses a binary constraint tree to find an optimal solution. Every node of the tree inherits the constrains from the node above. A

solution is created which fulfills the constrains of a leave. If the solution is invalid, the leave creates new nodes, which prohibit the action, which made the solution invalid. The nodes with the lowest costs are examined first [10].

For each one of these solvers exist different variants and enhancements. More details can be found in Felner et al. [10].

The category of suboptimal solvers include search-based suboptimal solvers. They tend to be not extremely fast but near optimal [10]. Hierarchical Cooperative A* for example creates an order in which the agents plan their path to the target. The path is reserved and the following agents are not allowed to have a conflict with reserved paths [34]. Bounded suboptimal solvers are also part of the search-based suboptimal solvers. They let the user choose a value w, which leads to a search where a solution is guaranteed to cost less than $w \bullet C$, where C is the cost of the optimal solution [2]. Another category of suboptimal solvers is the Rule-based solvers. They usually find a solution relatively fast, but return solutions that are often not near the optimal solution. Many of them are modified optimal MAPF solvers like the Greedy CBS which is a CBS-based MAPF solver designed for finding a solution as fast as possible [2]. Some approaches fit as hybrids into both categories [10].
The co-evolutionary approach of this thesis falls into the category of the intelligent MAPF solvers. Other intelligent approaches for MAPF would be the virtual spring method [28], the artificial bee colony for online MAPF in Liang and Lee [21], the evolutionary algorithms [25] and the switching formation strategy for multi-robots in Dai et al. [7].

3.2 Multi Robot Path planning using cooperative co-evolutionary Algorithms

The cooperative co-evolutionary adaptive genetic algorithm (CCAGA) is the oldest co-evolutionary solver for multi robot path planning in literature. Agents have their own subpopulation, in which the individuals evolve. It is a simple algorithm, which shrinks the decision-space by only allowing waypoints at the edges of obstacles. Agents' cooperation is forced via their evaluation. Individuals of one subpopulation are evaluated by pairing them with the best individuals of the other subpopulation and evaluating them as one solution to the whole problem. The approach allows infeasible solutions and tries to sort them out by penalizing them [4].
In Chen [5] each agent has its own population and uses a chaotic genetic algorithm. The algorithm uses a weighted sum approach to optimize the objectives path length, smoothness and time with penalties for having too much velocity, running into obstacles or conflicting with the best individual

of the other agents. The mutation and the crossover rates are adaptive. The algorithm uses a chaos operator, which disturbs individuals. If they result in a better individual, they replace the old one.

In Kala [18] the subpopulations are divided in populations of the slave genetic algorithm (SA) and a subpopulation of the master genetic algorithm (MA). Every subpopulation of the slave algorithm optimizes the path of a single agent with an evolutionary algorithm. An individual of the master algorithms' population picks individuals out of the subpopulation of the slave algorithm to create a MAPF solution. The master algorithm evolves its population also evolutionary. Individuals of the SA cooperate with each other using their evaluation. An individual of the SA is evaluated by its path length and by how well it would work together with the other agents. The second part of the evaluation is done by calculating the sum of all collisions that an individual i of an agent j would have if it replaces the current picked solution of the agent j in the five best master algorithm solutions. The genes of a slave algorithms' individuals are represented by an integer, which defines how it will react at the next crossing. A crossing is defined as a place where the agent has more move options than following the path or turning around. The first gene decides the direction the individual takes at the first crossing; the nth gene decides the direction the individual takes at the nth crossing. Once all genes are used up, while the individual did not reach the endpoint, it resets once and starts using the genes again, culminating into the fact that an agent does not always end at the designated target. Additionally, the algorithm uses a scattered crossover, which will rip paths apart. The algorithm was only tested on maze-like maps, which heavily benefits the action-at-next-crossing representation [18]. The approach of Kala [18] was mentioned in Pan et al. [28] where the author wrote that the algorithm of kala suffers "of slow convergence rate, local optimum and ignoring cooperation between populations". It is to assume that the representation and the operators of the SA are more likely to be responsible for these shortcomings than the general structure of the algorithm.

The approach in Qu et al. [32] has a subpopulation for every agent ,which optimizes the weighted sum of the objectives path length, safety and smoothness for the associated agent. Waypoints with straight paths between them are used. The paper presents a modification operator, which deletes waypoints between two waypoints either to avoid obstacles or to accelerate convergence. The algorithm selects the representatives for each subpopulation by optimality. The cooperation between the agents happens after that, using the island approach: Every agent randomly selects another agent. A second fitness value for every representative regarding the collision with every representative of the other agent is calculated. According to this fitness value, the elite individual of this agent for this generation is selected. The elite individuals form the final solution.

In Korayem et al. [20] the concept of a master and a slave algorithm was used too. In this case the slave algorithm solved the individual pathfinding problem for each agent (here nanoparticles). However, the premise of the problem is different: The nanoparticles do not move simultaneously but in sequence and task allocation was done as well. The slave algorithm uses a weighted sum approach to optimize three different objectives as one. Additionally, an artificial potential field was used to repair infeasible solutions in the SA to find better routes.

The algorithm in Muthiah and Saad [24] uses an order similar to Hierarchical Cooperative A* in which the routes of the algorithm are determined. The order is defined by the size of the decision space. The following agents see the routes of the agents before as moving obstacles. A genetic algorithm determines the routes. Additionally, some heuristics are used, which are not fully explained. The author claims that the found path of the genetic algorithms is the "optimal path", which is due to the nature of genetic algorithms questionable. Then again, the text found was only an extended abstract of a thesis, which could not be accessed.

In Sarkar et al. [33] a co-evolutionary algorithm is presented where the agents perform a multi-agent pathfinding task while having multiple targets without a predefined order. The agents evolve in their own population with their fitness function being the path length. At the end of a generation, every individual of one population is paired with every individual of the other populations to calculate their interaction cost. With the sum of the interaction cost between every combination of individuals between agents and their path lengths, the elite set of chromosomes is formed. For the representation of the individual paths, waypoints are used which are connected with a straight path. The GA also uses a deletion operator, which deletes waypoints and replaces them sometimes with other nodes to minimize the path length.

This concludes the co-evolutionary algorithms found in literature. The common ground between these approaches is that every agent has their own population. The main differences can be found in the cooperation between the populations, in the picking of the overall solution and in the evolution of the subpopulations, with cooperation and picking of an overall solution being distinctive features of a co-evolutionary algorithm.
The term cooperation here means how collisions with other robots are avoided. In Muthiah and Saad [24] the cooperation is done by seeing the paths of over agents as moving obstacles. In all the other texts cooperation is done by the fitness function. In Kala [18], in Cai and Peng [4] and in Chen et al. [5] the conflicts are part of the fitness function with which the subpopulation is evolved by. In Sarkar et al. [33] and in Qu et al. [32] the subpopulations are evolved by a fitness function, which does not

consider other agents. The cooperation of these two approaches is done at the end of each generation by a second fitness function, which selects individuals based on their conflict behavior with the other subpopulations. In Sarkar et al. [33] all individuals are compared with each other while in Qu et al. [32] the representatives of an individual only communicate with the representatives of one other agent instead of all agents. In Cai and Peng [4] and in Chen et al. [5] the cooperation is done only with the best combination of solutions and in Kala [18] the cooperation is done with the y best combinations of individuals.

The picking of the best combination in Sarkar et al. [33] and in Qu et al. [32] is done by the results of second fitness function. The picking of the best combination in Cai and Peng [4] and in Chen et al. [5] is just selecting the best individuals. In Kala [18] another genetic algorithm, the master algorithm picks the best individuals.

3.3 Multi-objective Optimization of MAPF with Genetic Algorithm

In Literature just two paper exist on the topic of solving MAPF multi-objectively:

In Weise et al. [42] the MAPF problem is solved multi-objectively with an evolutionary algorithm. Individuals of the algorithm contain paths for every agent, which consist of an array of a fixed number of waypoints, which are translated into nodes using the Dijkstra algorithm. The crossover operator exchanges a random waypoint of one random agents from two individuals with each other and the mutation operator changes the position of a random agents‘ waypoint by one node. The optimized objectives are makespan, flow-time, which is the same as sum of costs, and the overlaps objective. The overlaps objective counts all edge conflicts and vertex conflicts of the given solution. Selection is done with the NSGA-II and NSGA-III operators.

In Mai and Mostaghim [23] another multi-objective multi-agent pathfinding approach for swarm robotics was formulated. The algorithm focuses on decentralized path planning. Since the approach is done for the swarm robotics, a motion-based problem is formulated instead of the theoretical classic MAPF problem. A remodelled NSGA-II approach is used, where the individuals contain a path for every agent made out of waypoints. The crossover is a two-point crossover between a pair of randomly selected agents. For mutation a Gaussian mutation, which moves one of the waypoints of a randomly selected agent in its neighbourhood, is used. Additionally, a smoothing operator, which selects a waypoint of an randomly selected agent and puts it on the path between its previous waypoint and it‘s subsequent waypoint, is used. The optimized objectives are the risk objective, which minimizes collision potential with obstacles and other agents, and path length, which tries to minimize all the paths.

Other than these two, no paper about solving the multi-agent pathfinding problem multi-objectively in literature exists as far as the literature search in this thesis goes. Often in multi agent path planning multiple objectives are optimized by using the weighted sum approach:

For example in Chen et al. [5] the weighted sum for each subpopulation for the objectives distance, time and smoothness with penalties for constraint violation was optimized. Also in Qu et al. [32] objectives path length, safety and smoothness are optimized with the weighted sum approach.

The approach in this thesis aims to solve the MAPF problem by optimizing the objectives makespan, sum of costs and overlaps by solving the problem multi-objectively by creating a set of pareto-optimal solutions regarding the three objectives. With this, the thesis differentiates itself from the weighted sum approaches. The algorithm in this thesis differentiates itself from Weise et al. [42] in the following points: The overlaps objective of this thesis minimizes vertex and swapping conflicts instead of vertex and edge conflicts. Forbidding vertex conflicts implies that edge conflicts are also forbidden [38]. In that sense, penalizing vertex conflicts means that edge conflicts are also penalized. Additionally the overlaps objective in this paper penalizes a vertex conflict between more than one agents exponentially. Furthermore, a co-evolutionary algorithm is used in this thesis. Other than that by not fixing the number of waypoints, the approach in this thesis is able to explore the whole decision space. Other than by using a co-evolutionary approach and optimizing different objectives the approach in this thesis also differentiates itself from Mai and Mostaghim [23] by attempting to solve a problem near to the classical MAPF problem.

3.4 Single-objective Optimization in Comparison to Multi-objective Optimization in Pathfinding

In Ahmed and Deb [1] a comparison between a single objective optimizing EA and a multi-objective optimizing EA with NSGA-II in the context of singe agent pathfinding is done. The optimized objectives are: the Path length, the path vulnerability, which maximizes the distance to obstacles, and the smoothness, which sums up the angle of each turn. In the tests, the one single objective optimizer optimized path length. Another single objective optimizer optimized the path vulnerability. For the multi-objective optimizer existed a version which was optimizing path length and path vulnerability while using smoothness to keep diversity instead of the crowding distance. The second multi-objective optimizer optimized all three objectives.

The results on a less complex environment with a grid size of 16 x 16 and randomly placed obstacles showed that the extreme values of the multi-objective optimizer were approximately the same as the results of the single objective optimizers. On a larger more complex environment a 64x64 grid with

a higher obstacle density, the single objective solver was not able to find a feasible solution anymore neither with the path length objective nor with the path vulnerability objective. The multi-objective solver with three objectives on the other hand was able to find feasible solutions in 80% of the time. Ahmed and Deb interpreted that these results must have occurred because the multi-objective EA is able to keep a high diversity in the population while a single objective EA is more likely to lose this diversity [1].

3.5 Summary of the state of the Art and discussion

In this chapter the MAPF solver are categorized and a few of them are described. Furthermore, all the co-evolutionary MAPF solver as well as all approaches which solve the MAPF problem multi-objectively were explained. Additionally the findings of a paper, which examined the differences between solving the single agent pathfinding problem single objectively and multi-objectively, were described.

Most of the co-evolutionary approaches used the fitness function to cooperate between agents while one approach was searching for the paths of the agents by order. The "by order" approach in Muthiah and Saad [24] does not follow the guidelines of Potter and Jong [31] and since the first agents do not care about the following agents, they will likely end in local optima. The two approaches, which used one fitness function without communication to the other subpopulation to evolve the subpopulation and second fitness function with communication to find the best combination [32,33], are most likely to end up in local optima too, since the subpopulations evolve without knowing of the other populations. In addition, the idea of just using the best-known solution to avoid collision might lead to bad performance [32,33]. The findings in Bucci and Pollack [3] which were described in chapter 2.2, showed that to minimize the chance of falling into local optima all available solutions should be used to evaluate the individuals. While all solutions might be too much since the computational time needed might be too high, at least the x best solutions should be used.

Since the problem of the thesis is to solve MAPF multi-objectively, the Master Slave architecture of Kala [18] seems like a good way to deal with the problem. The Master algorithm can pick the best solutions to solve the multi-objective MAPF problem. The Slave algorithm must evaluate the paths also by the objectives of the Master algorithm. All the co-evolutionary approaches, which were optimizing more than one objective, used the weighted sum approach to find the best solutions [5,20,32]. Since the studies in Ahmed and Deb [1] showed that solving the single agent path-planning problem multi-objectively leads to better solutions, this thesis will examine how well a co-

evolutionary algorithm, which uses a weighted sum approach for the subpopulation, does compared to a co-evolutionary algorithm, which solves the subpopulation multi-objectively.

Many approaches used waypoints to represent paths. While most of them use a straight line from waypoint to waypoint, the approach in Weise et al. [42] uses the Dijkstra algorithm. A straight line is easier to compute but will lead to infeasible solutions. This is especially the case, if the environment is maze-like. Because of that, this thesis will also use a shortest path algorithm like Dijkstra. The waypoint approaches also used deletion operators to fasten the convergence, which will be adapted by the approach of this work too.

The research on the multi-objective MAPF problems also showed that no co-evolutionary approach was yet used to solve the problem. Additionally the algorithm presented differentiates itself in more aspects like using the whole decision space.

4 Materials and Methods

In this chapter the concept of the two variants of the co-evolutionary multi-objective MAPF solver are explained and their implementation is described. In chapter 4.1 the basic idea of the algorithm is presented. In chapter 4.2 an overview of the functions of the algorithm is given. In chapter 4.3 the functions and their implementation are explained in more detail.

4.1 Architecture of the algorithm:

In order to reduce the decision space, the problem is decomposed by using a co-evolutionary algorithm. An evolutionary algorithm – slave algorithm (SA) - optimizes the route of each agent. Another evolutionary algorithm – the master algorithm (MA) - searches for solutions for the multi-objective MAPF problem by combining the best solutions of the subpopulations, which are optimized by the slave algorithm. These best solutions are the representatives of the subpopulations.
The MA optimizes the objectives makespan, sum of costs and overlaps, which are fully explained in the problem description in chapter 1.1. The SA minimizes the objectives path length and collision count. The path length (*PL)* objective stands for the number of vertices passed and the collision count represents the collisions between an individual and the representatives of the other agents' subpopulations.
For all individuals *i* the path length *PL* is the length of the single agent plan π_i of individual *i*:

$$PL_i = |\pi_i| \qquad (4.1)$$

The || operator expresses here the path length

The collision count CLC_i objective of an individual *i* of the subpopulation of an agent *A* penalizes $individual_{i,A}$ for having vertex conflicts with representatives of the other agents with one and penalizes $individual_{i,A}$ for having possible swapping conflicts with the representatives of the other agents by the Collision-Swapping-Penalty-Value: C_{SPV}. A possible swapping conflict occurs for each representative of the other agents which is at the same vertex v at time step *x+1* as the $individual_{i,A}$ is at time step *x*. This way, agents are penalized for each agent they might have had a swapping conflict with.

The collision count CLC_i for an individual i is described by equation 4.2:

$$\delta(\mathrm{x})=\begin{cases}1, x = 0\\0, x \neq 1\end{cases}$$

$$CLC_i = \sum_{x=0}^{|\pi_i|}\sum_{j=1,i\neq j}^{k}\sum_{r=1}^{\mathrm{N_R}}(\delta(\pi_i(\mathrm{x}) - \pi_{jr}(\mathrm{x})) + C_{\mathrm{SPV}} \bullet \delta(\pi_i(\mathrm{x}) - \pi_{jr}(\mathrm{x}+1)) \quad (4.2)$$

With N_R being the number of representatives every agent and $|\pi_i|$being used as maximal time step value since time and path length are unitless, discrete, and increase by the same value at each step.

The optimization of the path length objective in the SA directly optimizes the MA objectives makespan and sum of costs. The optimization of the collision count objective in the SA is supposed to optimize the overlaps objective in the MA.
The subpopulation of the agents cooperate via the collision count objective. Optimizing this objective should minimize conflicts with other agents. Two versions of the SA are implemented in this algorithm. One which uses a weighted sum approach to evaluate individuals and another one which uses a non-dominated sorting algorithm like described in Deb et al. [8] to select a set of pareto optimal solutions.
For better understanding, the single-objective SA is abbreviated to SO SA and the multi-objective SA to MO SA. The cooperative co-evolutional genetic algorithm using the SO SA is called SO SACCGA and the cooperative co-evolutional genetic algorithm using the MO SA is called MO SACCGA

4.2 Algorithm Overview

This chapter gives an overall overview of the algorithm. The individual functions are explained in detail in the following chapters.

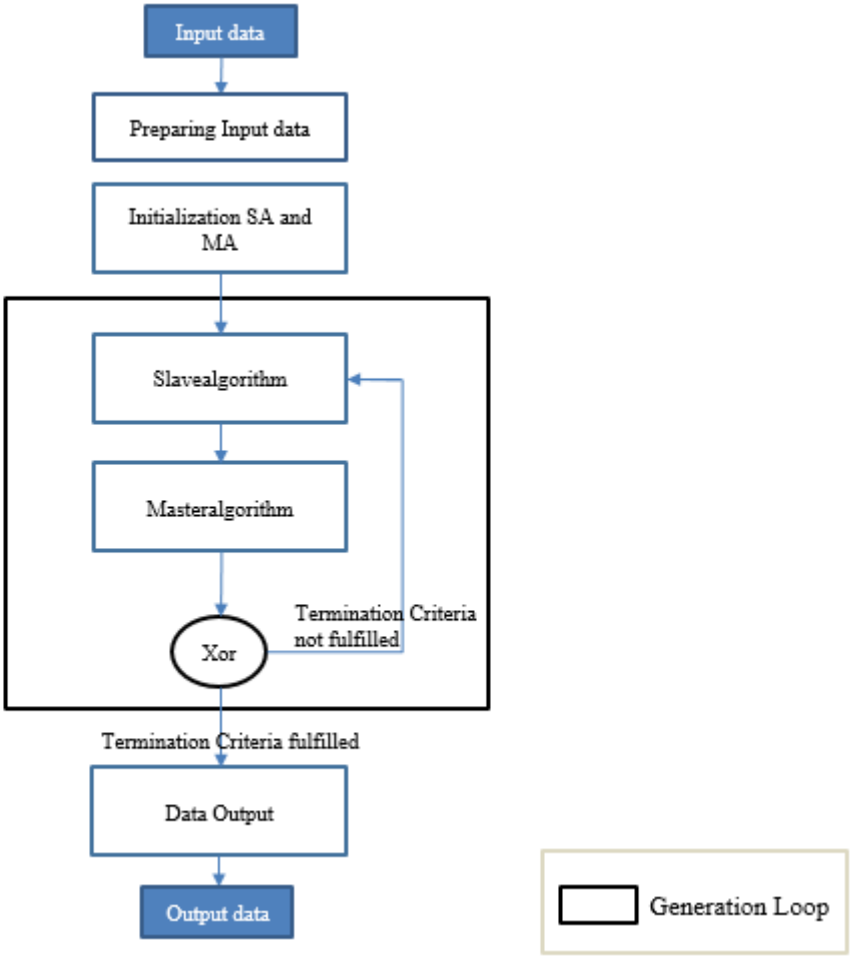

Figure 2: Overview of the algorithm architecture

Figure 2 shows the basic structure of the algorithm. At the beginning of the algorithm, input data is imported. The input data contains data on the problem such as map layout and number of agents start- and target point and setting parameters (chapter 4.3.4). First, the input data is being prepared. Secondly, the first individuals of the first generation of the SA subpopulations and the MA subpopulation are initialized and partially evaluated (chapter 4.3.5).

After that the first generation of the algorithm starts. Within one generation, one generation of the SA for the subpopulation of each agent is executed. Then, one generation of the MA is performed. The results are then saved and a check is carried out to determine whether the termination criterion is met. If it is fulfilled, the results are put out and the algorithm terminates. If not, the next generation begins.

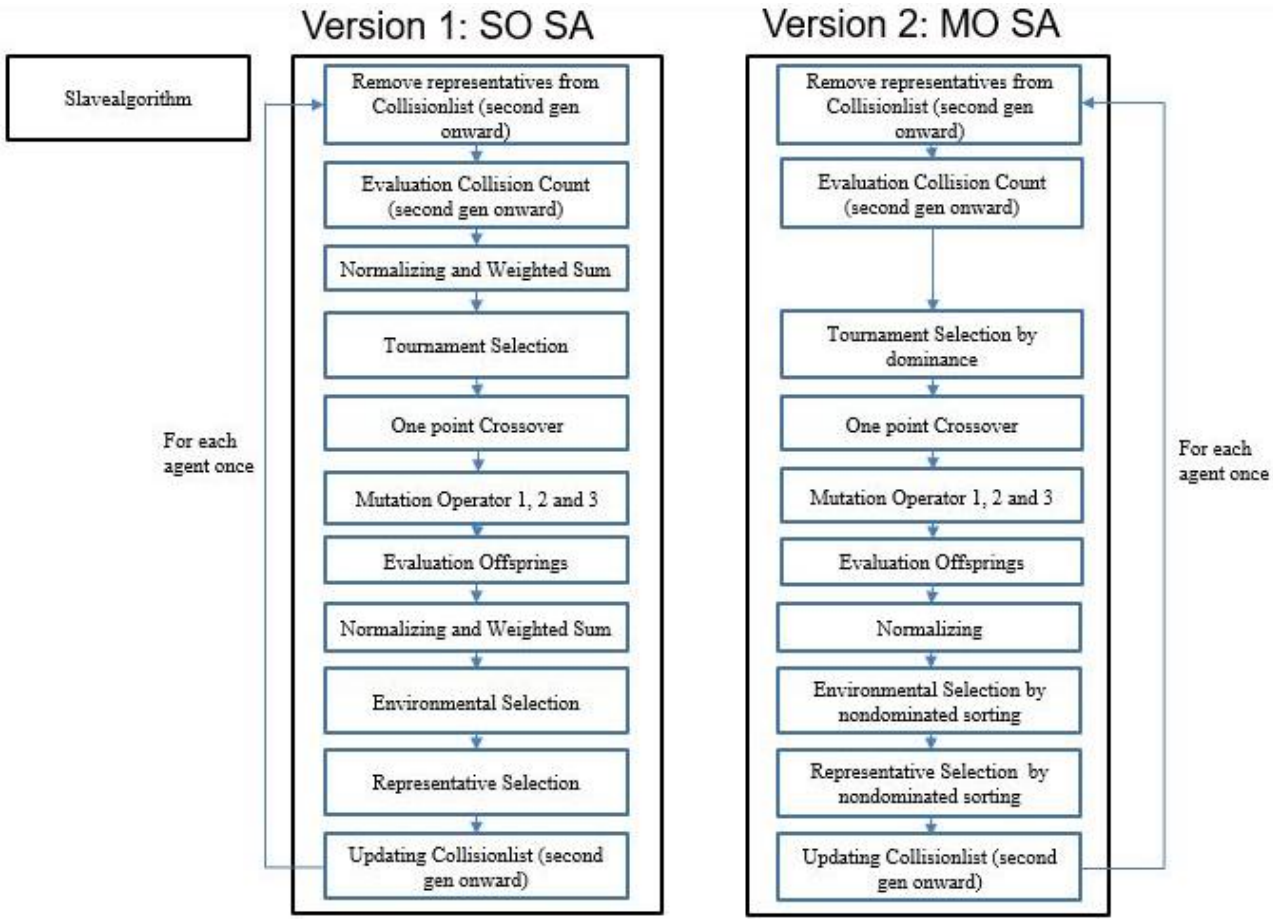

Figure 3: Sequence of slave algorithm functions of the SO SA and MO SA

Figure 3 shows how the slave algorithm works. A pseudocode of the MO SA is presented in algorithm 1 and of SO SA in algorithm 2. The parts of the MO SA and the SO SA, which are different from another, are highlighted in both algorithms. Every agent A has their own population $SAPop_A$, which consists of Individuals - $SAPop_A$={$Individual_{1,A}$,...,$Individual_{N_{SA\ Individuals}}$} (with $N_{SA\ Individiuals}$ being the size of the subpopulation of the SA) and their fitness values for the objective collision count $Fit_{CLC,Individual_{i,A}}$ and the objective path length $Fit_{PL,Indidvidual_{i,A}}$, which are mapped to the corresponding individual. Additionally, every agent has their own representatives R_A={r_1 ,...,r_{N_R}}R_A ⊆ $SAPop_A$. The first thing the SA does and only after the first generation is to update the Fit_{CLC} value of all the individuals. This must be done because the Fit_{CLC} value of the individuals changes by changing representatives of the other agents. For the evaluation of the collision count objective the $Collisionlist_{VX}$ is used. The $Collisionlist_{VX}$ is a two dimensional matrix with the dimensions vertices V and time steps X. A cell C_{vx} of the $Collisionlist_{VX}$ holds the information how many representatives of all agents are at vertex v at time step x.

$$C_{VX} = \sum_{j=1}^{k}\sum_{r=1}^{N_R}(\delta(\pi_{jr}(x) - v)) \quad (4.3)$$

Before the Fit_{CLC} can be evaluated the information of the own agents $representatives_A$ must be removed from the $Collisionlist_{VX}$. Next, the parents are selected. In the multi-objective SA the parents are selected by tournament selection with dominance as selection criteria. The SO SA has to normalize the fitness-values and calculate the weighted sums first. Then the parents are selected by tournament selection with weighted sums being the selection criteria. After this, in both versions the parents are used for crossover with the SA crossover probability (*SACXPB)* as probability. The parents, which are not affected by crossover, are simply copied into the offspring. Then the offspring are mutated by three different mutation operators each one having their own mutation probability (*SA MUTBP1, SA MUTBP2, SA MUTBP3)*. If a copied parent is not affected by any changes, it will be deleted. Next the Fit_{CLC} and the Fit_{PL} values of the offspring are determined and the next generation is selected. For the MO SA, the fitness-values are normalized and the non-dominated-sorting-selection like in Deb et al. [8] is used to determine the next population and the new representatives of the agent. For the SO SA the fitness-values are normalized, the weighted sum is calculated and the new population is formed by the best individuals of the old population and the newly created offspring. The individuals with the highest weighted sum values form the representatives of the agent. Lastly, the information of the new representatives is added to the $Collisionlist_{VX}$.

Algorithm 1: Slave algorithm Multi-objective (MO SA)

Input: Population of the agent A*:* $SAPop_A$, Matrix to calculate collision count: $Collisionlist_{VX}$,representatives of agent *A:* R_A, the fitness values for the individuals of $SAPop_A$: $Fit_{CLC,SAPop_{A_{,,}}}$, $Fit_{\mathrm{PL},SAPop_A}$, the counter of the generation: *gen*, SA crossover-probability: *SACXPB*, SA mutation-probabilities: *SAMUTBP1, SAMUTBP2, SAMUTBP3*

Output: New_SAPop_A, New_R_A

for every Agent:

| **if** gen $>$ 1:

| | $Collisionlist_{VX}$= removeRepresentativesFromCollisionlist($Collisionlist_{VX}$, R_A)

| | $Fit_{CLC,SAPop_A}$ =evaluateCLC($SAPop_A$, $Collisionlist_{VX}$)

| *Offspring* =tournamentselectionByDominance($SAPop_A$)

| **for** $Offspring_i$, $Offspring_{i+1}$in *Offspring*:

| | **if** *SACXPB* <random(0,1):

| | | $Offspring_i$, $Offspring_{i+1}$=crossoverSA($Offspring_i$,$Offspring_{i+1}$)

| **for** $Offspring_i$ in *Offspring*:

| | **if** *SAMUTBP1*< random(0,1):

| | | $Offspring_i$=mutationGeneDeletion($Offspring_i$)

| | **if** *SAMUTBP2*< random(0,1):

| | | $Offspring_i$=mutationShiftInNeighbourhood($Offspring_i$)

| | **if** *SAMUTBP3* < random(0,1):

| | | $Offspring_i$=mutationInsertRandomWaypoint($Offspring_i$)

| | **if** $Offspring_i \subseteq SAPop_A$: # copied parents which weren't changed are deleted

| | | delete $Offspring_i$

| | $Fit_{\mathrm{CLC},Offsprings}$,$Fit_{\mathrm{PL},Offsprings}$=evaluateCLC&PL(*Offspring*, $Collisionlist_{VX}$)

| $SAPop_A$=$SAPop_A$ ∪ *Offspring*

| $norm\_Fit_{CLC,SAPop_A}$,$norm\_Fit_{\mathrm{PL},SAPop_A}$=normalizeFitness($Fit_{\mathrm{CLC},SAPop_A}$,$Fit_{PL,SAPop_A}$)

| New_SAPop_A=nonDominatedSortingSelection($SAPop_A$, $N_{\mathrm{SA\ Individuals}}$)

| New_R_A = nonDominatedSortingSelection (New_SAPop_A, N_{R})

| $Collisionlist_{VX}$=updateCollisionlist($Collisionlist_{VX}$, New_R_A)

return New_SAPop_A, New_R_A

Algorithm 2: Slave algorithm Singe-objective (SO SA)

Input: Population of the agent A*:* $SAPop_A$, Matrix to calculate collision count: $Collisionlist_{VX}$,representatives of agent *A:* R_A, the fitness values for the individuals of $SAPop_A$: $Fit_{CLC,SAPop_A}$, $Fit_{\mathrm{PL},SAPop_A}$, the current generation: *gen*, SA crossover-probability: *SACXPB*, SA mutation-probabilities: *SAMUTBP1, SAMUTBP2, SAMUTBP3*

Output: New_SAPop_A, New_R_A

for every Agent:

| **if** gen > 1:

| | $Collisionlist_{VX}$= removeRepresentativesFromCollisionlist($Collisionlist_{VX}$, R_A)

| | $Fit_{\mathrm{CLC},SAPop_A}$ =evaluateCLC($SAPop_A$, $Collisionlist_{VX}$)

| | $normWeightedSum_{SAPop_A}$=normalize&FormWeightedSum($Fit_{\mathrm{CLC},SAPop_A}$,$Fit_{\mathrm{PL},SAPop_A}$)

| *Offspring*=tournamentSelectionByWeightedSum($SAPop_A$, $normWeightedSum_{SAPop_A}$)

| **for** $Offspring_i$, $Offspring_{i+1}$in *Offspring*:

| | **if** *SACXPB* <random(0,1):

| | | $Offspring_i$, $Offspring_{i+1}$=crossoverSA($Offspring_i$,$Offspring_{i+1}$)

| **for** $Offspring_i$ in *Offspring*:

| | **if** *SAMUTBP1*< random(0,1):

| | | $Offspring_i$=mutationGeneDeletion($Offspring_i$)

| | **if** *SAMUTBP2*< random(0,1):

| | | $Offspring_i$=mutationShiftInNeighbourhood($Offspring_i$)

| | **if** *SAMUTBP3* < random(0,1):

| | | $Offspring_i$=mutationInsertRandomWaypoint($Offspring_i$)

| | **if** $Offspring_i \subseteq SAPop_A$: # copied parents which weren't changed are deleted

| | | delete $Offspring_i$

| | $Fit_{\mathrm{CLC},Offsprings}$, $Fit_{\mathrm{PL},Offsprings}$=evaluateCLC&PL(*Offspring*, $Collisionlist_{VX}$)

| $SAPop_A$=$SAPop_A$ ∪ *Offspring*

| $normWeightedSum_{SAPop_A}$= normalize&FormWeightedSum($Fit_{\mathrm{CLC},,SAPop_A}$,$Fit_{\mathrm{PL},,SAPop_A}$)

| New_SAPop_A=environmentalSelection($SAPop_A$, $normWeightedSum_{SAPop_A}$, $N_{\text{SA Individiuals}}$)

| New_R_A=representativeSelection(new_$SAPop_A$, N_{R})

| $Collisionlist_{VX}$=updateCollisionlist($Collisionlist_{VX}$, New_R_A)

return New_SAPop_A, New_R_A

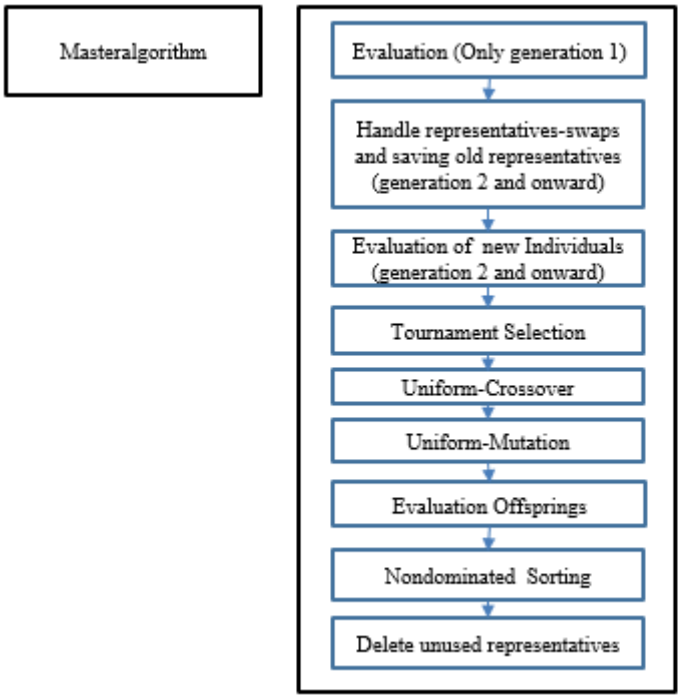

Figure 4: Sequence of master algorithm functions

Figure 4 shows the process of the MA. A pseudocode of the MA is presented in Algorithm 3. The MA has its own population $MAPop = \{Individual_{MA,1} \ldots Individual_{MA,N_{\text{MA Indiviudals}}}\}$ with $N_{\text{MA Indiviudals}}$ being the numbers of individuals in the MA subpopulation. Every Individual $Individual_{\text{MA},i}=\{Gene_1 \ldots Gene_k\}$ has a Fitnesstuple = $(Fit_{Makespan}, Fit_{SumOfCost}, Fit_{Overlaps})$ for the fitness-values of the three objectives makespan, sum of costs and overlaps and consists of exactly as many genes as there are agents. The value of a gene refers towards one of the representatives of the agent. The representatives, which are referred to, can be of the current generation or of the generations before. In generation one, the first thing the MA does is to evaluate the newly initialized population. This can only be done after the first iteration of the SAs, since they do not have any representatives before. After that, the MA handles the changes in the representatives. This is only done at the second generation and onwards. The individuals, which refer to changed representatives, keep pointing at the old representative. An additional version of the individual is created, which points at the new representative, which took the spot of the old representative. A change in representatives occurs if the SA finds a better individual to fill that spot. The old representatives, which are still used by the MA, are saved. The newly created individuals are evaluated and added to the population. Next, the parents are selected by tournament-selection with dominance as selection criteria. The parents are either used for crossover or copied into the offspring depending on the crossover probability. The used crossover-operator is the uniform-crossover-operator. With the chance of the mutation probability, the offspring are mutated. After that, the offspring are evaluated. The non-dominated-

sorting algorithm (like used in NSGA-II) is used to pick the best individuals from the old generation and the offspring to form the new population and to determine the set of pareto dominant solutions. Lastly, the saved old representatives, which are not used anymore, are deleted. The MA returns its population, the saved old representatives and the found pareto front.

Algorithm 3: Master algorithm (MA)

Input: Population of the MA: *MAPop*. representatives of all agents of the current generation: $R_k = \{\sum_{A=1}^{k}\sum_{r=1}^{Number\ of\ representatives} R_{A,r}\}$, representatives of all agents the generation before the current generation: old_R_K, list of saved representatives from old generations: $Old\_Representatives_R$, current generation: *gen,* crossover-probability*: MACXPB,* mutation-probability*: MAMUTBP*

Output: *MAPop, Pareto_Front*

if *gen*=0:

| $Fitness_{MAPop}$ = evaluateMA(*MAPop*,R_k)

if *gen*>0:

| *New_Individuals*, $Old\_Representatives_R$= handlingChangesOfRepresentatives(*MAPop*,R_k,old_R_k, $Old\_Representatives_R$)

| $Fitness_{New\_Individuals}$ = evaluateMA(R_k, $Old\_Representatives_R$)

| *MAPop*=*MAPop* ∪ *New_Individuals*

Offspring= tournamentselectionByDominance (*MAPop*)

for $Offspring_i$, $Offspring_{i+1}$ **in** *Offspring*:

| **if** *MACXPB* >random(0,1):

| | $Offspring_i$, $Offspring_{i+1}$ = uniformcrossoverMA($Offspring_i$, $Offspring_{i+1}$)

while $Offspring_i$ **in** *Offspring*:

| **if** *MAMUTBP* >random(0,1):

| | $Offspring_i$'=uniformmutation($Offspring_i$)

| **if** $Offspring_i \subseteq$ *MAPop*:

| | delete $Offspring_i$

$Fitness_{Offspring}$=evaluateMA(*Offspring*, R_k, $Old\_Representatives_R$)

MAPop=*MAPop* ∪ *Offspring*

New_MAPop, Pareto_Front=nonDominatedSorting(*MA_pop*, $N_{\text{MA Individuals}}$)

$Old\_Representatives_R$=deleteUnusedOldRepresentatives(New_MAPop $Old\_Representatives_R$)

return *New_MAPop*, *Pareto_Front*

4.3 In Depth Explanation of the Algorithm Operators and Implementation

In this chapter, the functions of the algorithm are explained in detail. The functions are described in the order they are used in the algorithm. In 4.3.1, the used programming language and the used framework are mentioned. In chapter 4.3.2, the representation of the individuals of the MA and the SA is described. In chapter 4.3.3, the A* algorithm used to connect the waypoints of the SA individuals is clarified. In chapter 4.3.4, the Input data of the algorithm and in chapter 4.3.5 the initialization process is explained. From chapter 4.3.6 to 4.3.13, all functions of the SA and from chapter 4.3.14 to 4.3.20 all functions of the MA are described. In chapter 4.3.21, the termination conditions and data output are explained. Lastly, the algorithm is summarized and discussed in chapter 4.4.

4.3.1 Programming Language and DEAP-framework

The implementation was done in python. Additionally, functions from the evolutionary computation framework Distributed Evolutionary Algorithms in Python (DEAP) were used. DEAP provides several modules for implementing evolutionary algorithms [15]. A few modules like the binary Heap implementation from Singh [35] and the merge sort algorithm implementation from [30] were also used.

4.3.2 Representation of Individuals

In this thesis, every vertex has a numerical value for identification. The individuals of the SAs consist of two forms of representation: waypoints and the consecutive vertices, in which the waypoints are converted to. $Individual_{i,A} = \{w_{s_A}, w_0 \dots w_n, w_t \; ; \; w_{s_A}, \; v_0 \dots v_l, w_0, v_{l+2} \dots w_{t_A}\}$. Figure 5 shows a possible conversion from waypoints to consecutive vertices. The first waypoint and the last waypoint as well as the first and the last vertices are the start *s* and the target point *t* of the agent. The start and target points cannot be changed by any of the operators. The number of waypoints between start and target points are variable. The waypoints and vertices are numerical and point towards one of the vertices of the map. The two forms of representation are necessary, because the same set of waypoints can lead to a different set of consecutive vertices. The reasoning behind this is explained in the following chapter 4.3.3.

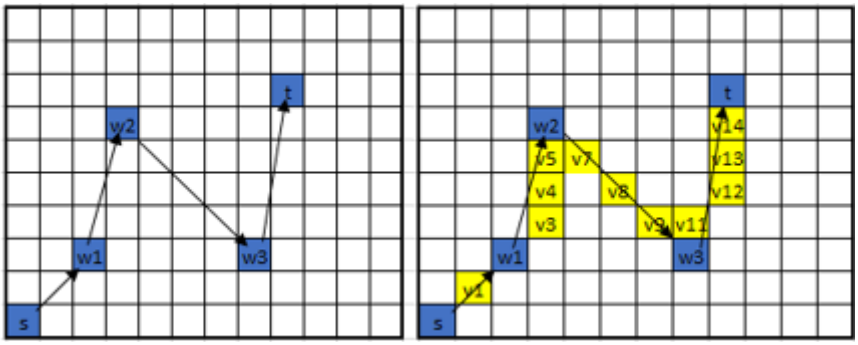

Figure 5: Interaction between waypoints and consecutive vertices

The individuals of the MA have as many genes as there are agents. $Individual_{MA,i} = \{gene_0, \dots, gene_k\}$. Each gene refers to a representative from a different subpopulation of an agent. Figure 6 shows how each gene of the MA refers to a representative from the subpopulations of the agents.

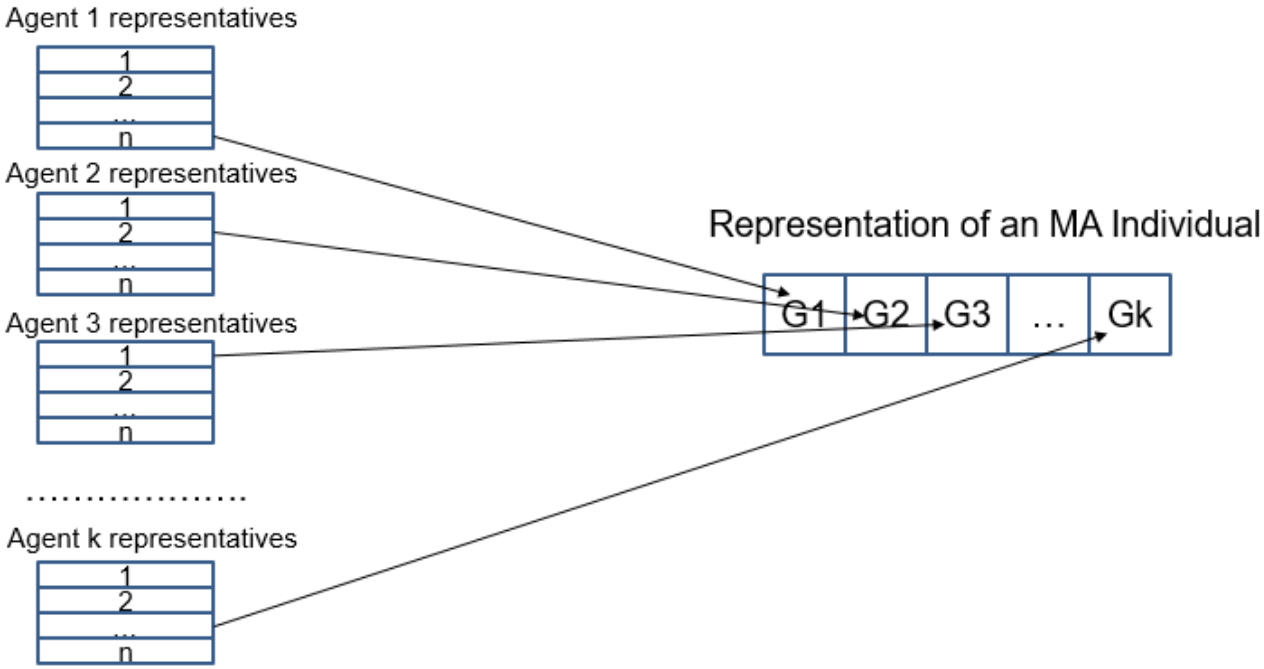

Figure 6: Representation of a MA individual

4.3.3 Converting Waypoints to consecutive Vertices

The A* algorithm is used to translate the waypoints into neighboring vertices. A possible heuristic could be used, which calculates the direct route disregarding obstacles between the examined point and the target point. With the existing assumptions, the calculation of the heuristic distance h would be described by the following function:

$$h=max(|w_{1y}-w_{2y}|,|w_{1x}-w_{2x}|) \quad (4.4)$$

With w_{1x} and w_{1y} being X- and Y-coordinates of one waypoint and w_{2x} and w_{2y} being X- and Y-coordination of the second waypoint.

This is the recommended heuristic for this algorithm for most maps. Nevertheless, in this thesis, precomputed distances are used as the A* heuristic. For this purpose, in the beginning of the algorithm the distance between each points is determined and saved. This distance can then be used to determine the route to the next waypoint. The disadvantage of this variant is that the preparation of the map involves exponential time depending on the number of vertices on the map. The advantage is that the A* algorithm requires the minimum effort to compute the path between two waypoints. This is particularly advantageous on small maps with many agents and with many obstacles and the resulting long routes. The main reason this variant is used here, is that the precomputed distances are practical for the experiments, since they only have to be calculated ones for each map. The difference in the results between the two heuristics should be small.

Additionally, two variants of the A* algorithm are implemented. A deterministic and a stochastic one. While the deterministic version works like a classic A* algorithm, the stochastic variant works as follows: Whenever the next node is pulled from the priority queue of the known nodes, the algorithm checks if there is more than one node with the same highest priority. Out of these nodes, the algorithm pulls out a node randomly with evenly distributed probabilities. The used priority queue is a min-heap. Therefore, nodes with the same priority are found fast. For the min-heap, the implementation in Singh [35] was used and altered. The stochastic version does not lead all possible paths to be equally likely. Instead, all branches of a fork are equally likely, as shown seen as an example in the figure 7.

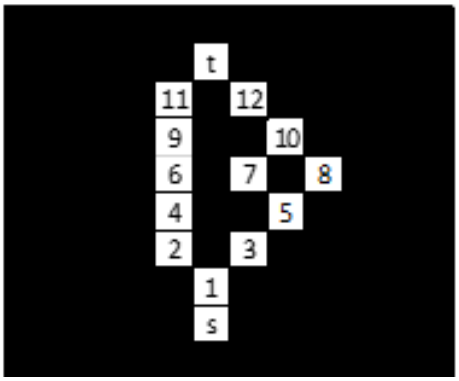

Figure 7: Example for the probability of the branches

Using the stochastic A variant on the shown grid to find a fastest path from s to t could end in the consecutive vertexes a={s,1,2,4,6,9,11,t},b={s,1,3,5,7,10,12,t} and c={s,1,3,5,8,10,12,t}. The probabilities of getting each path is different. The algorithm has at vertex 1 a 50-50-chance of moving to 2 or 3 and then at vertex 5 another 50-50 chance of moving to vertex 7 or 8 resulting into the probabilities*: p_a*=50%,*p_b*=25%,*p_c*=25%*

This has two side effects. First this procedure makes the A* algorithm computationally more expensive. Secondly, the stochastic variant ensures that every change in a waypoint leads to new calculations of the consecutive vertices between this waypoint and others. To help the stochastic variant, every time the A* algorithm is used, a number of vertices between the waypoints are converted into waypoints depending on the length of the route (see figure 8). This is used to cause path sections to solidify. As a result, changing a waypoint does not change a very long route section. How many waypoints are added and which A* variant is used is tested during the parameterization tests.

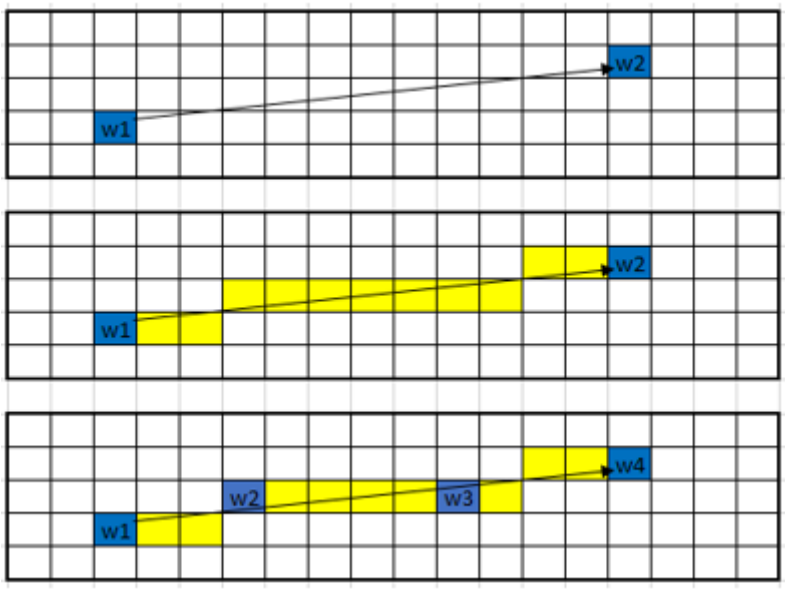

Figure 8: A add extra waypoints example*

4.3.4 Input data and Preparation:

The input data can be classified into two categories:

- **Environment-related data**
- **Parameter settings**

The environment-related data consists of all the data on the MAPF problem. This includes the map, the number of agents and the associated start *s* and target points *t*. The start and target points are saved as lists and the number of agents as a constant. The map is converted to a graph *G=(V,E)*. Each vertex on the map is represented by a node. Each node has, among other variables, an X- and Y- coordinate, a numerical value for identification and a list of pointers to all neighboring nodes. Additionally, for

every agent a list of all the points, which are reachable from the start position (the connectivity-list) is created. This is important for all maps, which have sections of passable space, which are isolated from the rest of the map by obstacles. With this connectivity-list, it is possible to only use reachable waypoints. Furthermore, a distance matrix from every point to every other point is created for the A* algorithm heuristic.

The parameter settings include all parameters that can be changed in the algorithm. These parameters are described in the Appendix section A.

4.3.5 Initialization

The initialization function creates individuals for the subpopulation of the SA and the MA until the population size is reached. For each individual of a SA subpopulation, h waypoints are randomly determined with evenly distributed probabilities from the associated connectivity-list with $h \in \{0 \leq j \leq N_{\text{Startinggenes}} | j \in \mathbb{N}\}$ and $N_{\text{Startinggenes}}$ being an input parameter. Furthermore, the associated start s_A and target t_A point are added and the consecutive vertices between the waypoints are determined by the A* algorithm. The MA individuals consist of one gene for every agent. The individuals are created by assigning one of the representative-slots to every gene (one representative per agent).

Initially, only the individuals of the SA subpopulations are evaluated and only according to the objective path- length by counting the number of actions/consecutive vertices. The fitness value of the MA individuals ($Fit_{Makespan}$, $Fit_{SumOfCosts}$, $Fit_{Overlaps}$) and the fitness value Fit_{CLC} of the SA individuals can not be determined yet because the representatives have to be specified beforehand.

4.3.6 SA Collisionlist

Every generation after the first generation, the individuals of the SA subpopulations are evaluated with regard to the objective collision count. For this, the $Collisionlist_{VX}$ needs to be prepared. The $Collisionlist_{VX}$ is a two dimensional matrix with the dimensions Vertices V and Time steps X. A cell C_{vx} of the $Collisionlist_{VX}$ holds the information how many representatives of all agents are at vertex v at time step x (equation 4.3). Before the collision count evaluation at the beginning of a SA generation, the information of the representatives of the own agent is removed from the collision list. This prevents collisions of an individual of an agent with the representatives of the same agent to be included in the calculation of the collision count. Algorithm 4 illustrates how the removal is done. At

the end of the generation, the information of the new representatives is added to the $Collisionlist_{VX}$ like seen in algorithm 5.

Algorithm 4: Remove Representatives from Collisionlist
Input: Representatives of Agent A: R_A, $Collisionlist_{VX}$
Output: $Collisionlist_{VX}$'
for π_i in R_A:
| **for** x=0; x< π_i; x++ :
| | $Collisionlist_{VX}$[x][π_i[x]]= $Collisionlist_{VX}$[x][π_i [x]]-1

Algorithm 5: Collisionlist update/Creation
Input: Representatives of Agent A: R_A, $Collisionlist_{VX}$# if Gen one $Collisionlist_{VX}$=[]
Output: $Collisionlist_{VX}$'
CollisionlistVertices=[0]*$N_{NumberOfVertices}$
for π_i in R_A:
| **for** x=0; x< π_i; x++ :
| | **if** length($Collisionlist_{VX}$)<x:
| | | $Collisionlist_{VX}$.append(copy(CollisionlistVertices))
| | $Collisionlist_{VX}$[x][π_i[x]]++

4.3.7 SA Evaluation Collision Count

Starting with generation two, the individuals of the SA are evaluated with regard to the objective collision count at the beginning of the SA generation. Algorithm 6 shows how the evaluation is done. The fitness value $Fit_{CLC,i}$ of an individual *i* is increased by one for each representative of the other agents the individual shares a vertex *v* with at the same time step *x*. Additionally, the collision count fitness value is increased by the $C_{SwappingPenaltyValue}$ for every representative of the other agents which is in the same vertex *v* at time step *x+1* at which individual i was in at time step *x*. The information how many representatives of the other agents are at vertex *v* at time step *x* and *x+1* is taken out of the $Collisionlist_{VX}$.
The described evaluation process at the beginning of an SA only happens from the second generation onward, because the Fit_{CLC} values can only be calculated once the agents have representatives. The described process must happen before the parents are selected, because the representatives might have changed since the last evaluation and thus the collision count fitness value needs to be updated before selection.
If the SO SA variant is active, the two fitness values Fit_{CLC} and Fit_{PL} of each individual are normalized and the weighted sum is formed using the weights from the input data.

Algorithm 6: Evaluate collision count

Input: Population of agent A: $SAPop_A$, $Collisionlist_{VX}$, $C_{SwappingPenaltyValue}$

Output: $Fit_{CLC,SAPop_A}$

for π_i in $SAPop_A$:

| $Fit_{CLC,i}$=0

| **for** x=0; x< π_i; x++ :

| | $Fit_{CLC,i} = Fit_{CLC,i} + Collisionlist_{VX}$ [x][π_i[x]]

| | $Fit_{CLC,i}$= $Fit_{CLC,i}$+$C_{SwappingPenaltyValue}$ * $Collisionlist_{VX}$ [x+1][π_i[x]]

4.3.8 SA Selection

The selection process is used to create the parents for crossover. For the SO SA version, a simple tournament selection with two participants with the selection criteria of the weighted sum value is used. For the MO SA, the crowding distance is assigned and the parents are selected by tournament selection with two participants based on dominance. If the selected individuals do not dominate one another the crowding distance is used to determine the winner of the tournament selection. For this purpose the functions *assignCrowdingDist* – for crowing distance- and *selTournamentDCD* – for the tournament selection based on dominance- from the DEAP framework are used [14]. As many parents are selected as, there are individuals in the population.

4.3.9 SA Crossover Operator

One of the two One-Point Crossover variants is used, which are compared against each other in parametrization. In the first One-Point Crossover variant, a random value between zero and one is determined. The value is multiplied by the number of waypoints of each parent. The resulting product is the $Crossoverpoint_i$ for a $parent_i$. Figure 9 shows how the crossover variant works.

$$Crossoverpoint_i = round(random([0,1]) \bullet N_{Number\ of\ Waypoints}) \quad (4.5)$$

- [] being rounding brackets in this context
- With the $N_{Number\ of\ Waypoints}$ being the waypoints between the starting and target point

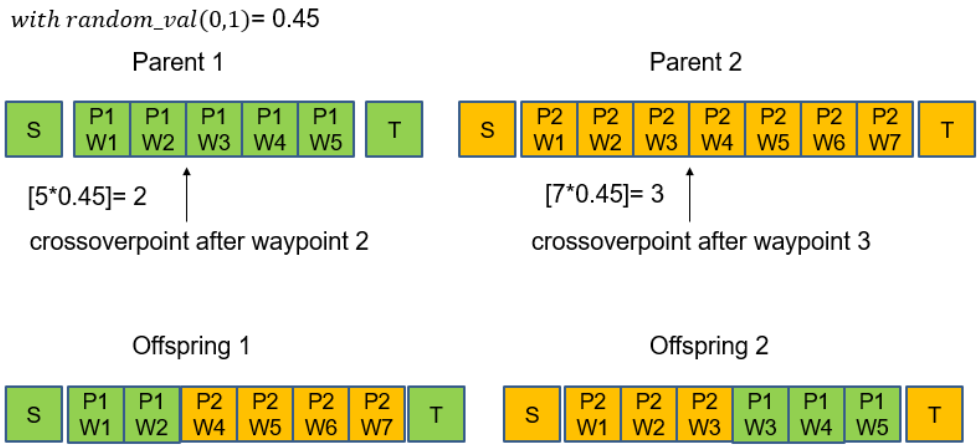

Figure 9: Crossover variant one

Since it is possible that the path length of the offspring increases after the crossover. The second crossover tries to reduce this effect. The second crossover variant randomly selects a waypoint of a parent and searches in the other parent for the waypoint that is closest to the first selected waypoint using the distance matrix. The crossover point is set after each of the two points. This crossover variant is supposed to minimizes the distance difference between the parents and the offspring: the maximal increase between the sum of the parent path lengths and the offspring' path lengths is two times the distance between the waypoints before and after the crossover point.

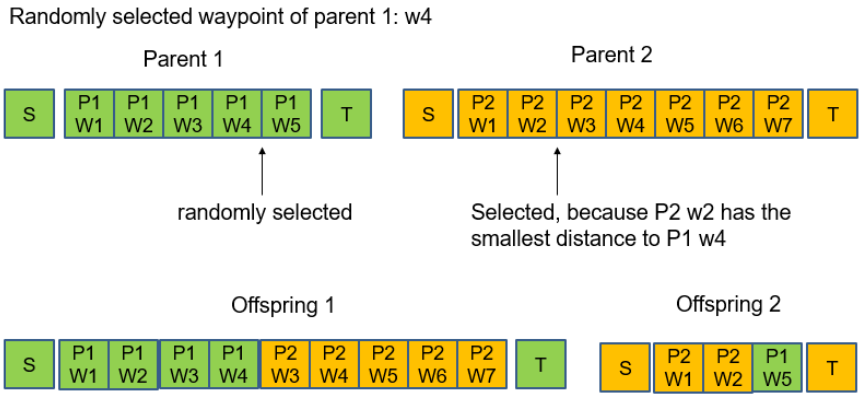

Figure 10: Crossover variant two - nearest point

Many of the approaches presented in chapter 3.2 and 3.3 use some kind of deletion operator to fasten the convergence. Since many of the created children have long path lengths, a deletion function was built into the crossover function to counteract this. A percentage value between zero and the CDV is set at random for each child – with CDV being the crossover-deletion-value of the input parameter. This is multiplied by the number of waypoints to the left and right of the crossover point in order to

determine how many of the waypoints are deleted starting from the crossover point. The result is rounded down. The deletion operator uses a percentage rather than a fixed number, because the A* value adds waypoints into the chromosomes.

$$N_{\text{deleted Waypoints to the left}} = \lfloor N_{\text{number of waypoints to the left}} * \ random[0, CDV] \rfloor \quad (4.6)$$

$$N_{\text{deleted Waypoints to the right}} = \lfloor N_{\text{number of waypoints to the right}} * \ random\_value(0, CDV) \rfloor \quad (4.7)$$

With $N_{\text{number of waypoints to the left}}$ being the number of waypoints to the left of the crossover point and $N_{\text{number of waypoints to the right}}$ being the number of waypoints to the right of the crossover point

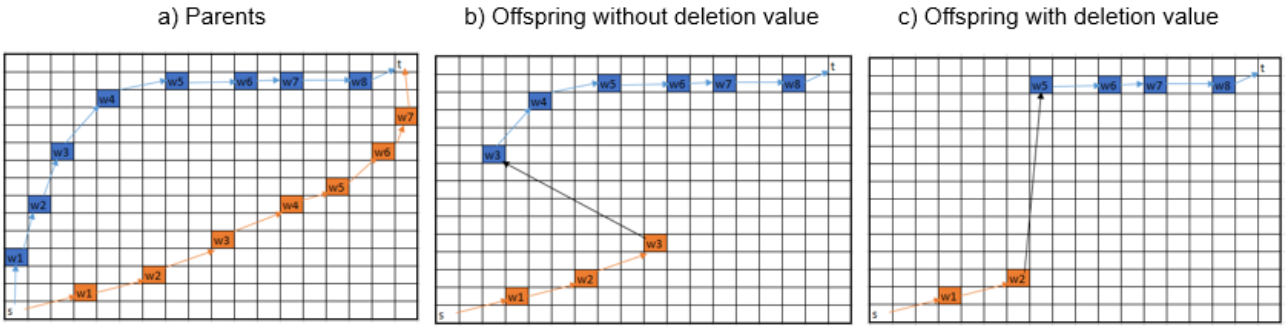

Figure 11: Deletion value example.

Figure 11 shows an example of the use of the deletion operator. The middle picture b shows a possible offspring made from the parents shown in the picture a. The path length of this offspring is far worse than the path length of the parents and the connection path even furthers the distance to the target. The picture on the right shows the difference with the deletion value. The path length is far more acceptable than it was before.

The vertices between the new waypoint connections are determined with the A* algorithm. In the special case, that a parent only has start and target points as waypoints, a random step is changed into a waypoint and the crossover takes place as described above. If there are no steps between a parents starting and target point, no crossover takes place.

4.3.10 SA Mutation Operators

In this thesis, three different mutation operators with different individual mutation probabilities are used:

Mutation Gene Deletion:

With the mutation probability of *SAMUTBP1* an individual is mutated by the Gene-Deletion mutation operator. This operator deletes any non-mandatory waypoint with the probability of

$p_{mut\_del} = \frac{1}{N_{\text{Non mandatory Genes}}}$ (4.8). An average of one waypoint is deleted during execution. The A* algorithm translates the new waypoint-connection into consecutive vertices. In the special case that there are no waypoints between start and target point, nothing is changed. Figure 12 shows how the Gene Deletion mutation works.

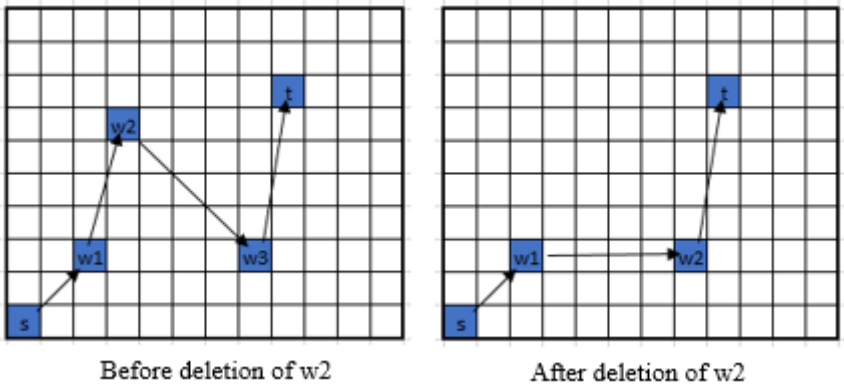

Figure 12: Mutation Operator Gene Deletion

Mutation Shift in Neighborhood

With the mutation probability of *SAMUTBP2* is mutated by the Mutation Shift in Neighborhood operator. Every one of the consecutive vertices of an individual mutated by this operator has a probability $p_{mut\_shift} = \frac{1}{N_{\text{Consequtive Vertices}-1}}$ (4.9) to be converted into a waypoint and moved in its neighborhood. If the chosen vertex is already a waypoint, it is just moved in its neighborhood. This way on average one waypoint is moved. The A* algorithm translates the new waypoint connections into consecutive vertices. Figure 13 shows how the Shift in Neighborhood mutation works.

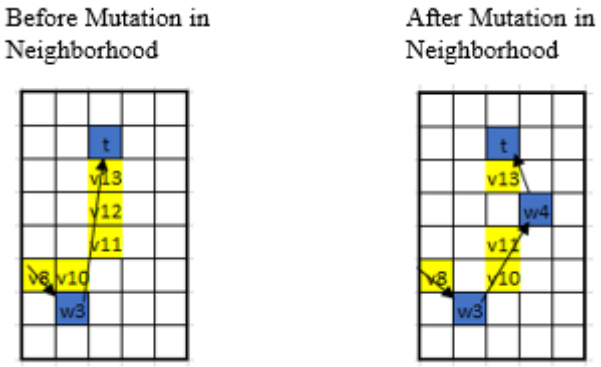

Figure 13: Mutation in Shift Neighborhood example

In the special case that there are no steps between the start and target point, a waypoint is added. This waypoint is either a neighbor of the start or target point. The A* algorithm determines the new consecutive vertices between the start-point and the new point and the new point and the target point. While in other approaches this kind of mutation is only used on waypoints, in this approach, it can also be used on the steps between the waypoint. This way the operator has a higher chance to change a path into avoiding a conflict if no waypoint is near that conflict.

Mutation Insert Random Waypoint

With the mutation probability of *SAMUTBP2* is mutated by the Insert Random Waypoint mutation operator. This operator adds a random waypoint into the chromosome of an individual. For every waypoint-to-waypoint connection, a random waypoint is added by the probability $p_{mut\_insert\_rand}=\frac{N_{\text{consecutive vertices between two waypoints}}}{PL_i}$ (4.10). On average one waypoint is added into an individual with a higher chance of being added between two waypoints that have a long path between them. If there are no consecutive vertices between start and target point a random waypoint is added between them. The A* Algorithm translates the new waypoint connections into consecutive vertices. Figure 14 shows how the Mutation Insert Random Waypoint works.

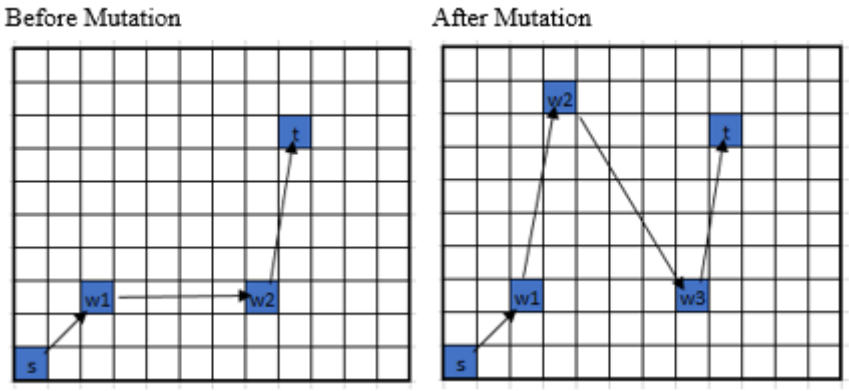

Figure 14: Mutation Insert Random Waypoint example

The Operator added a waypoint between former w1 and w2 becoming the new w2. The old w2 is now the new w3.

Additionally, similar to the crossover deletion function a random percentage between 0 and the *Mutation deletion value (MDV)* of the waypoints left and right to the added waypoint are deleted – with the *MDV being the Mutation deletion value* from the parameter input data and evenly distributed probabilities used. This way it is possible to insert a new waypoint and still keeping the path length low. After mutation, any offspring, which was not effected by any change, is deleted.

4.3.11 SA Evaluation

In the next step, the created children are evaluated. The Fit_{PL} value is calculated by counting the consecutive vertices and the Fit_{CLC} value, as described in chapter 4.3.7, by counting all vertex conflicts and all possible swapping conflicts with the representatives of the other agents and multiplying them with C_{SPV}. In generation one only the Fit_{PL} is calculated, since there are no representatives yet. In the SO SA version, the fitness values of all individuals, including the parents, are normalized and the weighted sum is calculated using the weights. In the MO SA the fitness values are also normalized for the crowding distance calculation.

4.3.12 SA SELECTION Next Gen and Representatives

The population of the next generation is build out of the best individuals of the old population and the children population. The SO SA version sorts the individuals by their weighted sum fitness value. For this purpose, the merge sort algorithm from Popović [30] is altered and used for the implementation. The MO SA uses the non-dominated sorting algorithm to build the population of the next generation. For this purpose, the non-dominated sorting algorithm from the DEAP framework is used [13]. The population size stays the same between generations. The representatives are chosen the same way (SO SA with merge sort, MO SA with non-dominated sorting). The number of representatives is defined by the number-representatives-variable.

4.3.13 SA Collisionlist update

Lastly, the $Collisionlist_{VX}$ is updated by inserting the information of the new representatives. Like stated in chapter 4.3.6 the $Collisionlist_{VX}$ holds the information of how many representatives of an agent are to a certain time step in a certain vertex. The insertion of new information from a representative works like described in algorithm 5. For every step of a representative, the associated tuple value is increased by one. A cell is identifiable by the combination of vertex and time step it represents equation 4.3.

4.3.14 MA Evaluation Generation One

The first generation of the MA starts with the evaluation of all individuals. The genes of the MA individuals refer to representatives of the subpopulations (like described in chapter 4.3.2). Algorithm 7 shows how to translate a gene of an individual into a path. To make equations more understandable genes of individuals of the MA are assumed to be equivalent to the associated path. Genes can refer to representatives of the current SA subpopulations or representatives of old SA subpopulations saved in the $\text{Old\_Representatives}_R$ list. The $\text{Old\_Representatives}_R$ list contains all saved representatives of the previous generations. The saving of old representatives is explained in chapter 4.3.15.

Algorithm 7: Reading out paths
Input: Individual of the MA subpopulation: $Individual_{MA,j} = \{gene_0, \dots, gene_k\}$, representatives of all agents of the current generation: R_k, list of all saved representatives: $\text{Old\_Representatives}_R$
Output: $Fit_{Overlaps}$
for $gene_i$ **in** $Individual_{MA,j}$:
| **if** $gene_i =< N_{\text{Number of Representatives for each agent}}$ **:**
| | $Path\_of\_individual_i = Representative_i[gene_i]$
| **else:**
| | $Path\_of\_individual_i = Saved\_Representatives[gene_i]$

The individuals are evaluated by the objectives makespan, sum of costs and overlaps described in chapter 1.1. algorithms 1.1, 1.2 and 1.3.

Makespan: The algorithm calculates the makespan fitness value $Fit_{Makespan}$ by determining the longest path length value of all paths being part of the MA individual.
Sum of Costs: The algorithm calculates the sum of costs fitness value $Fit_{SumOfCost}$ by summing up all path lengths of all paths being part of the MA individual.
For the overlaps fitness value $Fit_{Overlaps}$, all the collisions between the paths of an $individual_{MA,i}$ must be calculated. Algorithm 8 shows the pseudocode to the evaluation for better understanding. To calculate the swapping- and the vertex-conflicts two lists are used in the algorithm: $list_{VertexConflicts}$ and $list_{SwappingConflicts}$. $list_{VertexConflicts}$ contains the number of agents at every vertex *v* at one time step *x* and is used to determine the vertex conflicts. The second list, $list_{SwappingConflicts}$, has at the index of each vertex *v* a list of all paths of $individual_{MA,i}$, which were at time step *x+1* at the given vertex *v*. To initialize the lists, $list_{VertexConflicts}$ is filled with

zeroes one for each vertex. The $list_{SwappingConflicts}$ is filled with empty lists, one for each vertex. For each time step *x* three things are done:

First, the lists are updated: The vertex values of the $list_{VertexConflicts}$ are increased by one for each path of the $individual_{MA,i}$ being at the vertex at time step *x*. $List_{SwappingConflicts}$ is updated by putting a pointer of each path of the $individual_{MA,i}$ into the list with the vertex-index at which the agent is at time step *x+1*.

Secondly, for each path *i* the vertex in the $list_{VertexConflicts}$ where the agent *i* is at time step *x* is checked. The overlaps variable is increased by $(list_{VertexConflicts}[\pi_i\ [x]] - 1) \cdot 0.5$. For the swapping conflicts the list in $list_{SwappingConflicts}$-cell at the vertex-index corresponding to the agents location at time step *x* is checked. The list contains all agents which are in this vertex *v* at time step *x+1*. For each agent of this list, the location at time step *x* is compared with the examined paths *i* location at time step *x+1*. If they match the fitness value of the overlaps objective for the $individual_{MA,i}$ is increased by one. Third, the lists vertex values are set back to zero and to empty lists.

Algorithm 8: MA Fitness Evaluation of the Overlaps Objective

Input Population of the MA: *MAPop*, $Fit_{Makesspan,MAPop}$

Output: $Fit_{Overlaps}$

$list_{VertexConflicts}$ =[0]*$N_{NumberOfVertices}$

$list_{SwappingConflicts}$ =[]*$N_{NumberOfVertices}$

for $individual_g$ **in** *MAPop*:

| **while** *time step x=0, x<* $Fit_{Makesspan,g}$, *x++*

| | **for** π_i **in** $individual_g$: # get the paths referred to by the genes, fill lists

| | | *list_VertexConflicts*[π_i *[x]]=+1*

| | | *list_SwappingConflicts*[π_i *[x+1]].append*(π_i)

| | **for** π_i **in** $individual_g$:

| | | $Fit_{Overlaps,g} = Fit_{Overlaps,g}$ + *(list_VertexConflicts*[π_i *[x]]-1)*0.5*

| | | **for** $\pi_{j,j!=i}$ **in** *SwappingConflicts*[π_i *[x]]:*

| | | | **if** π_i *[x+1]==* π_j *[x]:*

| | | | | *Overlaps= Overlaps+1*

| | **for** π_i **in** $ndividual_g$: # remove location information from list

| | | *list_VertexConflicts*[π_i *[x]]=-1*

| | | *list_SwappingConflicts*[π_i *[x+1]].pop()*

4.3.15 MA handling Representatives swaps and saving old representatives

Like already stated, every individual of the MA has one gene for each agent and every gene points to one of the representatives of the agent. When one representative of the subpopulation of the SA is switched out, a solution of the MA, which uses this representative would be changed too. If that happens the MA might lose good solutions after each run of the SAs. To counteract this the MA uses following procedure starting with the second generation: For every individual of the MA which points at a representative of an agent, which was changed by the SA, the MA creates another individual. This new individual will point at the new representatives. The old individual refers towards the old representatives. For this reason, the old representatives, which are used by the MA, are saved in the $Old\_Representatives_R$ list.

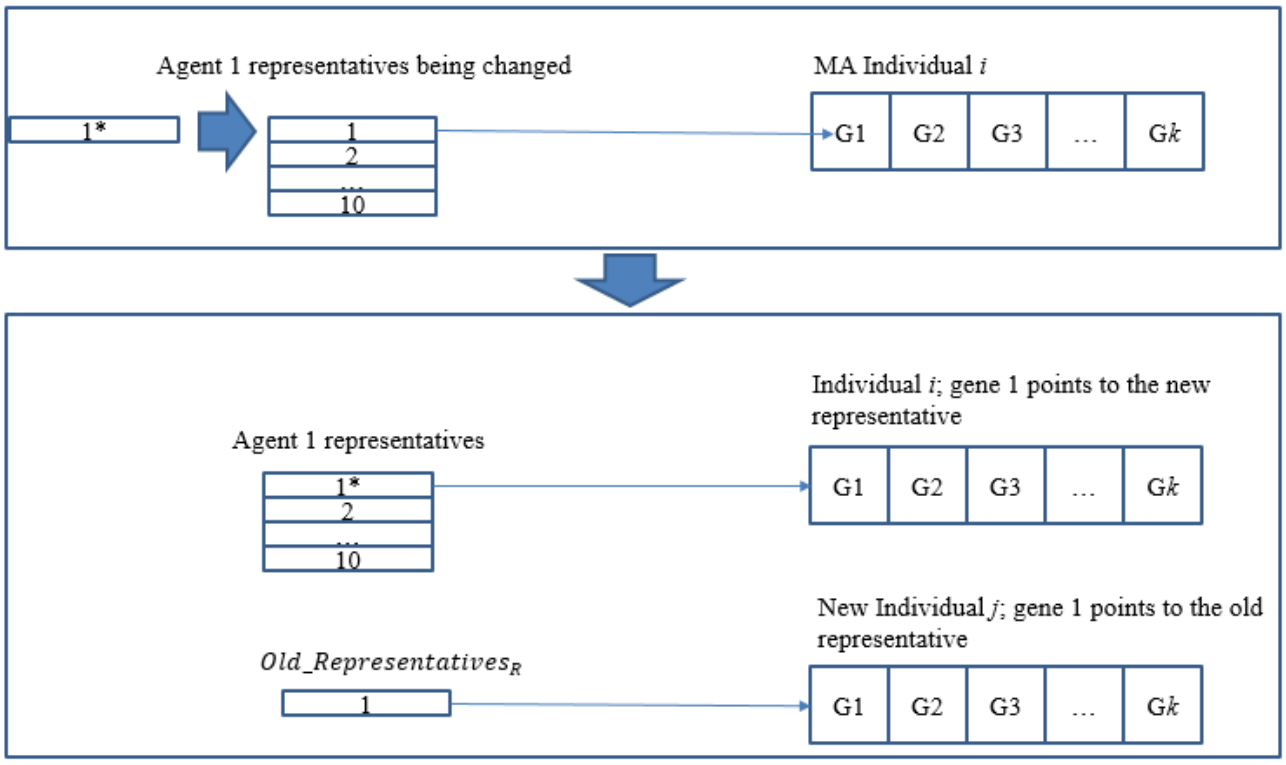

Figure 15: Handling of new representatives

This solution leads to an overall bigger population of the MA, which slows down the algorithm. On the other hand, the impact will not be too high, since the SA is expected to have the biggest impact on the time performance, because the SA has one subpopulation for each agent to evolve. Additionally, this solves two problems: Firstly, old solutions in the MA subpopulation do not disappear by changes in the subpopulation. Secondly, the new representatives of the SAs, which are

supposed to be better than the old ones, are part of the population faster. Otherwise, the mutation operator would be the only way to insert the new SA representatives into the MA population. The new solutions created by this function are then evaluated.

4.3.16 MA Selection

The parents are selected by tournament selection based on dominance with two participants for each tournament.

4.3.17 MA Crossover

For crossover, uniform-crossover like described in Narmadha et al. [26] is used. For this purpose, the uniform-crossover function from DEAP is used [11]. The cross probability is determined by the input crossover probability variable $MACXPB$.

4.3.18 MA Mutation

For the mutation of the MA offspring uniform mutation like described in Soni and Kumar [36] is used. For this purpose, the uniform mutation function from the DEAP framework is used [12]. The mutation probability is determined by the input Mutation-probability variable $MAMUTBP$. The range of possible values for a mutated gene are from one to the Number Representatives: $PossibleValForMAMutation \in \{1 \leq j \leq N_R / j \in \mathbb{N}\}$. This way, the mutated genes refer only towards representatives of the current generation. After the mutation, the newly created offspring are evaluated as described in chapter 4.3.11.

4.3.19 MA Environmental Selection

After the evaluation, the non-dominated sorting algorithm like described in Deb et al. [8] is used to select individuals of the newly created offspring and of the old population to create the next population. The new population size is defined by the MA Population Size variable. For this purpose, the non-dominated sorting algorithm from the DEAP framework is used [13]. Additionally the set of pareto optimal solutions is determined by the non-dominated sorting algorithm.

4.3.20 MA Deleting Saved Representatives

Lastly, the MA deletes old representatives from the saved representative list $(Old\_Representatives_R)$, which are not used by the new population.

4.3.21 Saving Data, Termination Criteria and Data Output

The Output data after each generation is saved as a data frame. For the purpose of testing the generation, the function evaluations, the steps of every agent of the solutions of the pareto front, the fitness values of the solutions of the pareto front, the random-seed value and the number of nodes traversed with the A* algorithm are saved every 10 Generations or if the agents terminate early.
The algorithm terminates when one of two conditions is met: Either the algorithm reaches the maximal number of generations or the algorithm finds a solution with the best possible fitness values. For this solution it is required that the path of each agent in this solution has the minimum possible path length value, thereby minimizing makes span and the sum of the costs, and the overlaps fitness value is zero.
Once the termination criteria is fulfilled, the saved output data is put out as a csv file and the algorithm terminates.

4.4 Summary and discussion

In this chapter, a co-evolutionary approach for solving the multi-objective multi-agent pathfinding problem was presented. The algorithm uses the master-, slave-architecture similar to Kala [18]. A subpopulation for every agent is created, which is evolved by the SA, and the MA evolves a subpopulation, which refers to the representatives of the SA to create a set of pareto optimal solutions for the MAPF problem.
Like stated in chapter 2.2 the tests in Bucci and Pollack [3] showed among other things that using every test available leads to a better solution. Since the evaluation of every solution of the *k* agents in all possible combination would use up too much time, the SA individuals are instead evaluated by objectives, which have a connection to the MA objectives. The SA path length objective optimizes the makespan and sum of costs objective of a MA solution. The SA collision count objective is supposed to optimize the overlaps objective of the MA. Care was taken here to evaluate an individual using the representatives of all other agents using the collision list. So in some way every available

test is used. Admittedly, an individual with a comparatively small collision count value can still lead to a comparatively high overlaps value since:

- The swapping collisions in the collision count objective are estimated,
- The collision count comes from using all representatives from all agents while in the overlaps objective only the collisions between one representative from every agent are examined,
- In the calculation of the overlaps objective vertex conflicts between many agents in one vertex are penalized exponentially, while vertex conflicts in the collision count can only be penalized linearly.

The SA uses waypoints like most of the approaches presented in chapter 3.2 and 3.3. The waypoints are translated into paths using the A* algorithm. Two variants of the A* variant were implemented: A stochastic and a deterministic one. Two heuristics were presented of which the precomputed distances will be used for the tests to minimize the computational power needed. Additionally, the A* algorithm introduces new waypoints into long paths to solidify paths. Two one-point crossover variants are introduced, which differentiate in the setting of the crossover point. Additionally, the crossover operator uses a deletion operator to fasten the convergence. Three mutation operators are used. A Waypoint deletion mutation, a mutation operator, which moves a step of the path in its neighbourhood, and a waypoint operator, which introduces a new random waypoint, and uses a deletion operator to smoothen the transition. The last mentioned mutation operator introduces new solutions to the population. Two different versions of the SA were implemented: One, which uses a weighted sum approach to optimize the objectives (SO SA), and another one, which solves the pathfinding problem multi-objectively (MO SA).

The genes of individuals of the MA refer to representatives of each agents' subpopulation. The MA saves old representatives if they are still used in the population. Otherwise, the MA is pretty straight forward in terms of multi-objective evolutionary algorithms using tournament selection, uniform crossover, uniform mutation and the non-dominated sorting algorithm.

The algorithm is computational expensive: The number of function evaluations of the algorithm is calculated by formula 4.11.

$$N_{\text{Functionevaluations}} = N_{\text{generations}} * 2 * (N_{\text{SA Individiuals}} * \mathrm{k} + N_{\text{MA Individuals}}) \quad (4.11)$$

With k being the number of agents and with $N_{\text{generations}}$ being the number of generations the algorithm needed. The "times 2" comes from the additional evaluation of the collision count in the SA and from the evaluation of the new individuals created by the "MA handles representatives"-function.

To reduce the computational cost it would have been useful to stop the evolution of subpopulations, which are unlikely to change anymore. Additionally, elitism would have been probably the better choice for the environmental selection in the SO SA. Other than that, different mechanisms for choosing the representatives could have been compared against each other. The representatives can be chosen by random choice, optimal choice or combined choice [32].

5 Experiments and Evaluation

In this chapter the two variants SO SACCGA and the MO SACCGA are compared with each other and analyzed. For this, both variants are first parameterized in chapter 5.2. In the following section chapter 5.3, the two variants are compared against each other in different environments with a different numbers of agents and different start and target point (scenarios). The quality of the co-evolutionary algorithm should be determined and to test thesis 1 (like formulated in chapter 1.):

> Thesis 1: MAPF multi-objective optimization with co-evolution works better if the subpopulations of the agents are optimized multi-objectively than if the subpopulations are optimized single-objectively with a weighted sum approach.

Additionally, the pareto fronts achieved from the tests are analyzed to determine which of the two algorithms achieves better results, if the decision maker weighs the objectives in the same way as the weighting in the slave algorithm of the SO SACCGA is done. This is done to examine thesis 2 (like formulated in chapter 1.):

> Thesis 2: Using a co-evolutionary approach, if the decision maker weighs the objectives of the Multi-objective MAPF problem with the same weights the objectives of the subpopulations of the agents are weighted using a weighted sum approach, then this weighted sum approach works better than optimizing the objectives of the subpopulations of the agents multi-objectively.

The environments and scenarios from the Mapf.info benchmarks are used for the analysis and parametrization. These benchmarks are described in more detail in Chapter 5.1. Finally, the results are summarized in chapter 5.4.

5.1 Benchmark

The Benchmarks from Mapf.info provide grid-based MAPF maps of real cities, videos games, open grids with or without obstacle, maze-like grids, warehouse-like grids and grids with room-forming obstacles like illustrated in figure 16.

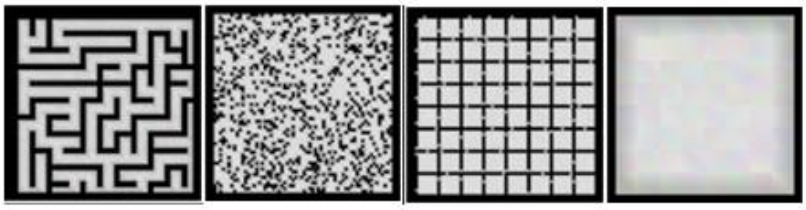

Figure 16: Mapf.info environments [37]

Additionally, each map has 25 random and even scenarios. Each Random scenario consist of 1000 randomly paired points on the largest reachable region [38]. The even Scenarios consist also of random points, whereby each path length-spectrum is represented by the same number of problems [19].

The used environments in this thesis are the open grids with and without obstacles, the maze-like and the room-like maps. These maps are used because they have distinctive features, which can be analyzed.

The Open Maps without obstacles differentiate only in size from each other:

Table 1:Empty Maps

Map type	**Size**
Empty	8 x 8
Empty	16 x 16
Empty	32 x 32

The Open Maps with random obstacles differentiate in size and percentage of obstacles:

Table 2: Random Maps

Map type	**Size**	**Ratio of obstacles to empty space**
Random	32 x 32	10
Random	32 x 32	20
Random	64 x 64	10
Random	64 x 64	20

The maze-like maps differentiate in size and space between walls:

Table 3: Maze Maps

Map type	**Size**	**Space between walls**
Maze	32 x 32	2
Maze	32 x 32	4

Lastly, the room-like maps differentiate in size and room size:

Table 4: Room Maps

Map type	**Size**	**Room size**
Room	32 x 32	3 x 3
Room	64 x 64	15 x 15
Room	64 x 64	7 x 7

Overall, the algorithm is tested on twelve different environments. For the rest of the work the word "problem" is used for a combination of a map, a scenario and the number of agents.

5.2 Parametrization

Value Presentation: The input-parameters are presented in Appendix section A. Table 14 in appendix section B shows the values for every parameter that are being compared against each other during parametrization. For parameters with natural limitations such as probabilities or weights, the values come from the entire spectrum. For the weighting of the collision count objective of the weighted sum approach 0.001 and 0.999 are chosen as extreme values instead of zero and one in order to create a lexicographical optimization instead of neglecting one of the objectives entirely. For all parameters without natural limitations, different and nevertheless promising values are chosen.

Procedure of parameterization:

In order to find a good parameter setting for both the SO SACCGA and the MO SACCGA, the following was done: In the first step, the parameters were improved by finding a promising basic setting. In the second step, the MO SACCGA and the SO SACCGA were parameterized separately.

In order to find the basic setting, all parameters were set to a promising pre-basic setting. Then, for each value setting of a parameter, 31 runs were carried out on seven different test setups illustrated in table 5. Before moving on to the next parameter, the tested parameter was reset to the default setting. The results for all settings of a parameter were then compared against one another. The Convergence and Diversity Metric hypervolume was used for the comparison. To do this, the pareto front fitness values of the repetitions were first normalized and then the hypervolume value was calculated. The settings were then compared by the hypervolume value in dependency to the number of function evaluations or in dependency to the number of nodes that the A* algorithm ran through. After the values for every parameter were compared against each other. The winning settings of the comparison formed the basic setting. This basic setting was then used as start setting to parameterize the MO SACCGA and the SO SACCGA separately. For the separate parameterization, the same procedure as in the first step was used, with the exception that the procedure was done separately for the MO SACCGA and SO SACCGA.

For calculation of the hypervolume values the python implementation from Wessing [43] was used.

Table 5: Test problems for parametrization.

The Feature setting refers to the map types' special feature described in chapter 5.1

Maptype	Size	Featuresetting	Number of Agents
Empty	8 x 8	-	32
Empty	16 x 16	-	45
Maze	32 x 32	2	32
Random	32 x 32	20	40
Random	64 x 64	20	50
Room	32 x 32	3 x 3	40
Room	64 x 64	7 x 7	50

The seven test setups contain different maps with an adapted numbers of agents. Care was taken to ensure that the number of agents is high enough to create an interesting problem. The starting and target points were taken from the even scenarios for each map.

To measure somehow the computational cost of the algorithm, function evaluations and A* traversed nodes are measured. While functional evaluations are used more frequently in the literature (like for example in Weise et al. [42]), the measurement of the A* traversed nodes is rarer. However, the only two parameters that effectively influence the number of function-evaluations are the population size of the SA and the population size of the MA. In the current state of the algorithm the A* algorithm takes a lot of time. Since precomputed distances are used as heuristic, there is not much room to optimize the A*. Additionally, the number of traversed nodes grows proportionally to the population size of the SA, which has probably the biggest impact on the number of function evaluations considering the SA has to run once for every agent every generation. For these reasons, all parameters, which have an effect on the number of function evaluations, are parameterized using the hypervolume value in dependence to the number of function evaluations. All the parameters, which have no effect on the number of function evaluation but might have an effect on the number of A*-traversed nodes, are optimized using the hypervolume value in dependence to the number of A*-traversed nodes.

For comparing the hypervolume values in dependence to the computational time the average hypervolume of the 31 runs of all values of a parameter were visualized with a 68% confidence interval (standard error) and put into a graph.

For the comparison a parameter value *a* was considered better than another value *b* of the same parameter if the graph showed that:

- *a* has a better average hypervolume value at generation 100 than *b* is having with the same computational cost while *a* has a smaller or as high computational cost as gen *b* after 100 generations
- *or b* has a smaller average hypervolume value at generation 100 than *a* is having with the same computational cost.

If both values *a* and *b* had the same hypervolume value in dependency to the computational cost after one value reaches generation 100, it was checked which value reached it first. Else the values were considered being as good as one another. The procedure was used to create the ranking tables 6 to 8 Table 6 shows the results of the first step of finding a basic setting. Table 7 and 8 show the results of the optimization of the MO SACCGA and SO SACCGA. The tables show for each parameter how their values were ranked on each environment. The resulting value on the far right of the table shows, which value won the comparison over all maps together. Table 9 explains the icons used in table 6 to 8.

Table 6: Parametrization Ranking table Basic setting

Finding the Basic setting									
					Maptypes				
Paramters	Startingvalue	empty-8-8.map	empty-16-16.map	maze-32-32-2.map	"random-32-32-20.map	random-64-64-20.map	"room-32-32-4.map	room-64-64-8.map	Resulting value
SA-Population-Size	20	20>28>>12	28>20>12	28>/20>12	28>20>>12	28>/20>>12	28>20>>12	28/20>>12	28
Maximal-Number-of-Starting-Chromosones	3	1>3>/2>>5	1>2>/3>5	5>3/2>1	1>2>>3>5	1>/2>/3>>5	1>2>3>>5	1>2>3>>5	1
SA-Crossover-Probability	0,25	0.25>0>0.5>1>0.75	1>/0.75>0.5/0.75>0	0.5>0>0.25>/0.75>1	0.75>/1>0.5>0>0.25	1/0.75/0.25>0.5>>0	1>0.75>0.5/0.25/0	1>/0.75>0.5>0.25>0	0.75
SA-Mutation-Probability-Shift-in-Neighborhood	0,75	0.75>1>0.5/0>0.25	0.5>1/0.75>0.25>0	1/0.75>0.5>0.25>>0	0.5>1>0.25>0>>0.75	0.75>/1>0/0.25/0.5	0.75/1>0.5>0/0.25	1/0.5>0.75>0.25>0	0.75
SA-Mutation-Probability-Gene-Deletion	0,25	1>0.25>/0.5/0>0.75	1/0.75>0.25>0.5>0	0.5>/1/0.75>0/0.25	1>0>0.5>>0.25/0.75	0.75/0.25/0.5>1>0	0.5>1>0.25>0.75/0	1>0.25>0.5/0/0.75	1
SA-Mutation-Probability-Insert-Random-Waypoint	0,5	0.5/0.25>0.75>1/0	0>0.25>0.5>0.75>1	0.25>0.5>0>0.75>1	0>0.25>1>0.5>0.75	0>0.25>0.5>0.75/1	0>0.25>0.5>1>0.75	0.25>0.5>0>0.75>1	0.25
A-star-Random-or-Stochastic-Variant-Switch	True	True>>False	True>>False	False>True	True>>False	True>>False	True>>False	True>>False	True
Weight-Collisioncount	0,75	0.75>0.99>0.5>>0.25>0.001	0.5>0.75>0.25>0.99>>0.01	0.75>0.999>0.5>>0.25>>0.001	0.5>0.25/0.75/0.999>>0.0001	0.5/0.25>/0.75>0.999>>0.001	0.5/0.75>0.999/0.25>>0.001	0.75/0.5>/0.999>0.25>>0.001	0.5
Collision-Swapping-Penalty-Value	0,25	0.25>0.1>0>0.5	0.1>0.25>0>0.5	0.5>0.25>0.1>>0	0.1>/0>0.5>>0.25	0.25>0.1/0.5>>0	0.25>0.5>>0.1>0	0.1/0.25>/0.5>>0	0.25
Crossover-Variant-Switch	True	True>>False	True>Flase	True>False	True>False	True>False	True>False	True>False	True
Extra-Waypoint-Datasets	Set 4	Set3>Set1>Set4>Set2	Set3/Set1>Set2/Set4	Set3>Set2>Set4/Set1	Set3>Set1>Set2>Set4	Set3>/Set4>Set2/Set1	Set3>Set4>Set1>Set2	Set1>Set2/Set4>Set3	Set 3
Crossover-deletion-value	75%	75>100>25>50>0>False	75>100>0/25/False/50	25/0/100>75/50/False	100>50>25>False>75	75>100>50>False/0>25	75/100/>/25>0>False>50	100>75/50>25/False/0	100%
Mutation-deletion-value	75%	75>0>100/50>25	100>75>25>0>50	100>50/0>75/25	50>100>25>0>75	75>100>50/0>25	75/25>100>0>50	100>75>50>25>0	100%
MA-Population-Size	100	100>148/48>>28	148>100>48>28	148>100>48>28	148>28/48>100	100/148>48>28	148>100>48>28	148>100>48>>28	148
MA-Mutation-Probability	0,5	0.5>0.75>0/0.25>1	0.5/0.25>/1>0.75/0	1>0.75>0.25>0.5/0	1>0.75>0>0.25>0.5	0.5/0.25>1>0.75>0	0.75>0.5/1>0/0.25	0.75>0.25/1/0.5>/0	0.75
MA-Crossover.Probability	1	0.75>1>>0.5>0.25>0	1>0.75>0.5>0.25>>0	0.25>0.5/0.75>1>0	0.5>/0.75>/0.25>1>0	1/0.75>0.5>0.25>>0	1>0.75>/0.5>0.25>0	1>/0.75>0.5/0.25>>0	0.75
Number-of-Representatives	5	5>10/20	5>10>>20	10>5>>20	5/10>>20	5>10>>20	5>/10>>20	5>10>>20	5

Table 7: Parametrization Ranking table MO SACCGA

Optimizing the MO SACCGA									
					Maptypes				
Paramters	Startingvalue	empty-8-8.map	empty-16-16.map	maze-32-32-2.map	"random-32-32-20.map	random-64-64-20.map	"room-32-32-4.map	room-64-64-8.map	Resulting value
SA-Population-Size	28	20>28>40	40>20>28	28>40/20	40>28/20	28>40>20	40>28>20	20>28>40	28
Max-Number-of-Starting-Chromosones	1	1>3>5>2	5>1/2>3	5>1>3/2	2/1>3/5	3>/1/2>5	2/5>1>/3	5/3>1>2	1
SA-Crossover-Probability	0,75	0.25>0.75>/0.5>1>0	0.25/0>0.75>0.5>1	0.25>0>0.75>0.5>1	0.75>0>0.25/1>0.5	1>0>0.5/0.25>0.75	0.5>1/0.75>0.25>0	1/0.75/0>0.25>0.5	0.25
SA-Mutation-Probability-Shift-in-Neighborhood	0,75	0.75>1>/0>0.5>0.25	1>0.75>0/0.5>0.25	0.75/0.5>1>0.25>>0	1>0.75>0.5>0.25>0	1>0/0.75/0.5>0.25	1>0.75>0.5>0.25>0	0.75>0.5/0.25/0/1	0.75
SA-Mutation-Probability-Gene-Deletion	1	0.5/0.25>1>0>0.75	0.5>0.75>0.25/0>1	0/0.25>0.5>1/>0.75	0.5>/1>0.25>/0>0.75	0.5>1>0.25/0.75/0	0.75>0.25>0>/1>0.5	0>0.25>0.5>0.75>1	0.25
SA-Mutation-Probability-Insert-Random-Waypoint	0,25	0.75>0.25>1>0>0.5	0.25>0.5>0.75>1>>0	0.25>0.5>0.75>1>>0	0.25>0.5>0.75>0>1	0.5>0.25>0.75>1/0	0.25>0.5/0.75/0.5>>0	0.25>0.5>0.75>1>0	0.25
A-star-Random-or-Stochastic-Variant-Switch	True	True>>False	True>False	False>True	True>False	True>>False	True>False	True>>False	True
Weight-Collisioncount	Moop	-	-	-	-	-	-	-	MOOP
Collision-Swapping-Penalty-Value	0,25	0.1>0.25>0>0.5	0.1>0.25>0>0.5	0.25>0.1>0.5>>0	0.1/0.25>/0.5>0	0.1>0>0.25>0.5	0.1>0.25/0.5>>0	0.25>0.1>0.5>0	0.1
Crossover-Variant-Switch	True	True>/False	False>/True	True>/False	False>/True	True>False	True>False	False/True	True
Extra-Waypoint-Datasets	Set 3	Set3>Set2/Set4>Set1	Set1>Set2/Set3/Set4	Set2>Set4>Set1>Set3	Set3>Set1>Set2>Set4	Set4>Set3>Set2>>Set1	Set1>Set2/Set3>Set4	Set2>Set4/Set1>Set3	Set2
Crossover-deletion-value	100%	0>50>100>75>25	25>100>75/0>50	25>0>50/75>100	25>100/50>75/0	50>0>25>100>75	25>0>100>50>75	25/0>50>100/75	25%
Mutation-deletion-value	100%	75>0>50>100>25	75/50>100>25/0	100>75>0>/50/25	75>100>0>25>50	50/100>75>0>25	25>100>0>50>75	100>50>25/75>0	75%
MA-Population-Size	148	148>100>28>48	148>100>48>28	148>100/48>28	148/100/48>28	100>148>48>28	148>100>48/28	148>100>48>28	148
MA-Mutation-Probability	0,75	0.75/0.5>1/0.25>>0	1/0.75/0.5>0.25>>0	1/0.75>0.5>0.25>>0	1>0.75>0.5>0.25>>0	0.25>0.75>/0.5>/1>>0	1>0.75>0.5>>0.25>0	1>/0.75>0.5>0.25>>0	0.75
MA-Crossoverprobability	0,75	1>0.75>0>0.25>0.5	0.5>/0.75/1>0.25>0	0/0.25>0.75>0.5>1	1/0.75>0.5>/0.25>0	0>0.25/0.75>0.5>1	0.25>0/0.75>1/0.5	0.25>/0.5>/0.75>0>1	0.5
Number-Representatives	5	10>/5>20	5>10/20	5/10>20	5>10>20	5>10>20	5>10>20	5>10>20	5

Table 8: Parametrization Ranking table SO SACCGA

Optimizing the SO SACCGA									
		Maptypes							
Paramters	Startingvalue	empty-8-8.map	empty-16-16.map	maze-32-32-2.map	"random-32-32-20.map	random-64-64-20.map	"room-32-32-4.map	room-64-64-8.map	Resulting value
SA-Population-Size	28	20>>28>40	40>>20>28	40>28/20	20>40>28	28>20>40	20>28>40	40>28>20	28
Max-Number-of-Starting-Chromosones	1	3>5>1>2	5>/3>1>2	3>/5>1>2	1>2>3>5	1>2>3>5	1/2/3>5	5>3/2>1	3
SA-Crossover-Probability	0,75	0>0.25>0.75>1>/0.5	0.5>0.25>0.75/0>1	0/0.25/0.5>0.75>1	1>0.5>0.75>0.25>0	0.75>0>1>0.5>0.25	0.5>0/0.75>0.25/1	0>0.25>0.75>0.5>1	0.25
SA-Mutation-Probability-Shift-in-Neighborhood	0,75	0.5>0.75>0/0.25>1	0.5>1>0.75>0>0.25	0.75>1>0.5>0.25>0	0.25>1>0.75>0.5>>0	0.25>0.75>0.5/0>1	1>0.75>0.5>0.25>0	1>/0.75>0.5>0.25>0	0.75
SA-Mutation-Probability-Gene-Deletion	1	0.5>1>0.75>0>0.25	0.5>0.25>0.75>0>1	0>0.25>0.5>0.75>1	0.5>1>0.25>0.75>0	1>0.75>/0.25>0.5>0	0>0.75>/1>0.5>0.25	0>0.25>0.75>/1/0.5	0.5
SA-Mutation-Probability-Insert-Random-Waypoint	0,25	0.5>0.25/1>0.75>>>0	1/0.75>0.5>0.25>>0	0.25>0.5>0.75>1>>0	0.25>/0.5>0.75>1>>0	0.25>0.5/0.75>1>>0	0.5/0.25>0.75>1>>0	0.25/0.5/0.75>1>>0	0.25
A-star-Random-or-Stochastic-Variant-Switch	True	True>>False	True>False	False>True	True>>False	True>>False	True>>False	True>False	True
Weight-Collisioncount	0,5	1>0.75>0.5>0.25>0	0.75>0.5>1>0.25>0	0.75>1>0.5>0.25>0	0.25/0.5/0.75>1>>0	0.25>0.5>0.75>1>0	0.5>0.75>1>0.25>>0	0.75>1>0.5>0.25>0	0.75
Collision-Swapping-Penalty-Value	0,25	0.1>0.25>0>0.5	0.1>0.25>0>0.5	0.25>0.5>0.1>0	0.5>0.25>0.1>0	0.25>0.5/0.1>0	0.25>0.1>0.5>0	0.25>0.5>0.1>0	0.25
Crossover-Variant-Switch	True	True>False	False>True	False>True	True>False	True>False	True>False	False>True	True
Extra-Waypoint-Datasets	Set 3	Set1>/Set3>Set4>Set2	Set4>Set1>Set2/Set3	Set4/Set2>Set1>/Set3	Set3>Set1>Set4>Set2	Set3>Set2>/Set4>Set1	Set3>Set1>Set4>Set2	Set4>Set2>Set1>Set3	Set4
Crossover-deletion-value	100%	75>100>50>0>25	0>100>50>75/25	25>50/0>100>75	75>50/0>/100>25	75>100/25>50>0	75>25>0>100>50	25>0>50>75/100	75%
Mutation-deletion-value	100%	75>0>100/50>25	0>25>75/100>50	100>25>75/0>50	75>0>/100>25/50	50>/100>25>75/0	50>25>75>100>0	100/75>50>25>0	75%
MA-Population-Size	148	100>/148>48>28	100>/148>48>28	148>100>48>28	100>/148>48>28	148>100>48>28	148>100>48>28	148>100>48>28	148
MA-Mutation-Probability	0,75	0.5>1>/0.75>0.25>0	0.25>0.5>0.75/1>0	0.75>0.5>1>0.25>0	0>0.5>0.25>0.75>1	0.75>1>/0.5>0.25>0	1>0.5>0.75>0.25>0	0.75>1>0.25>0.5>0	0.5
MA-Crossover Probability	0,75	0.75>0>/1>0.5>0.25	0.5/0.75>1>0>0.25	0.75>0>0.25>0.5>1	0.5>0.25>0.75>1>0	0.75>1>0.5/0.25>0	1/0.75>0.5>0.25>0	0.25/0.75>0.5>0/1	0.75
Number-Representatives	5	5>10>20	5>10>20	5>10>20	10>20>5	20>10>5	5>10>20	5>10>20	5

Table 9: Ranking tables Icon Legend

Icon	Explanation
a>>b:	a is by much better than b
a>b:	a is better than b
a>/ b:	a is a little better than b
a/b:	a and b are equal

Analysis of results of the parametrization: For the population size of the MA, a population of 148 individuals clearly seems to get the best results. A population of 100 sometimes seems to work better for some maps. Since a MA population of 100 uses less function evaluations it might converge faster. The reason why a population of 148 individuals is getting better results might be that the decision space of the MA grows exponential with the number of agents. For the mutation probability of the MA and the crossover probability of the MA values between 0.75 and 0.5 appear to work well.

The population size of the SAs seems to converge the fastest in both variants with the value of 28. From the two crossover variants, the second crossover operator was chosen which places the crossover point in front of the values that are close together. The small crossover probability of the SA shows that even that crossover operator is not working effectively. The selected crossover probability value is 0.25 for both variants and was closely followed by a crossover probability value of zero, which shows that there is potential for a better crossover operator. The crossover deletion value is at 25% for the MO SACCGA and 75% for the SO SACCGA. The 0% value seems to work well for some maps, especially for the MO SACCGA. This, coupled with the high degree of randomness of the algorithm, does not allow a clear statement to be made as to whether the deletion value in the crossover is useful for this algorithm. The fact that the dominant value for the mutation probability for the Mutation in Neighborhood operator is at 75% indicates that the mutation operator works well. The selected values for the probability of the Gene Deletion mutation operator of the SA are 25% and 50% for both variants. However, for both variants the decisions were difficult since for many maps the best value was different and the result is more of a compromise between the results for the different maps. The reason might be that the original effect of making the route smoother is reduced due to the adding of new waypoints. Deleting a through A* added waypoint doesn‘t shrink the path length but increases the number of A* traversed nodes. The Mutation Random Waypoint operator seems to be very important for the algorithm. Although this operator greatly increases the number of A* traversed nodes - far more than the other operators do - the value of zero leads to very bad results in contrast to the other values for all tested environments. However, a high value does not produce good results either for most maps. The operator takes care of two things: It introduces new waypoints and deletes waypoints in the process. The algorithm works best with a high maximum deletion value. In the Comparison of the stochastic A* variant with the deterministic A* variant, the stochastic one seems to dominate. The stochastic variant is for every map except the maze map in all three parametrization runs by far better. This might be the case because the maze environment gives less room for the stochastic algorithm to change

a path to a not yet used path. But it must be kept in mind that the way the algorithm is structured, the way the algorithm was tested and the way the problem was defined benefit the stochastic variant:

- The algorithm creates extra waypoints when creating long paths. This shrinks the randomness effect of the stochastic variant and makes the algorithm more reliable.
- The increase in complexity of the stochastic variant is not taken into account in the comparison.
- The simplification of the cost of vertical movements to one time step leads to a higher amount of fastest routes between two waypoints and therefore increases the effect of the stochastic variant.

For the number of representatives the lowest value of five representatives dominates. The number of representatives has two main effects in this algorithm. First, the decision space of the MA increases/decreases proportionally with this value. Secondly, the objective collision count of the SA is determined by checking how many collisions an individual has with the representatives of the other agents. The tests for the Collision-Swapping-Penalty-Value C_{SPV} lead for maps with less obstacles to smaller values and for maps with more obstacles to higher values. For the SO SACCGA, a value of 0.25 works best, while for the MO SACCGA a value of 0.1 seems to dominate. For the Extra-Waypoint-Datasets, which determines how many waypoints are added by the A* algorithm depending on the path length between two waypoints, the MO SACCGA works best with the Set, which creates the most waypoints, and the SO SACCGA works best with a Set, which creates a moderate number of waypoints. As for the weight of the collision count objective of the SO SACCGA 0.75 works best. A close second best is the value 0.5. The value 0.0001, which changes the optimization to a lexicographical optimization by prioritizing the path length objectives, achieves the worst results for all maps, while the value 0.9999, which prioritizes the collision count works fine. This shows that the collision count variable works well in terms of optimizing the three objectives. The weight for the path length is thereby 0.25.

The parameterization done in this chapter does by no means lead to an optimal parameter setting. Neither all combinations of settings are compared with one another, nor have enough test runs been carried out to make statistically stable statements. Furthermore, the settings were only applied to seven test problems with only one combination of start and target points used. However, using a parametrization, which leads to an optimal setting, would have gone beyond the scope of this thesis.

5.3 Comparison of the Single-objective Slave algorithm and the Multi-objective Slave algorithm

In this chapter, the results of the SO SACCGA and the MO SACCGA comparison are shown and analyzed. For the comparison, both variants were tested on the twelve different environments mentioned in chapter 5.1. Three different starting- and target-point-sets for each environment and different number of agents were used. For each problem setting (environment + scenario + agent count) 31 runs were done. For the three starting and target point sets the first three even scenarios from mapf.info [19] were used by taking the starting and ending point combinations from the front (for a 40 agents scenario 1 problem, the first 40 starting and target points from scenario 1 for the selected environment are taken). Starting with two agents for each problem the number of agents is increased by one until an agent count was reached for which it was expected that the quality of the solutions gets too low. The pareto front of each run was normalized and the GD and IGD values were calculated by using a fake pareto front, which was formed out of all pareto fronts found for this problem. The GD and IGD values were calculated as described in Ishibuchi et al. [17].

Out of the GD and IGD values for the 31 runs of each variants, the median and the interquartile range were calculated. These values were used to decide which variant does better for a given problem. Ties were determined using the Mann-Whitney-U-test with a significance level of 0.05 using the Mann-Whitney-U function from the SciPy module [41]. If the tests did not lead to a tie, the variant which found the fake pareto front in 100% of the cases was crowned as a winner for the problem. If no variant found the fake pareto front in 100% of the 31 runs, then the variant with the higher median was determined as the winner. If the median was the same, the visualized box plots were analyzed to determine the winner.

Additionally, the runs were compared by assuming that a decision maker weighs the objectives of the MA with the same weights, which were used to form the weighted sum in the SA. The collision count objective of the SA is designed to optimize the overlaps objective of the MA and the path length objective to optimize the makespan and sum of costs objective. Since the weights, which work best for the SA, are the 0.75 for the collision count objective and the 0.25 for the path length objective, the decision maker weighs the overlaps-objective with a weight of 0.75 and the makespan and sum of costs each with a weight of 0.125. The solution of the pareto front of each run, which minimizes the weighted sum is used for comparison. The weighted sum values were used to calculate the medians and interquartile ranges for both variants for each problem to decide, which variant solves the problem better. Ties are again

determined using the Mann-Whitney-U-test with a significance level of 0.05 and the winner was chosen like for the IGD and GD comparison.

Since both variants have the same population sizes for the SA and the MA it is assumed that the number of function evaluations is the same for both variants. To simplify things the results after 100 generations are compared against each other.

Figure 17 to 19 show grouped bar plots of a Win-Lose-Tie-Table (table 16 appendix section C) showing the results of the comparison in regards to the GD values, IGD values and weighted sums.

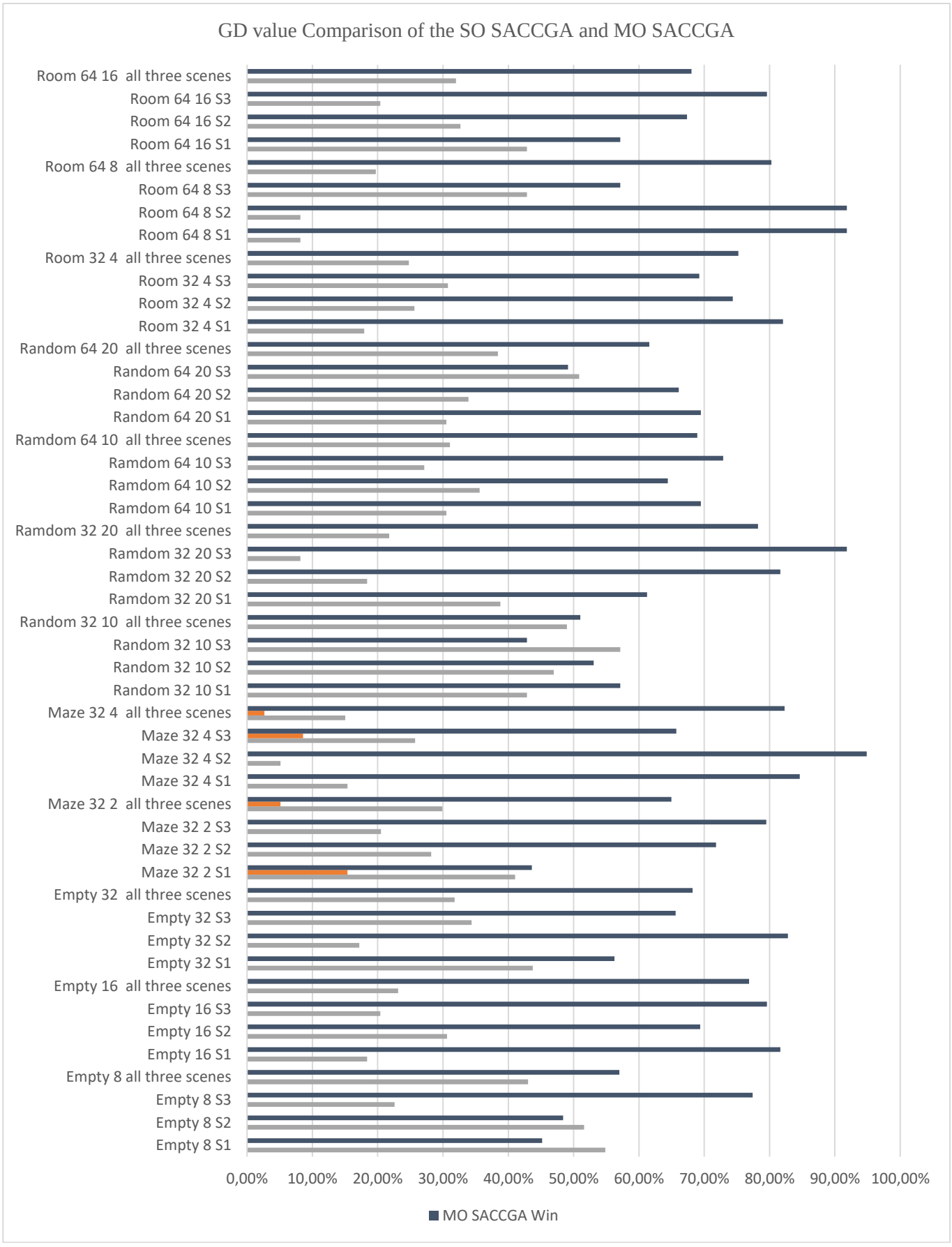

Figure 17: Grouped bar plot of the Win- Lose- Tie- Table of the comparison of the GD values of the SO SACCGA and the MO SACCGA

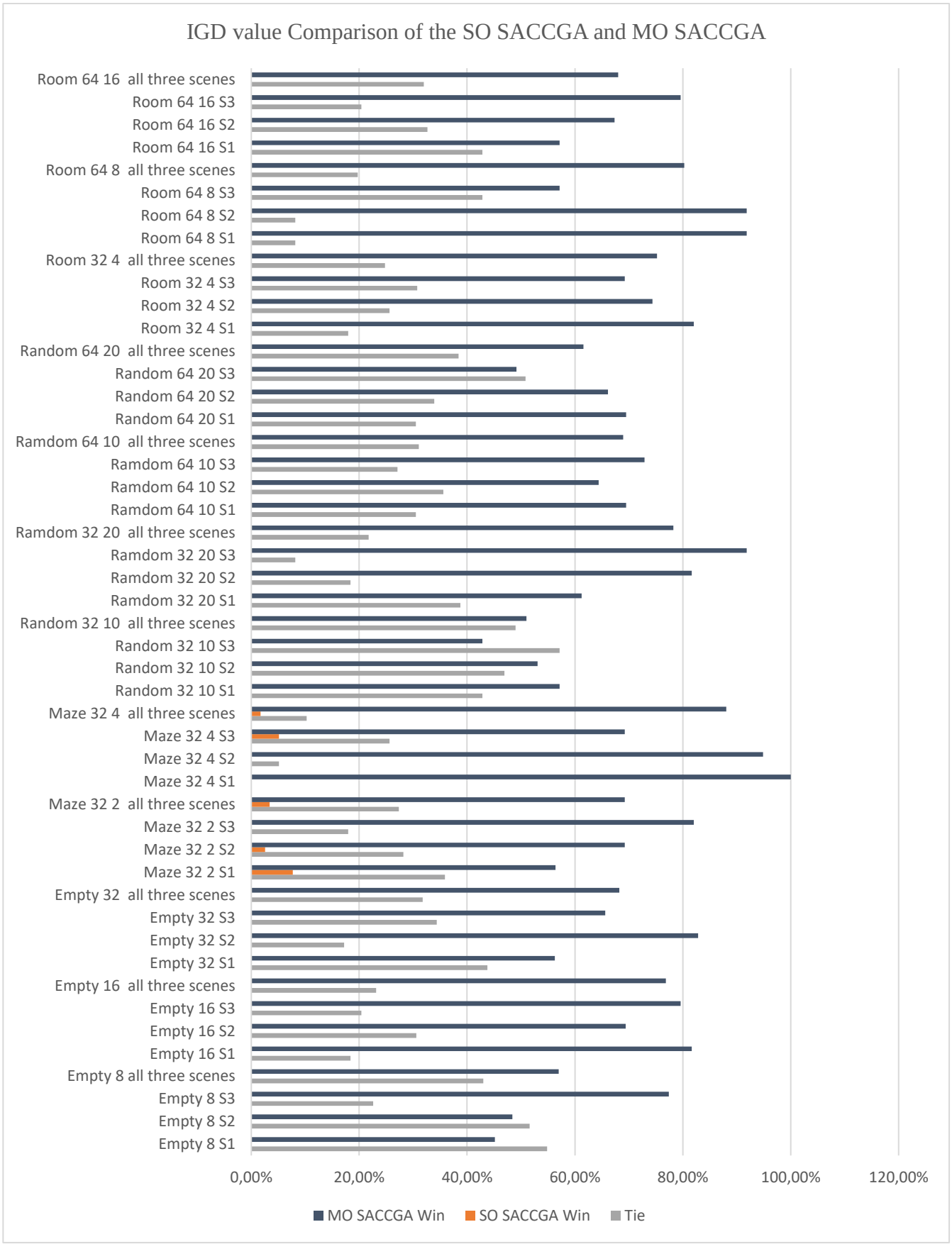

Figure 18: Grouped bar plot of the Win- Lose- Tie- Table of the comparison of the IGD values of the SO SACCGA and the MO SACCGA

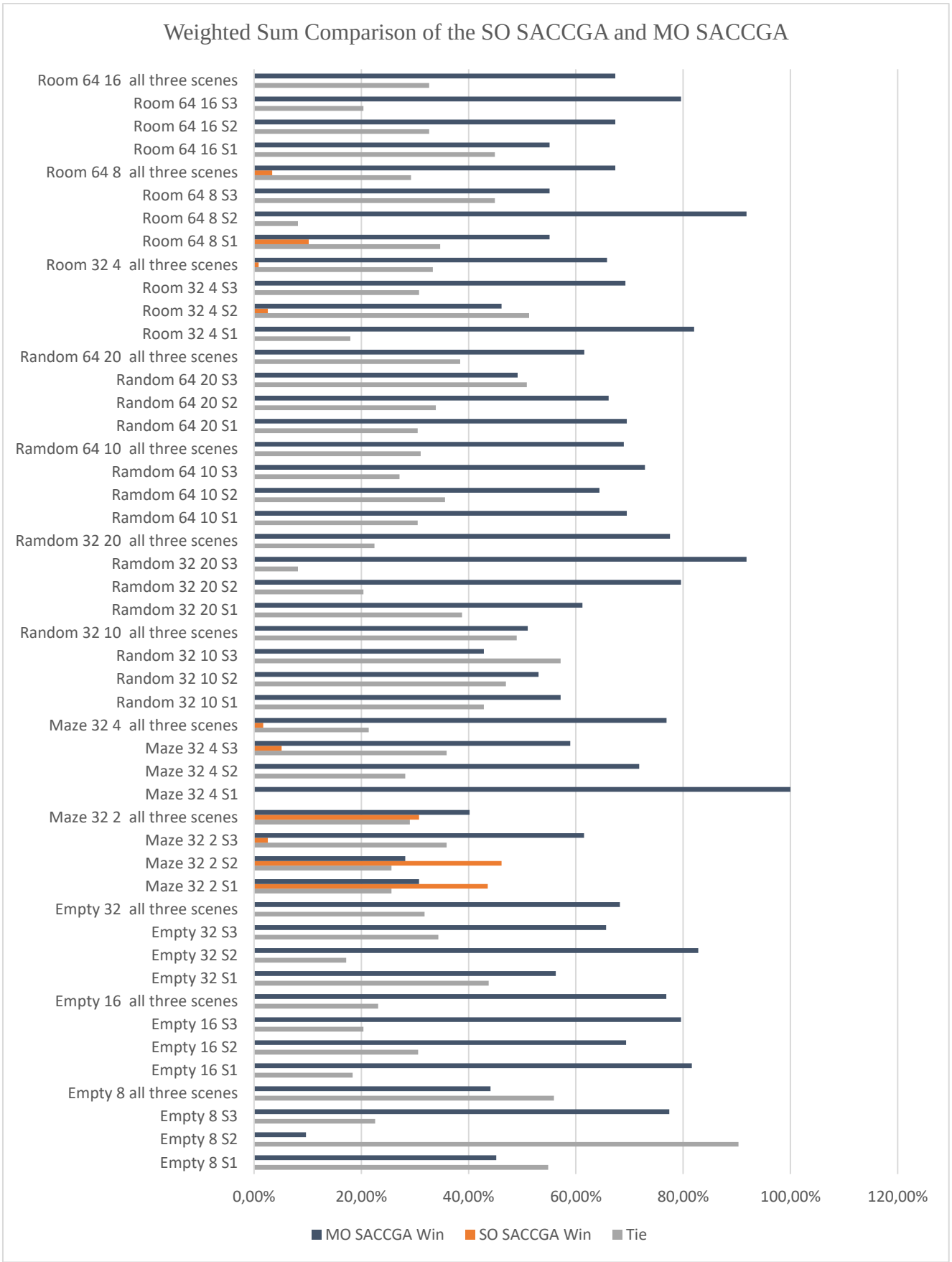

Figure 19: Grouped bar plot of the Win- Lose- Tie- Table of the comparison of the weighted sum values of the SO SACCGA and the MO SACCGA

First of all, these results show that the MO SACCGA works overall better for all scenario-map-combinations regarding GD and IGD. For three of the four map types, the SO SACCGA is not able to solve one problem better than the MO SACCGA. Only when it comes to the problem types Maze 32-32-2 with the scenarios one and two and Maze-32-32-4 with the scenario three, the SO SACCGA seems to work better for a few problems. But the results also indicate that the MO SACCGA solves most of the problems better in these environment-scenario combinations. There is only a small difference between the GD and IGD results. Only for the map types for which the SO SACCGA is a little more competitive, the MO SACCGA tents to work slightly better in the comparison of the IGD values than in the comparison of the GD values. Accordingly, one can conclude that the SA of the MO SACCGA passes on solutions that ensure more diversity than the SA of the SO SACCGA. The large proportion of ties is caused by problems with a small number of agents, for which both versions are able to find the pareto front in 100% of the cases.

In the case, where the decision maker weighs the objectives with the same weight the SO SA uses for its objectives, the results show that the MO SACCGA again gets overall better results than the SO SACCGA. For the empty and random map types, the MO SACCGA solves every problem as good as or better than the SO SACCGA. The SO SACCGA is only able to get a few more ties for these map types. In contrast to the GD IGD results, the SO SACCGA is able to solve many problems for the map types room and maze better than the MO SACCGA. Especially for the map Maze 32-32-2 with the scenarios one and two, the SO SACCGA solves more problems in a better way than the MO SACCGA does. For this particular scenario map combinations, the MO SACCGA solves in respect to the weighted sums the problems with a smaller number of agents better. The SO SACCGA solves the problems with a higher agent count better like illustrated in figure 20. However, even for those environments the MO SACCGA works far better than the SO SACCGA if all problems are considered. For Maze-32-32-4, the SO SACCGA gets even worse results than in the GD IGD comparison. For all other environments, however, the SO SACCGA performs either just as well or better as in the GD IGD comparison.

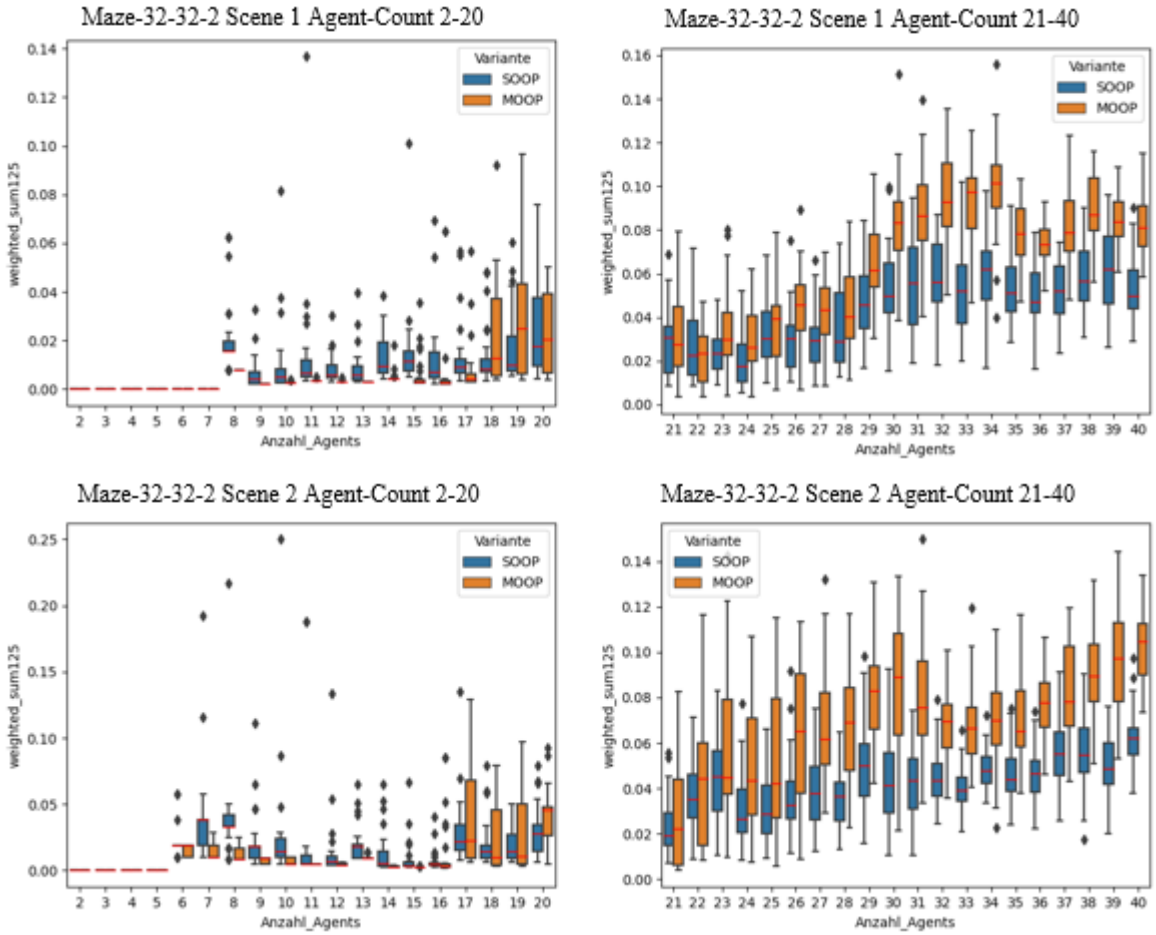

Figure 20: Boxplot-comparison of weighted sum values (minimization) of SO SACCGA and MO SACCGA for the problem of Maze-32-32-2 Scene 1 and 2

To analyze why the SO SACCGA is worse than the MO SACCGA and where the algorithm fails, it is examined what kind of problems lead the SO SACCGA and the MO SACCGA to not being able to find the whole pareto front. Table 10 shows with at which agent count the SO SACCGA and the MO SACCGA versions are still able to find the complete fake pareto front in over 90%, over 75% and over 50% of the runs for a particular map scenario combination. Additionally, the column „Single Dominant solution“ shows until which number of agents the pareto front consists of just one solution, which dominates every other solution.

Table 10: Number of agents, at which the algorithm results get worse

Max number of agents	Maptype and Scenario	Single dominant solution	MO SACCGA			SO SACCGA		
			90%	75%	50%	90%	75%	50%
32	Empty 8 Sz1	26	18	26	26	17	17	26
32	Empty 8 Sz2	25	12	13	16	12	13	16
32	Empty 8 Sz3	8	23	23	26	8	8	8
50	Empty 16 S1	till the end	14	20	28	11	13	14
50	Empty 16 S2	23	23	39	39	16	16	19
50	Empty 16 S3	49 (unsure)	36	36	45	10	25	35
65	Empty 32 Sz1	till the end	65	65	65	33	36	51
65	Empty 32 Sz2	till the end	65	65	65	11	36	36
65	Empty 32 Sz3	till the end	42	53	56	25	26	26
40	Maze 32 2 Sz1	7	13	13	14	7	7	7
40	Maze 32 2 Sz2	5	5	5	15	5	5	5
40	Maze 32 2 Sz3	8	9	9	10	8	8	8
40	Maze 32 4 Sz1	never existed	0	12	15	0	0	0
40	Maze 32 4 Sz2	3	3	3	3	3	3	3
40	Maze 32 4 Sz3	5	5	5	5	5	5	5
50	Random 32 10 Sz1	till the end	46	48	50	31	31	33
50	Random 32 10 Sz2	49 (unsure)	40	47	48	25	25	35
50	Random 32 10 Sz3	45 (unsure)	35	35	38	26	26	38
50	Ramdom 32 20 Sz1	23	30	32	39	21	23	23
50	Ramdom 32 20 Sz2	10 (unsure)	10	10	12	10	10	10
50	Ramdom 32 20 Sz3	37 (unsure)	18	18	18	4	18	18
60	Ramdom 64 10 Sz1	till the end	24	24	46	20	21	24
60	Ramdom 64 10 Sz2	till the end	29	29	50	21	21	30
60	Ramdom 64 10 Sz3	till the end	31	38	42	17	17	25
60	Random 64 20 Sz1	54	43	43	43	19	22	31
60	Random 64 20 Sz2	18	41	53	60	18	18	18
60	Random 64 20 Sz3	56	41	50	52	24	40	44
40	Room 32 4 Sz1	8	11	11	11	8	10	11
40	Room 32 4 Sz2	11	17	17	19	11	11	11
40	Room 32 4 Sz3	12	22	22	22	14	16	20
50	Room 64 8 Sz1	5	14	14	14	5	5	5
50	Room 64 8 Sz2	5	10	14	15	5	5	5
50	Room 64 8 Sz3	22	22	22	22	22	22	22
50	Room 64 16 Sz1	26	31	32	34	21	26	26
50	Room 64 16 Sz2	19	21	22	22	19	19	19
50	Room 64 16 Sz3	11	17	29	32	11	11	11

What can be seen is that for many map-scenario combinations the performance does not decrease granularly as the number of agents increases, but decreases sharply at some point by increasing the agent count by one. The word *performance* in this context refers to the quality of the solution and not to the computational cost. These cases are often found when the set of pareto optimal solutions changes from a single-dominant solution to several solutions. This shows that some features of the problem, which leads the problem to have a pareto front consisting of more than one solution, leads also the algorithms to have a worse performance. Furthermore, this particular case often only affects the SO SACCGA. To analyze these effects even farther table 11 lists all the cases, in which by increasing the number of agents by one a high drop in the percentage of runs occurs, in which the whole pareto front was found. Moreover, they are categorized according to the number of versions they affect and whether or not the pareto front consists of more than one solution.

Table 11: Problems, which show high drops in performance categorized by how many versions they affect and if the pareto front consists of more than one solution before and after the drop

Category	Map	Szenario	Agentcount before drop in performance to after	Percentage of finding whole pareto front before agentcount increase and after in %	
High drops in performance both version, just one solution in pareto front				**SO SACCGA Values**	**MO SACCGA Values**
	Empty 8-8	2	16 to 17	70-25	70-38
	Ramdom 32 20	3	18 to 19	87-6	100-32
	Random 64-10	1	24 to 25	74-16	100-67
	Random 64-20	1	43 to 44	22-3	96-41
High drops in performance both version, more than one solution in pareto front					
	Empty 16-16	2	39 to 40	0-0*	80-22
	Room 32-32-4	1	11 to 12	70-3	100-16
	Room 32-32-4	3	22 to 23	22-0	100-6
High drops in performance one version, just one solution in pareto front					
	Random 32-32-10	2	25 to 26	96-58	100-100
	Random 32-32-10	3	26 to 27	100-65	100-97
	Random 64 10	2	21 to 22	100-58	100-100
	Random 64-10	3	17 to 28	93-35	100-97
	Random 64-20	1	19 to 20	100-54	100-100
High drops in performance one version, more than one solution in pareto front					
	-	-	-	-	-
High drops in performance both version, by transition from one solution in pareto front to more solutions					
	Empty 8-8	1	26 to 27	54-0	80-22
	Random 32 20	2	10 to 11	97-0	100-51
	Maze 32-32 2	2	5 to 6	100-12	100-45
	Maze 32-32 4	1	nothing to 2	(100)-6	(100)-41
	Maze 32-32 4	2	3 to 4	100-0	100-19
	Maze 32-32-4	3	5 to 6	100-6	100-0
	Room-64-64-8	3	22 to 23	100-0	100-42
High drops in performance one version, by transition from one solution in pareto front to more solutions					
	Empty 8-8	3	8 to 9	100-0	100-100
	Random 32 20	1	23 to 24	87-29	100-100
	Random 64 20	2	18 to 19	100-12	100-100
	Maze 32-32-2	1	7 to 8	100-9	100-100
	Maze 32-32-2	3	8 to 9	100-19	100-97
	Room 32-32-4	1	8 to 9	100-71	100-100
	Room 32-32-4	2	11 to 12	100-12	100-97
	Room-64-64-8	1	5 to 6	100-10	100-100
	Room-64-64-8	2	5 to 6	100-0	100-83
	Room-64-64-16	1	26 to 27	84-35	100-100
	Room-64-64-16	2	19 to 29	97-19	100-100
	Room-64-64-16	3	11 to 12	100-6	100-100

It becomes clear that there is a large amount of cases, in which a high drop in performance occurs. These cases either affect both the SO SACCGA and the MO SACCGA or only the SO SACCGA. The MO SACCGA is not affected by any major drop in performance, which does not also affect the SO SACCGA. However, there are cases, which only affect the performance of the SO SACCGA. One can see that most of the high drops in performance happen in cases in which the pareto front changes from consisting of one solution to multiple ones and most of them only affect the SO SACCGA.

It can also be seen that each scenario of the room maps appears in the table. The SO SACCGA, in particular, suffers from high drops in performance in room map-problems when the pareto front changes from a single-dominant one to multiple solutions. The two Maze Maps are also represented with all scenarios. It can be seen that the Maze-32-32-4 always causes problems

for both variants, as well as when the pareto front changes from a single solution to multiple ones. In addition, such a change happens for both maze maps while the agent-count is still low. Maze-32-32-4 Scenario 1 presents the most striking problem, as both versions show low performance, starting with an agent count of two. Empty-8-8 of the Empty maps is represented with all scenarios, two of which also occur when the pareto front switches from a single to multiple solutions. With the exception of Empty-16-16 scenario 2 and Empty-8-8, the Empty maps show only granular changes in the performance during the increase of the agent-count. The Random maps scenarios with 10% obstacles are represented in the list but do not show the greatest performance drops. The low amount of high performance drops in the large Empty and the Random maps with 10% obstacles can be explained. The high performance drops happen mostly when the pareto front changes from consisting of one solution to multiple ones. Meanwhile all examined problems from the scenarios of Empty-32-32 and Random-64-64-10 as well as scenario 1 from Random-32-32-10 and scenario 1 from Empty-16-16 never changed from having just one dominant solution in the pareto front. For Empty-16-16 scenario 3 and Random-32-32-10 scenario 2 and 3 it is also very likely that a single dominant solution existed, but the two variants were not able to find them. This can be the case because the performance of the variants decreased sharply towards the end. In contrast to the Random environments with 10% obstacles, all three scenarios of Random-32-32-20 and two of the scenarios of Random 64-64 20 are represented in the list, which means that the 10% obstacles more leads to cases which the algorithm handles badly.

In this chapter, many problems and a few problem-characteristics were identified, which the SO SACCGA and partially the MO SACCGA solve poorly. In the following, the reasons behind the high drops in performance are further analyzed.

Chapter 5.3.1 examines whether the difference in the calculation of the swapping conflict in the SA and the MA lead to some of the problems mentioned.

In chapter 5.3.2 the representatives of the SA of both versions are further analyzed to find out if they lead to some problems mentioned in this chapter.

5.3.1 Further analyzation of the SO SACCGA and MO SACCGA results: Swapping Conflict Calculation

The swapping conflict in the SA is not precisely calculated but rather estimated. The SA just penalizes an individual *i* with a higher collision count fitness value for every representative *r* of the other agents, which are in the same vertex at time step *x+1*, in which individual *i* was at

time step *x*. In contrast to that, the MA calculates the real number of swapping-conflicts. Additionally, the separate parametrization of the SO SACCGA and the MO SACCGA lead to different penalty values for the swapping conflict, which might have an influence on the results of the comparison.

To analyze the swapping-conflict calculation the Maze-32-32-4 Scenario 1 Number of Agent 2 problem was analyzed. Although the problem only has two agents, it belongs to the cases in which both variants have low performances like shown in table 11.

The illustration 21 shows a solution for the problem visualized. For the following analyzation, the blue path in figure 21 belongs to the *blue agent* and the green path belongs to the *green agent*. This solution is an excerpt from the "fake pareto front" found where the makespan values and the sum of costs are optimal and the overlaps value has the value 1. That means the agents take the fastest route and collide with each other once, which results in the objective values: Makespan = 49.0, sum of costs = 97.0, overlaps = 1.0. In figure 21 it can be seen that the conflict is a swapping conflict. The agents can only avoid each other if one extends its route because the vertices, at which the agents cross, are mandatory for the fastest route given the start- and target points. Figure 23 shows the mandatory points of the fastest routes around the conflict point. This means that the true pareto front must consist of at least two solutions. The second solution of the pareto front requires that the agent in blue avoids the agent in green like to see in figure 22. This results in the objective values: Makespan = 49.0 sum of costs = 98.0, overlaps = 0.0. Although it does not matter what the route to get there looks like, as long as it is one of the fastest ways, the agents have no leeway in the area around the conflict point.

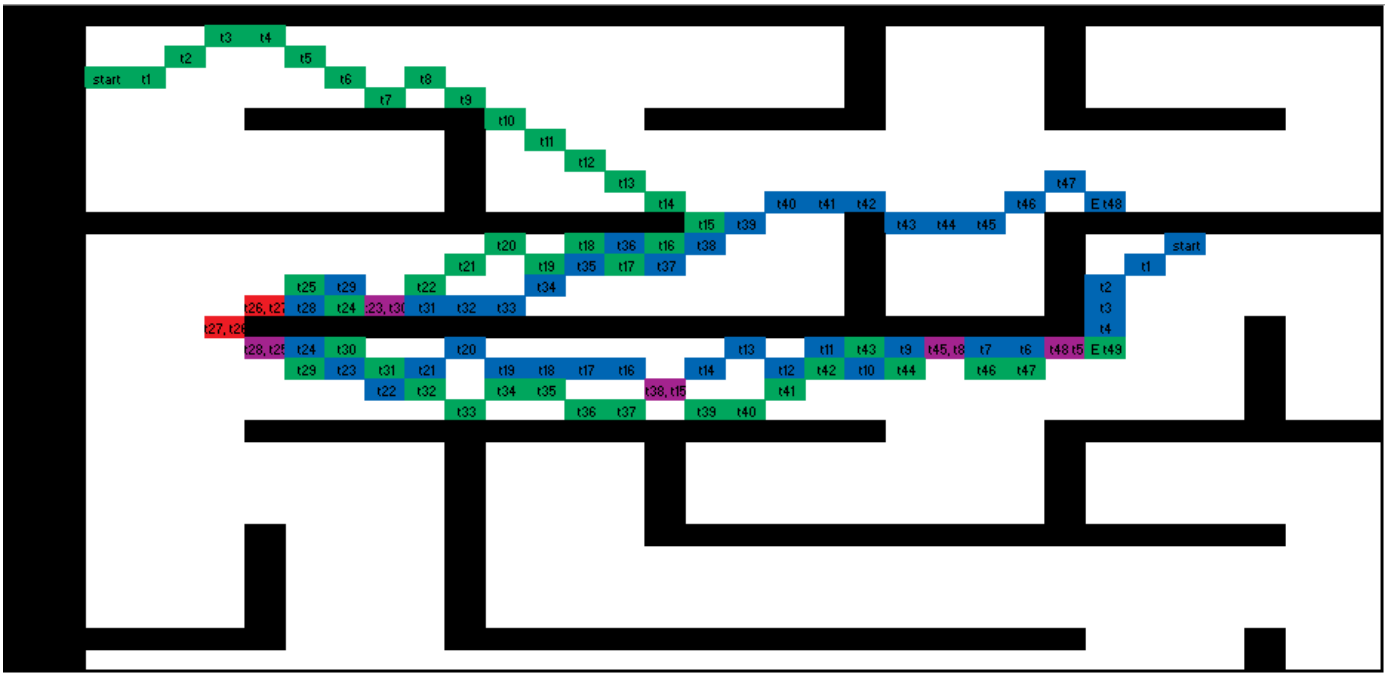

Figure 21: Maze-32-32-4, scenario 1, agent-count 2: Paths with Swapping Collision

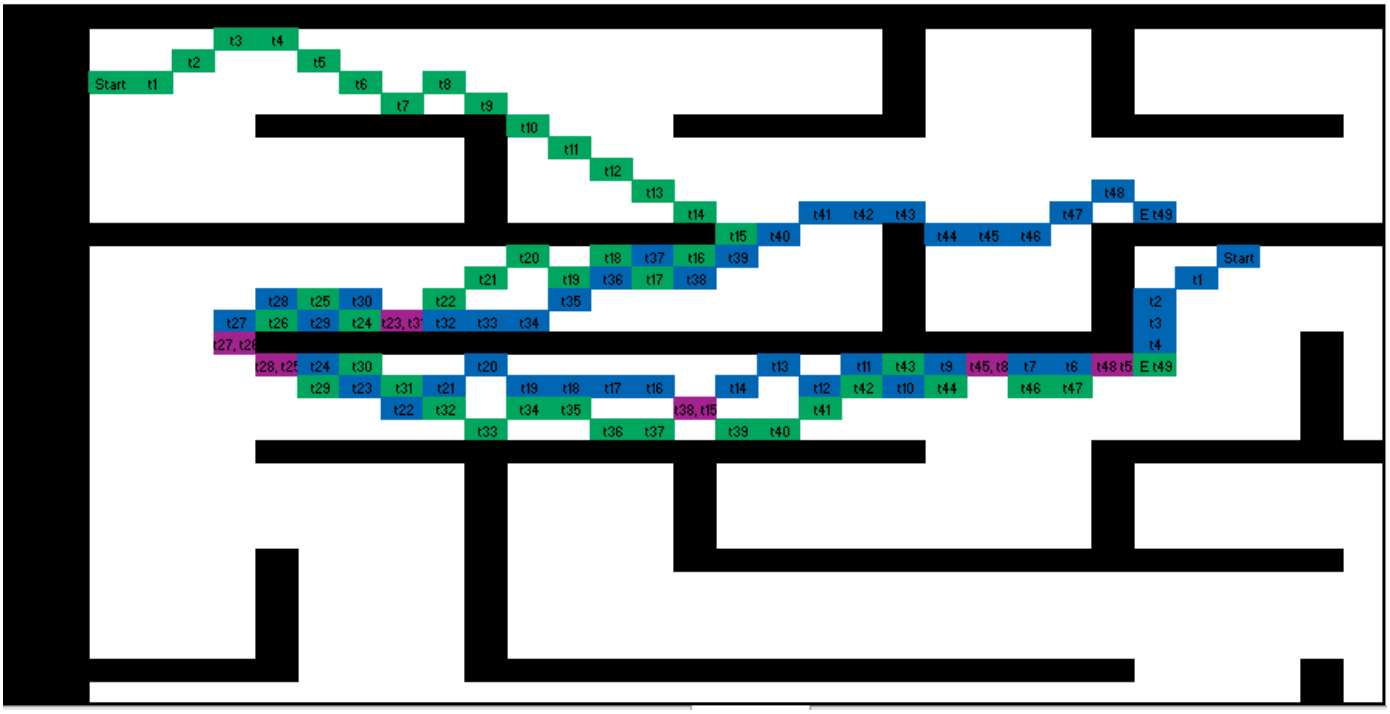

Figure 22: Maze-32-32-4, scenario 1, agent-count 2: Paths to avoid Swapping Collision

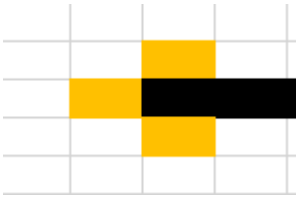

Figure 23:Maze-32-32-4, scenario 1, agent-count 2:Mandatory vertices for fastest path both agents

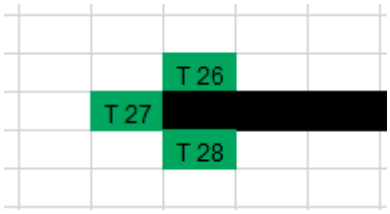

Figure 24:Maze-32-32-4, scenario 1, agent-count 2: Agent Green path for pareto optimal solution

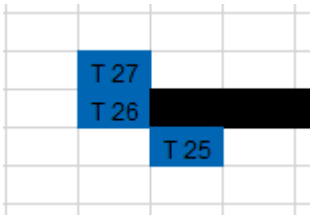

Figure 25:Maze-32-32-4, scenario 1, agent-count 2: Agent Blue path for pareto optimal solution with avoiding

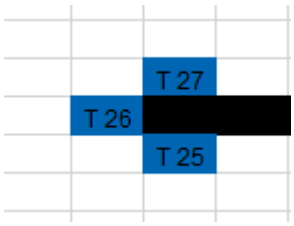

Figure 26: Maze-32-32-4, scenario 1, agent-count 2: Agent Blue path for pareto optimal solution without avoiding

If we assume that there is only one representative in green, whose route around the conflict point looks exactly as it does in the illustration 24 the “with avoiding”(figure 25) and “without avoiding” (figure 26) routes for the blue agent are evaluated in the SA, then:

- The route "with avoiding" receives a path length of 49 and a collision count of one time the swapping collision penalty value C_{SPV}, because agent green moves into the vertex at time step 27 at which agent blue was at time step 26.
- The route “without evading” receives a path length of 48 and a collision count of one time collision penalty value C_{SPV}, since agent green also moves here into the vertex at time step 27, at which agent blue was at time step 26.

This means that the route without evasion dominates the route with evasion in the SA. They both have the same value in the collision count objective, although one solution leads to collision and the other does not. Therefore, the algorithm would not be able to find the route with a robustness value of zero anymore. The algorithm is much more likely to produce a

solution without evasion because individuals are initialized with a low amount of waypoints and because in the first generation, individuals are only evaluated by their path length.

This would not be the case if the swapping conflicts were calculated precisely. To prove that another calculation of the collision count of the SA was programmed. In this version, the $Collisionlist_{VX}$ has been modified.

The $Collisionlist_{VXA}$ is a three dimensional matrix with the dimensions Vertices V, Time steps X, and Agents A. Instead of showing how many agents are in a particular vertex at a particular time step, it stores a pointer of the representatives in a list on the vertex and time step index. As a result, it can refer to the agents that are in a particular vertex at a particular time step. This way it is possible to determine exactly how many representatives of the other agents are at a time step x + 1 in the same vertex as an individual *i* at the time step *x* and how many of the same representatives are in time step *x* in the same vertex as the individual *i* at the time step *x + 1* during the evaluation in the SA as seen in algorithm 9. Algorithm 10 and 11 show how the $Collisionlist_{VXA}$ is created and updated. In the new version, the collision count $CLC\_precise$ of an individual is increased by one for every representative with which it has a swapping collision like seen in equation 5.1:

$$\delta(\mathrm{x})=\begin{cases}1, x = 0\\ 0, x \neq 1\end{cases}$$

$$\varepsilon(\mathrm{x})=\begin{cases}1, x = 2\\ 0, x \neq 2\end{cases}$$

$$CLC\_precise = \sum_{x=0}^{|\pi_i|}\sum_{j=1,i\neq j}^{k}\sum_{r=1}^{\mathrm{N_R}}(\delta(\pi_i(\mathrm{x}) - \pi_{jr}(\mathrm{x})) + \varepsilon(\delta(\pi_i(\mathrm{x}) - \pi_{jr}(\mathrm{x}+1))+ \delta(\pi_i(\mathrm{x}+1) - \pi_{jr}(\mathrm{x}))) \quad (5.1)$$

With $|\pi_i|$ being used as maximal time step value since time and path length are unitless, discrete, and increase by the same value at each step.

Algorithm 9: Evaluate collision count with precise collision swapping conflict calculation

Input: Population of agent A: $SAPop_A$, $Collisionlist_{VXA}$, $C_{SwappingPenaltyValue}$

Output: Fit_C

for π_i in $SAPop_A$:

| $Fit_{C,i}$=0

| **for** x=0; x< π_i; x++ :

| | $Fit_{C,i} = Fit_{C,i}$ + getLength($Collisionlist_{VXA}$ [x][π_i[x]])

| | **for** π_j in $Collisionlist_{VX}$ [x][π_i[x+1]]:

| | | **if** π_i[x] == π_j [x+1]:

| | | | $Fit_{C,i} = Fit_{C,i}$ +1

Algorithm 10: Remove Representatives from Collision list with precise collision swapping conflict calculation

Input: Representatives of Agent A: R_A, $Collisionlist_{VXA}$

Output: $Collisionlist_{VXA}$'

for π_i in R_A:

| **for** x=0; x< π_i; x++ :

| | **for** π_j in $Collisionlist_{VXA}$[x]

| | | **if** π_j==π_i:

| | | | **del** π_j #delete pointer

Algorithm 11: Collisionlist update/Creation with precise collision swapping conflict calculation

Input: Representatives of Agent A: R_A, $Collisionlist_{VX}$# if Gen one $Collisionlist_{VX}$=[]

Output: $Collisionlist_{VX}$'

CollisionlistVertices=[]*$N_{NumberOfVertices}$

for π_i in R_A:

| **for** x=0; x< π_i; x++ :

| | **if** length($Collisionlist_{VX}$)<x:

| | | $Collisionlist_{VX}$.append(copy.copy(CollisionlistVertices))

| | $Collisionlist_{VX}$[x][π_i[x]].append(π_i) #append pointer

This calculation was used in the MO SACCGA and the SO SACCGA for each problem of the category "high drops in performance both version, by transition from one solution in pareto front to more solutions"; the same category the Maze-32-32-4 scenario 1 agent count 2 problem was belonging to. Table 12 shows the results for the new swapping calculation in comparison to the results of the old swapping calculation in the context of finding the whole set of pareto dominant solutions.

Table 12: Using the new swapping calculation for the problems of the category "high drops in performance both version, by transition from one solution in pareto front to more solutions"

Mapname	Sc.	Number of Agents	Pareto_Front_found SO SACCGA with new swapping calculation	Pareto_Front_found SO SACCGA with old swapping calculation	Pareto_Front_found MO SACCGA with new swapping calculation	Pareto_Front_found MO SACCGA with old swapping calculation
Empty-8-8.map	1	27	0%	0%	26%	22%
Maze-32-32-4.map	1	2	90%	6%	100%	41%
Maze-32-32-4.map	2	4	48%	0%	19%	19%
Maze-32-32-4.map	3	6	55%	6%	26%	0%
Maze-32-32-2.map	2	6	3%	12%	97%	45%
Random-32-32-20.map	2	11	0%	0%	48%	51%
Room-64-64-8.map	3	23	0%	0%	58%	42%

While the performance for the maps Empty-8-8, Random-32-32-20 and Room-64-64-8 remained largely the same, the performance for all four maze problems generally improved.

This demonstrates that the difference in the calculation of the swapping conflicts between the SA and the MA lead to a few of the categorized problems.

Additionally it was checked if the problems of the category "high drops in performance for one version (SO SACCGA), by transition from one solution in pareto front to more solutions", which only affected the SO SACCGA, are also solved with better performance when using the precise calculation of the swapping conflicts.

Table 13: Using the new swapping calculation for the problems of the category "high drops in performance for one version (SO SACCGA), by transition from one solution in pareto front to more solutions"

Mapname	Scenario	Number of Agents	Pareto_Front_found SO SACCGA with new swapping calculation	Pareto_Front_found SO SACCGA with old swapping calculation
Empty-8-8	3	9	100%	0%
Random-32-32-20	1	24	74%	29%
Random-64-64-20	2	19	0%	12%
Maze-32-32-2	1	8	25%	9%
Maze 32-32-2	3	9	6%	19%
Room-32-32-4	1	9	80%	71%
Room-32-32-4	2	12	3%	12%
Room-64-64-8	1	6	100%	10%
Room-64-64-8	2	6	0%	0%
Room-64-64-16	1	27	58%	35%
Room-64-64-16	2	20	83%	19%
Room-64-64-16	3	12	54%	6%

The outcomes show that results for the Empty-8-8, Random-32-32-20 and the Room-64-64 environments got far better and that some of the high drops in performance were traced back to the swapping calculation. This shows that the SO SACCGA was more affected by the swapping conflict calculation than the MO SACCGA. The reason behind this might be that differences in the evaluation between subpopulation and overall solution might have less negative influence when the subpopulations are solved multi-objectively, as this provides more diversity. Another explanation could be that the higher penalty value of the SO SACCGA was leading the SO SACCGA to obtain worse solutions. Although it might be interesting to find out what the reason behind this effect is, it will not be further analyzed in this thesis.

The precise calculation of the swapping collisions was also used for the problems of Empty-8-8, Random-32-32-20, Room-32-32-4 and Maze-32-32-2 scene 1 to find out if the calculation also had an effect on the comparison of the two variants. This was done because the collision count parameter of 0.25, which the SO SACCGA uses, worked better for the maze and room –

maps and the collision count parameter of 0.1 worked better for empty and random maps (like seen in table 6 to 8). For the room and maze maps, the SO SACCGA was able to solve some problems in respect to the weighted sum value better than the MO SACCGA, while the MO SACCGA was better for the other maps than the SO SACCGA in respect of every used metric.

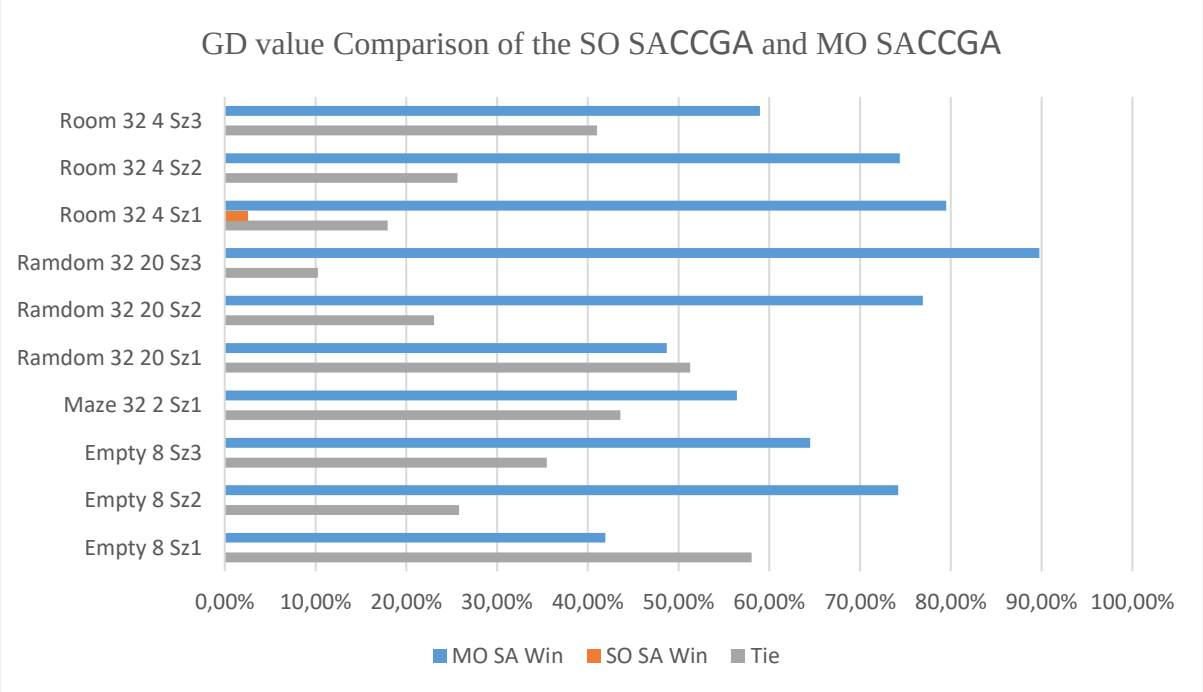

Figure 27: Grouped bar plot of the Win- Lose- Tie- Table of the comparison of the GD values of the SO SACCGA and the MO SACCGA with precise swapping conflict calculation

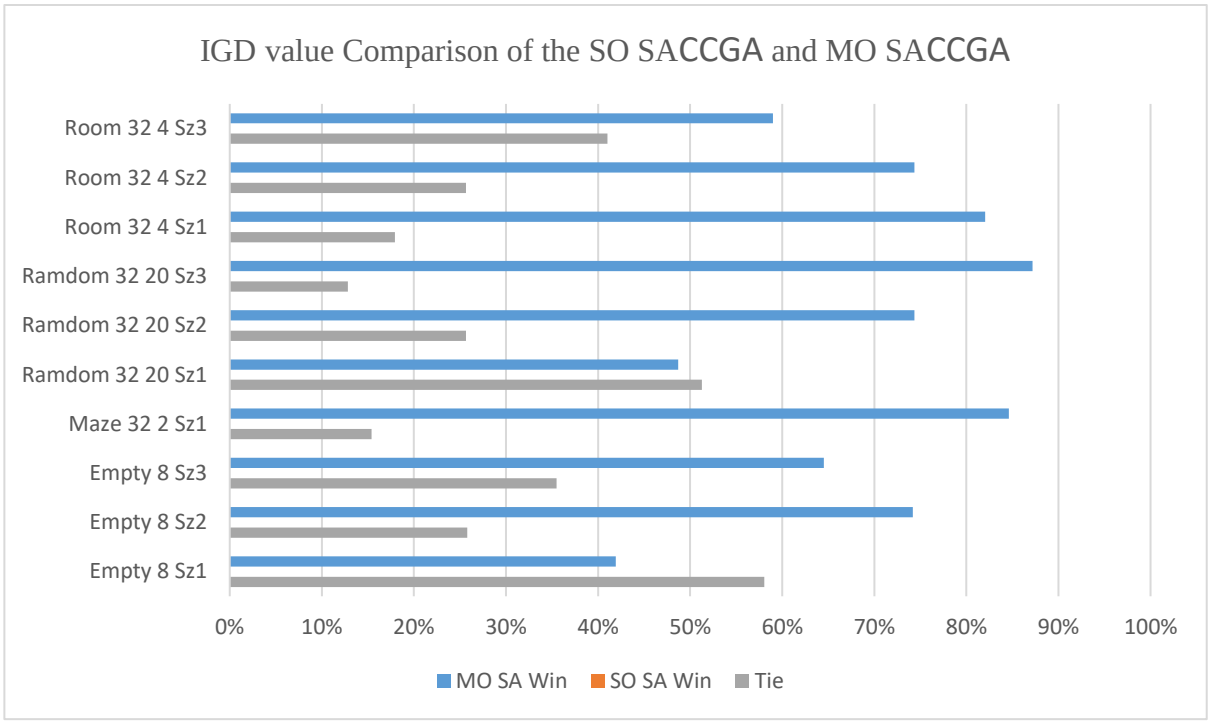

Figure 28: Grouped bar plot of the Win- Lose- Tie- Table of the comparison of the IGD values of the SO SACCGA and the MO SACCGA with precise swapping conflict calculation

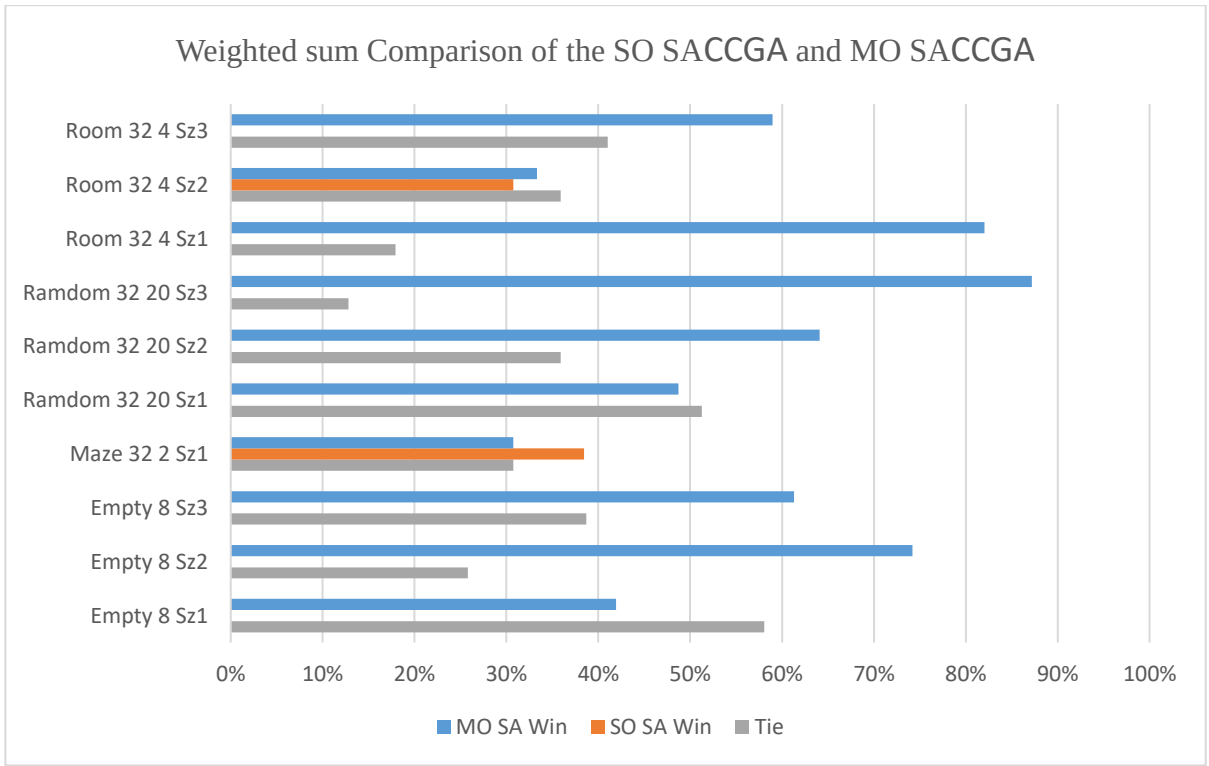

Figure 29: Grouped bar plot of the Win- Lose- Tie- Table of the comparison of the weighted sum values of the SO SACCGA and the MO SACCGA with precise swapping conflict calculation

The results shown in figure 27 to 29 are similar to the results without the precise swapping collisions. In terms of GD and IGD values the MO SACCGA version dominates. In terms of

weighted sum values, with the same weights used as in the SA of the SO SACCGA version, the SO SACCGA version solves the problems of the Maze-32-32-2 and the Room-32-32-4 scenario 2 with high agent count better, while the MO SACCGA solves problems for the same map scenario combinations with lower number of agents better.
It is to conclude that the MO SACCGA solves most problems better than the SO SACCGA. For some problems, in which the environment is more complex (maze/ room map with high number of agents), the SO SACCGA is better in finding solutions in terms of the same weighing of the objectives as in the SA. However, in most of the examined problems this is not the case.

5.3.2 Further analyzation of the SO SACCGA and MO SACCGA results: Local Optima

As shown in table 12 the results for the examined "Maze-32-32-4, scenario 1, agent-count 2" – problem got much better. However, although the problem is not too complex, the SO SACCGA does still not find all pareto optimal solutions in every run. To assure that this only affects the SO SACCGA, 101 additional runs were performed for the same test setup. The results show that the MO SACCGA is able to find the whole pareto front 132 times out of the overall 132 runs. The SO SACCGA is only able to find the whole pareto front 118 times out of the 132 runs. It is certain that the 11% of the runs, in which the SO SACCGA is not able to find the whole pareto front, embody a problem of the algorithm that seems to only affect the SO SACCGA. All repetitions from the 11% were able to find the pareto optimal solution with the fastest routes and one collision: Makespan=49; sum of costs=97; overlaps=1 (figure 21). However, they were not able to find the pareto optimal solution in which agent blue evades the agent green: makespan=49; sum of costs=98; overlaps=0 (figure 22). Instead, solutions were generated in which agent green evades agent blue, which leads to the same value in the sum of costs objective but to a worse value in the makespan objective: makespan=50; sum of costs=98; overlaps=0 and makespan=50; sum of costs=99; overlaps=0. The SO SACCGA ends up in a local optima. This is particularly interesting in the context of the thesis, because the solution with the fitness values makespan=49, sum of costs=98 and overlaps=0 is also the optimal solution for the calculation of the weighted sum value, where the decision maker weighs the objectives with the same weights as used in the SO SA. For one of the runs of the 11% the paths of the representatives slave algorithm at generation 99 were visualized in the conflict-enabling part of the map in figure 30a. Although the whole paths of the representatives are not completely the same, they act the same in the part of the map in, which the collisions occur. The representatives show no diversity in this crucial part of the path. In this state of the run, it is

impossible to plant a new solution into the populations, which would be integrated as representative even if this new solution is part of the global optimum because the paths of the representatives of both agents are fitted perfect to each other. Therefore, it is impossible for the SO SACCGA to leave the local optima. A visualization of the representatives of the SO SA of a run (figure 30b), which found the whole pareto front was found, shows the same phenomenon. Although the whole pareto front was found in this run, the paths of the representatives do not show any diversity around the conflict point. In both SO SA runs, the solution where the agents take the fastest routes and collide once is not represented in the representatives anymore. This solution was found in generation one. This is very likely to happen, since many solutions start with a small number of waypoints and in the first generation solutions are only evaluated by their path length, because no representatives exist at that point. This means the SO SA finds parts for a pareto optimal solution by the first generation and converges away from this solution. If the problem made this solution harder to obtain it would not be able to find it anymore. For the comparison, the paths of the representatives of the MO SACCGA around the conflict area were visualized too. The representatives of both agents of the MO SACCGA run show diversity around the conflict area. Also all solutions from the pareto front are represented in the representatives. Several solutions that would lead to local optima are represented as well.

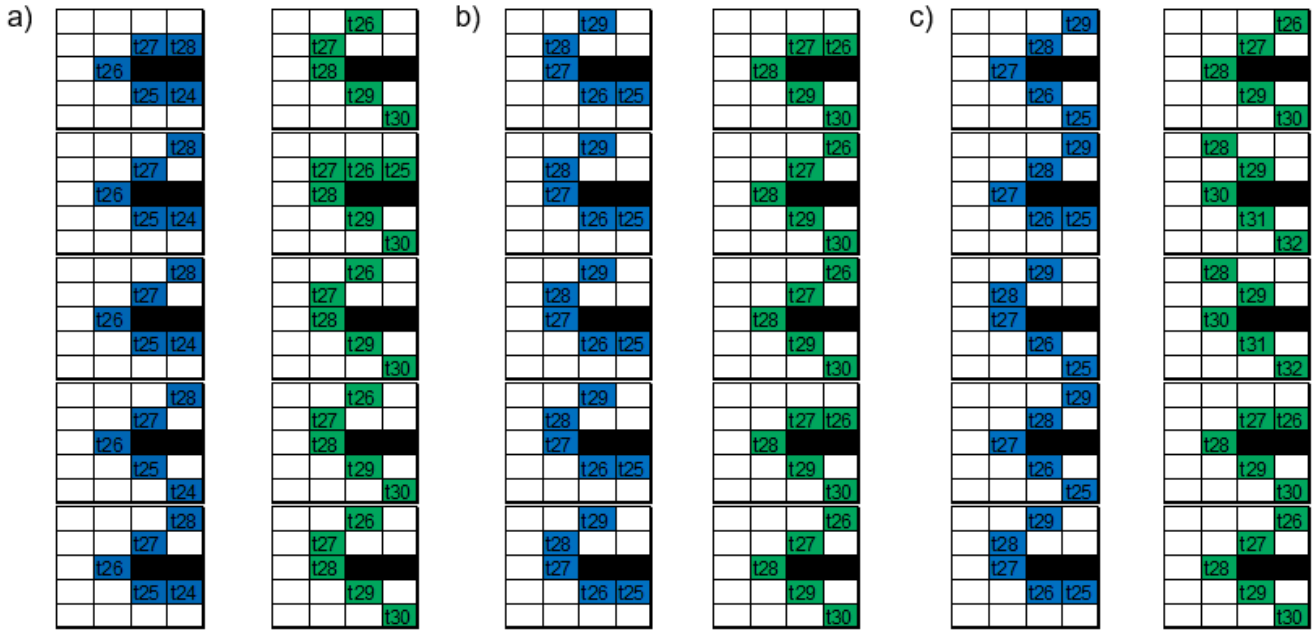

Figure 30: Paths of representatives around the conflict for the "maze-32-32-4 scene 1 number of agents 2" –problem

a)SO SACCGA run, where parts of the pareto front were not found; b) SO SACCGA run, where whole pareto front was found; c) MO SACCGA run, where whole pareto front was found

In conclusion, the lack of diversity in the SA leads to the SO SACCGA being partially unable to grasp the entire pareto front and ending up in a local optima. Additionally, the SO SA converges from some solutions away, which are part of the pareto front. The MO SA in contrast has more diversity in the paths of the representatives, which leads it to find the entire pareto front more easily.

In Wiegand [44] it was already stated that co-evolutionary algorithms often find themselves even in simple problems in local optima and Ahmed and Deb [1] stated that solving single agent pathfinding multi-objectively often leads to better a solution (or at least feasible ones) than if solved single objectively. Solving the subpopulations in coevolution multi-objectively, leads less often to local optima, was somewhat similarly stated in Bucci and Pollack [3]. Solving the subpopulation of every agent in MAPF multi-objectively and avoiding local optima with this procedure is a new finding as far as the literature research of this work goes.

5.4 Summary of the Experiments and discussion

This chapter first explained the MAPF.info benchmarks, on which the various tests were carried out. The MO SACCGA and the SO SACCGA versions were initially parameterized, so that the variants could be compared with one another. The SO SACCGA and MO SACCGA were then compared with regard to their GD and IGD values as well as weighted sum values. It turned out that the MO SACCGA dominated on most of the problems according to all used metrics. The MO SACCGA did work even better in the comparison of the IGD values than in the comparison of the GD values. Only for some maze map problems and room map problems with a high agent count did the SO SACCGA seem to be better in respect to the weighted sum value than the MO SACCGA.

To find out why the SO SACCGA was worse on most of the maps and where the limitations of both algorithms lay, problems and problem properties were sought, leading to poorer performance on the versions. It was found out that the performance often does not decrease granularly when the agent count increases, but suddenly by the increase of the agent count by one. Problems found that showed this feature were categorized and a correlation was identified between the change of the pareto front consisting of one solution to more than one solution and the drops in performance.

One of the problems, which showed a high drop in performance and a pareto front consisting of more than one solution, showed that the drops in performance were related to the different calculation of the swapping collision between the SA and the MA. To test whether this affects

several problems, the calculation of the SA was changed. The new calculation was applied to several problems that had high drops in performance when increasing the agent count by one. Several of these problems were solved with far better performance. Therefore, it is very likely that the difference in the calculation of the SA and the MA lead to the high performance drops. Some of these low performances, which got better after the new calculation only affected the SO SACCGA. It is therefore plausible that the difference in calculation of the collisions between the SA and the MA affected the SO SACCGA more than the MO SACCGA. The theories have been voiced in this thesis that the reason behind this might be:

- That differences in the evaluation between subpopulation and overall solution might have a less negative influence when the subpopulations are solved multi-objectively, because of the higher diversity in the subpopulation
- Or that the higher penalty value was leading the SO SACCGA to obtain worse solutions.

The correctness of the theories were not analyzed further. Additionally, with the precise calculation of the swapping collision the SO SACCGA and MO SACCGA were again compared against one another. The results showed that the MO SACCGA is still better in all maps except the maze map in terms of weighted sum and dominated in terms of the other metrics for all other environments. The SO SACCGA was able to solve high agent counts for more complex maps (partially the room map and the maze map) better in terms of the weighted sum than the MO SACCGA.

The behavior of the representatives of the SO SACCGA, which leads the algorithm to fall into a local optima, was analyzed based on one problem. It was found out that the representatives of the weighted sum approach converge to one solution and only find the other solutions of the multi objective MAPF problem by chance on the way. This shows that the weighted sum approach in the subpopulation is not fitting to be part of a multi objective MAPF solver. This also explains the bad IGD values compared to the MO SACCGA version. Furthermore, it was found out that the representatives do not show diversity in fitness and therefore do not offer different solutions in crucial parts of the path. It was also shown that the SO SACCGA lands in local optima even in less complex problems because the SA was solved single-objectively. For this particular test problem, this did not happen to the MO SACCGA and the representatives show diversity. It is possible that the representatives of the SO SA will quickly adapt to one another due to the lack of diversity of the routes and thus end up in a local optimum without a way out. The finding on the behaviour of the representatives of the SO SA are based on multiple runs for one problem. Although it is highly likely that this phenomenon applies to multiple of the problems, it was not proven.

The experiments in this thesis agree with the thesis 1 of this work: “MAPF multi-objective optimization with co-evolution works better if the subpopulations of the agents are optimized multi-objectively than if the subpopulations are optimized single-objectively with a weighted sum approach”. The reasons are most likely that the low diversity in the subpopulations of the weighted sum approach leads it to not finding the whole pareto front and falling into local optima. Additionally, the fact that the weighted sum approach converges from pareto optimal solutions away shows that a weighted sum approach should not be used to evaluate subpopulations, if the whole solution should be optimized multi-objectively.

The experiments disagree for the most part the thesis 2 of this work: “Using a co-evolutionary approach, if the decision maker weighs the objectives of the Multi-objective MAPF problem with the same weights the objectives of the subpopulations of the agents are weighted using a weighted sum approach, then this weighted sum approach works better than optimizing the objectives of the subpopulations of the agents multi-objectively.” The experiments showed that the multi-objective solving of the subpopulations works best for most environments even if the decision maker uses the same weighs as the weighted sum approach uses. The SO SACCGA appeared to be better for some complex maps with high agent count. The reason why the SO SACCGA is worse for many maps could be the lower diversity in the representatives of the SO SA, which lead into the local optima. For the Maze-32-32-4 scenario 1 agent count 2 problem, this was certainly the case. It is also questionable, if all parts of a problem are optimally solved according to certain weights, that the overall solution is then optimized according to the same weights. That could be very problem-dependent. Why the SO SACCGA was better in solving more complex environments with high agent count sometimes better than the MO SACCGA was not found out in this thesis. Unfortunately, it is not proven that the same results will occur if different objectives are used or parts of the algorithm are changed.

As far as the quality of the algorithm is concerned, the MO SACCGA is able to solve many of the problems well compared to the SO SACCGA. Table 10 shows the limitations, at which the algorithm is no longer able to find the entire fake pareto front reliable. Even so, optimality is not the goal of the algorithm, since this is not an optimal solver. However, the algorithm is very complex and requires a large number of function evaluations, as demonstrated with formula 4.11. A comparison with similar work in the same field was not done. Furthermore, the correct Pareto front could not be determined, since there are still no optimal solvers for MAPF with for the three objectives used in this thesis.

6 Conclusion and Future Work

This chapter summarizes the results of this thesis (chapter 6.1) and provides suggestions for future work (chapter 6.2).

6.1 Conclusion

In this thesis, the basics of cooperative coevolution were conveyed. Through literature search, an overview of the MAPF solver, the co-evolutionary MAPF approaches and the solver for the multi-objective MAPF problem is given. Furthermore, findings from comparing the solving of the single agent pathfinding problem single-objectively and multi-objectively are described.
A cooperative co-evolutionary algorithm was presented in this master thesis. This algorithm has a subpopulation for every agent, which are optimized by a genetic slave algorithm. Using the best solutions of every agent a genetic master algorithm searches for the best combinations of the paths to optimize the objectives, makespan, sum of costs and overlaps of the multi-agent MAPF problem. The slave algorithm optimizes the objectives path length, which optimizes the solutions makespan and sum of costs as well as the collision count objective, which is supposed to optimize the overlaps objective.
Two variants of the slave algorithm were implemented: One variant, which solves the subpopulations single objectively with a weighted sum approach (SO SACCGA), and another variant, which solves the subpopulations multi-objectively using the non-dominated sorting algorithm (MO SACCGA). The variants were parameterized and compared to each other. The comparison showed that MO SACCGA works better in terms of the GD and IGD metrics for nearly all the examined problems. Therefore, the experiments approve the content of the thesis 1. Some of the reasons why the SO SACCGA is worse than the MO SACCGA for multi-objective optimization of the MAPF problems are:

- The low diversity in the subpopulations of the weighted sum approach, lead to a fall into local optima
- The weighted sum approach converges away from some pareto optimal solutions, which do not satisfy the weights.

The MO SACCGA showed diversity in its subpopulations and the ability to avoid local optima. This also shows that a weighted sum approach should not be used to evaluate subpopulations, if the whole solution should be optimized multi-objectively.

The MO SACCGA works better than the SO SACCGA in terms of having a decision maker, who weighs the objectives the same way, in which the subpopulations of the agents of the SO SACCGA weigh their objectives. The SO SACCGA appears to be better for some complex maps with high agent count. The experiments disapprove the content of the thesis 2. The reason why the SO SACCGA is worse for many problems is most likely the low diversity in the subpopulations, which leads into the local optima. Why the SO SACCGA was sometimes better in solving more complex environments with high agent count than the MO SACCGA is not further analyzed.
The experiments also showed that solving the subpopulation of the agents multi-objectively avoids local optima and that the SO SACCGA is more affected by differences between the objectives of the subpopulation and the objectives of the whole solution.
Since there are no optimal solvers with the corresponding answers, it is not clear how good the quality of the solutions produced by the algorithms is. However, it is safe to say that the MO SACCGA is better than the SO SACCGA in terms of solving the MAPF problem multi-objectively. The algorithm is although computationally expensive.

6.2 Future Work

This work raised many research questions, which were out of the scope of this thesis:

6.2.1 Research topics in MAPF with genetic algorithms:

In this thesis, a stochastic A* algorithm was used to connect waypoints to each other. The parametrization showed that the stochastic A* algorithm works better for most environments than the deterministic one. However, it must be mentioned that the algorithm by adding waypoints and the test setup by neglecting the additional computational expenses and the movement simplifications greatly favoured the stochastic algorithm. One question that would be interesting for future work is whether and under what conditions a stochastic A* algorithm for connecting waypoints is better than a deterministic A* algorithm for genetic algorithms in MAPF. Many areas of application of MAPF require that MAPF concepts can also be used in Euclidean space. Further questions would be, if a stochastic A* algorithm is advantageous and how one can transfer these advantages of stochastic solutions to the Euclidean space, on which there are only rarely several fastest solutions.

6.2.2 Research topics in MAPF with co-evolution:

The findings in Wiegand [44] showed that Co-evolution often leads to local optima. In Panait et al. [29] and in Bucci and Pollack [3] approaches were presented on how to avoid these local optima. The approach in Panait et al. [29] needs information about the best values. In Bucci and Pollack [3] it was suggested to use every test available to evaluate an individual and treat the evaluation multi-objectively with every test being one objective. Tests in this context are all solutions, which can be created by combining the solutions of the subpopulations. Forming every solution out of k agents and evaluating every individual by all these tests is computationally expensive. In this thesis, an agent was evaluated by its cooperation with the representatives of all other agents. The single objective solver of the slave algorithm was still leading into local optima. The multi objective solver was able to avoid these local optima. Since there are no studies on how local optima in co-evolution in MAPF can be avoided, this should be done in future works. The multi objective solver of this thesis is a good starting point for these works.

6.2.3 Research topics in multi-objective MAPF

Another interesting topic for future work is the multi-objective solving of MAPF. One thing, which was missing for this thesis, was an efficient optimal solver for multi-objective MAPF. Using overlaps as one of the objectives makes the problem more complex since agent path combinations, which lead to collisions, can still be part of a pareto optimal solution. If this solver can be adjustable for different objectives then the solution space of different MAPF problems can be further analyzed.

The last research suggestion would be similar to the work done in Ahmed and Deb [1] in which the quality of solutions between the single objective and the multi-objective solving of single agent pathfinding was compared to each other. It is possible that multi-objective MAPF genetic algorithms can find better solutions for single objectives than single objective MAPF genetic algorithms because the multi-objective MAPF genetic algorithms might keep more diversity in the population. A comparison of the two algorithms would be interesting for future work.

Appendix

Section A: Input Parameter

Regarding the SA:

SA Population Size $N_{SA\ Individiuals}$: Defines the number of individuals in every SA subpopulation.

Maximal Number of Starting Genes $N_{\mathrm{Startinggenes}}$: Defines the number of maximal starting genes for agents in initialization. The algorithm chooses a random number between zero and $N_{\mathrm{Startinggenes}}$ as number of waypoints for every fresh-initialized individuals.

SA Crossover-probability: Defines the probability of the crossover operator of the SA.

SA Mutation-probability Shift-in-Neighborhood *SAMUTBP2*: Defines the mutation probability of the Shift-in-Neighborhood-Mutation operator of the SA.

SA Mutation-probability Gene Deletion *SAMUTBP1*: Defines the mutation probability of the Gene-Deletion-Mutation operator of the SA.

SA Mutation-probability Insert Random Waypoint *SAMUTBP3*: Defines the mutation probability of the Insert Random Waypoint Mutation operator of the SA.

A* Variant Switch: If true, the algorithm uses the stochastic A* version. If false, the algorithm uses the deterministic A* version.

SA MOOP Switch: If true, the SA uses the multi-objective algorithm. If false, the SA uses the single-objective algorithm.

Weight collision count $weight_2$: Defines the weights used in the weighted sum approach of the SO SA. The weight-collision count variable is the weight of the objective collision count. Since only two weights are used, the path length objective $weight_1$ is automatically $weight_1 = (1 - weight_2)$.

Collision-Swapping-Penalty-Value C_{SPV}: Defines how much the collision count objective is increased for each representative of the other agents which the individual might have a swapping collision with. A possible swapping conflict occurs if the representatives of the other agents are at the same vertex at time step *x+1* as the individual at time step *x*.

Crossover-Variant-Switch: Defines which of the two one point-crossover-variants is used. If true, the crossover operator chooses randomly one waypoint from one parent and the nearest waypoint to this waypoint from the second parent. If false, the crossover operator sets the crossover point in both parents into the same spot in dependence on the length of the parents.

Crossover-deletion-value CDV: Defines how many waypoints are deleted by the crossover operator.

Mutation-deletion-value *MDV*: Defines how many waypoints are deleted by the mutation operator Insert Random Waypoint.

Extra-Waypoint-Datasets: The Extra-Waypoint-Datasets defines how many consecutive vertices are converted into waypoints depending on the length of the path when using A* to translate waypoints into steps.

Regarding the MA:

MA Population Size $N_{MA\ \text{Individiuals}}$: Defines the number of individuals in the MA population.

MA Crossover-probability *MACXPB* : Defines the probability of the MA crossover operator.

MA Mutation-probability *MAMUTBP*: Defines the individual mutation probability of the MA mutation.

Regarding the SA and the MA

Maximal Number of Generations: Defines the number of generations before the algorithm terminates.

Number of Representatives N_R: Defines the number of representatives the SA selects. The MA individuals refer to these representatives.

Section B Parameter values

Table 14:Parameter values

SA Inputparameter	Value 1	Value 2	Value 3	Value 4	Value 5
SA-Population-Size	12	20	28	40	
Maximal-Number-of-Starting-Chromosones	1	2	3	5	
SA-Crossover-probability	0	0,25	0,5	0,75	1
SA-Mutation-probability Shift in Neighborhood	0	0,25	0,5	0,75	1
SA-Mutation-Probability-Gene-Deletion	0	0,25	0,5	0,75	1
SA-Mutation-Probability-Insert-Random-Waypoint	0	0,25	0,5	0,75	1
A-star-Variant-Switch	True	False			
Weight-Collisioncount	0,0001	0,25	0,5	0,75	0,9999
Collision-Swapping-Penalty-Value	0	0,1	0,25	0,5	
Crossover-Variant-Switch	True	False			
Extra-Waypoint-Datasets	Set 1	Set 2	Set 3	Set 4	
Crossover-deletion-value	0	0,25	0,5	0,75	1
Mutation-deletion-value	0	0,25	0,5	0,75	1
MA Inputparameter:					
MA-Population-Size	28	48	100	148	
MA-Mutation-Probability	0	0,25	0,5	0,75	1
MA-Crossoverprobability	0	0,25	0,5	0,75	1
Regarding MA and SA					
Maximal-Number-of-Generations	100				
Number-of-Representatives	5	10	20		

Table 15: Extra waypoints dataset values

	n	k	n	k	n	k	n	k	n	k	n	k	n	k	n	k	n	k	n	k
Set 1	0-10	0	10-20	1	20-30	2	30-40	3	40-50	4	50-60	5	60-70	6	70-80	7	80-90	8	90+	9		
Set 2	0-5	0	5-10	1	10-15	2	15-20	3	20-25	4	25-30	5	20-35	6	35-40	7	40-45	8	45-50	9	50+	10-15
Set 3	0-20	0	20-40	1-2	40-60	2-3	60-80	3-4	80+	4												
Set 4	0-5	0	5-10	1	10-20	1-2	20-30	1-3	30-40	2-4	40+	3-5										

The options for the Extra Waypoint Datasets are presented in table 15. The number of waypoints, which are being inserted, is always depending on the path length between two waypoints and the chosen Extra waypoint Dataset. Set 4 for instance creates 0 waypoints if path consists of 5 vertexes or less, 1 waypoint if path has 5 to 10 vertexes or less, 1 to 2 waypoints for 10 to 20 vertexes, 1 to 3 waypoints for 20 to 30 vertexes, 2-4 for 30 to 40 vertexes and 3 to 5 if the path length exceeds 40 vertices. Set 2 creates the most waypoints, Set 3 the least and Set 1 and 4 are in between.

Section C Win Lose Tie Table

Table 16: Win-Lose-Tie-Table of the comparison of the SO SACCGA and the MO SACCGA

Win Loose Tie Table									
	GD	GD	GD	IGD	IGD	IGD	Weighted Sum		
Maptype + Scene		SO SACCGA	MO SACCGA		SO SACCGA	MO SACCGA		SO SACCGA	MO SACCGA
	Tie	Win	Win	Tie	Win	Win	Tie	Win	Win
Empty 8 S1	54,84%	0,00%	45,16%	54,84%	0,00%	45,16%	54,84%	0,00%	45,16%
Empty 8 S2	51,61%	0,00%	48,39%	51,61%	0,00%	48,39%	90,32%	0,00%	9,68%
Empty 8 S3	22,58%	0,00%	77,42%	22,58%	0,00%	77,42%	22,58%	0,00%	77,42%
Empty 8 all three scenes	43,01%	0,00%	56,99%	43,01%	0,00%	56,99%	55,91%	0,00%	44,09%
Empty 16 S1	18,37%	0,00%	81,63%	18,37%	0,00%	81,63%	18,37%	0,00%	81,63%
Empty 16 S2	30,61%	0,00%	69,39%	30,61%	0,00%	69,39%	30,61%	0,00%	69,39%
Empty 16 S3	20,41%	0,00%	79,59%	20,41%	0,00%	79,59%	20,41%	0,00%	79,59%
Empty 16 all three scenes	23,13%	0,00%	76,87%	23,13%	0,00%	76,87%	23,13%	0,00%	76,87%
Empty 32 S1	43,75%	0,00%	56,25%	43,75%	0,00%	56,25%	43,75%	0,00%	56,25%
Empty 32 S2	17,19%	0,00%	82,81%	17,19%	0,00%	82,81%	17,19%	0,00%	82,81%
Empty 32 S3	34,38%	0,00%	65,63%	34,38%	0,00%	65,63%	34,38%	0,00%	65,63%
Empty 32 all three scenes	31,77%	0,00%	68,23%	31,77%	0,00%	68,23%	31,77%	0,00%	68,23%
Maze 32 2 S1	41,03%	15,38%	43,59%	35,90%	7,69%	56,41%	25,64%	43,59%	30,77%
Maze 32 2 S2	28,21%	0,00%	71,79%	28,21%	2,56%	69,23%	25,64%	46,15%	28,21%
Maze 32 2 S3	20,51%	0,00%	79,49%	17,95%	0,00%	82,05%	35,90%	2,56%	61,54%
Maze 32 2 all three scenes	29,91%	5,13%	64,96%	27,35%	3,42%	69,23%	29,06%	30,77%	40,17%
Maze 32 4 S1	15,38%	0,00%	84,62%	0,00%	0,00%	100,00%	0,00%	0,00%	100,00%
Maze 32 4 S2	5,13%	0,00%	94,87%	5,13%	0,00%	94,87%	28,21%	0,00%	71,79%
Maze 32 4 S3	25,71%	8,57%	65,71%	25,64%	5,13%	69,23%	35,90%	5,13%	58,97%
Maze 32 4 all three scenes	15,04%	2,65%	82,30%	10,26%	1,71%	88,03%	21,37%	1,71%	76,92%
Random 32 10 S1	42,86%	0,00%	57,14%	42,86%	0,00%	57,14%	42,86%	0,00%	57,14%
Random 32 10 S2	46,94%	0,00%	53,06%	46,94%	0,00%	53,06%	46,94%	0,00%	53,06%
Random 32 10 S3	57,14%	0,00%	42,86%	57,14%	0,00%	42,86%	57,14%	0,00%	42,86%
Random 32 10 all three scenes	48,98%	0,00%	51,02%	48,98%	0,00%	51,02%	48,98%	0,00%	51,02%
Ramdom 32 20 S1	38,78%	0,00%	61,22%	38,78%	0,00%	61,22%	38,78%	0,00%	61,22%
Ramdom 32 20 S2	18,37%	0,00%	81,63%	18,37%	0,00%	81,63%	20,41%	0,00%	79,59%
Ramdom 32 20 S3	8,16%	0,00%	91,84%	8,16%	0,00%	91,84%	8,16%	0,00%	91,84%
Ramdom 32 20 all three scenes	21,77%	0,00%	78,23%	21,77%	0,00%	78,23%	22,45%	0,00%	77,55%
Ramdom 64 10 S1	30,51%	0,00%	69,49%	30,51%	0,00%	69,49%	30,51%	0,00%	69,49%
Ramdom 64 10 S2	35,59%	0,00%	64,41%	35,59%	0,00%	64,41%	35,59%	0,00%	64,41%
Ramdom 64 10 S3	27,12%	0,00%	72,88%	27,12%	0,00%	72,88%	27,12%	0,00%	72,88%
Ramdom 64 10 all three scenes	31,07%	0,00%	68,93%	31,07%	0,00%	68,93%	31,07%	0,00%	68,93%
Random 64 20 S1	30,51%	0,00%	69,49%	30,51%	0,00%	69,49%	30,51%	0,00%	69,49%
Random 64 20 S2	33,90%	0,00%	66,10%	33,90%	0,00%	66,10%	33,90%	0,00%	66,10%
Random 64 20 S3	50,85%	0,00%	49,15%	50,85%	0,00%	49,15%	50,85%	0,00%	49,15%
Random 64 20 all three scenes	38,42%	0,00%	61,58%	38,42%	0,00%	61,58%	38,42%	0,00%	61,58%
Room 32 4 S1	17,95%	0,00%	82,05%	17,95%	0,00%	82,05%	17,95%	0,00%	82,05%
Room 32 4 S2	25,64%	0,00%	74,36%	25,64%	0,00%	74,36%	51,28%	2,56%	46,15%
Room 32 4 S3	30,77%	0,00%	69,23%	30,77%	0,00%	69,23%	30,77%	0,00%	69,23%
Room 32 4 all three scenes	24,79%	0,00%	75,21%	24,79%	0,00%	75,21%	33,33%	0,85%	65,81%
Room 64 8 S1	8,16%	0,00%	91,84%	8,16%	0,00%	91,84%	34,69%	10,20%	55,10%
Room 64 8 S2	8,16%	0,00%	91,84%	8,16%	0,00%	91,84%	8,16%	0,00%	91,84%
Room 64 8 S3	42,86%	0,00%	57,14%	42,86%	0,00%	57,14%	44,90%	0,00%	55,10%
Room 64 8 all three scenes	19,73%	0,00%	80,27%	19,73%	0,00%	80,27%	29,25%	3,40%	67,35%
Room 64 16 S1	42,86%	0,00%	57,14%	42,86%	0,00%	57,14%	44,90%	0,00%	55,10%
Room 64 16 S2	32,65%	0,00%	67,35%	32,65%	0,00%	67,35%	32,65%	0,00%	67,35%
Room 64 16 S3	20,41%	0,00%	79,59%	20,41%	0,00%	79,59%	20,41%	0,00%	79,59%
Room 64 16 all three scenes	31,97%	0,00%	68,03%	31,97%	0,00%	68,03%	32,65%	0,00%	67,35%

References

[1] F. Ahmed and K. Deb, *Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms*, Soft Computing 17 (2013), pp. 1283–1299.

[2] M. Barer, Sharon, Guni, Stern, Roni, and A. Felner, *Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem*, in: *Proceedings of the Seventh International Symposium on Combinatorial Search (SoCS-2014), 15 - 17 August 2014, Prague, Czech Republic*, S. Edelkamp and R. Barták, eds. AAAI Press, Palo Alto, Calif., 2014, pp. 19–27.

[3] A. Bucci and J.B. Pollack, *On identifying global optima in cooperative coevolution*, in: *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, U.-M. O'Reilly and H.-G. Beyer, eds., the 2005 conference, Washington DC, USA, Jun. 25-29, 2005. ACM Press, New York, New York, USA, 2005, p. 539.

[4] Z. Cai and Z. Peng, *Cooperative Coevolutionary Adaptive Genetic Algorithm in Path Planning of Cooperative Multi-Mobile Robot Systems*, Journal of Intelligent and Robotic Systems 33 (2002), pp. 61–71.

[5] J. Chen, D. Yang, N. Matsumoto, and Y. Yamane, *Multi-robot Path Planning Based on Cooperative Co-evolution and Adaptive CGA*, in: *2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IEEE Computer Society, ed., 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Hong Kong, Dec. 18-22, 2006. IEEE, 2006, pp. 547–550.

[6] L. Cohen, *Efficient Bounded-Suboptimal Multi-Agent Path Finding and Motion Planning via Improvements to Focal Search*, Dissertation, 2020.

[7] Y. Dai, Y. Kim, S. Wee, D. Lee, and S. Lee, *A Switching Formation Strategy for Obstacle Avoidance of a Multi-Robot System Based on Robot Priority Model*, ISA transactions 56 (2015), pp. 123–134.

[8] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, *A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II*, in: *Parallel Problem Solving from Nature PPSN VI*, G. Goos, J. Hartmanis, J. van Leeuwen, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, eds., Lecture Notes in Computer Science Vol. 1917. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 849–858.

[9] K. Dresner and P. Stone, *A Multiagent Approach to Autonomous Intersection Management*, jair 31 (2008), pp. 591–656.

[10] A. Felner, R. Stern, S.E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N.R. Sturtevant, G. Wagner, and P. Surynek, *Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges*, in: *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS 2017)*: 16-17 June 2017, Pittsburgh, Pennsylvania, USA, A. Fukunaga and A. Kishimoto, eds. AAAI Press, Palo Alto, California, 2017.

[11] F.-A. Fortin, F.-M. de Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, *Evolutionary Tools: deap.tools.cxUniform(ind1, ind2, indpb)*. 2009. Available at https://deap.readthedocs.io/en/master/api/tools.html#deap.tools.cxUniform. Accessed December 4, 2020.

[12] F.-A. Fortin, F.-M. de Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, *Evolutionary Tools: deap.tools.mutUniformInt(individual, low, up, indpb)*. 2009. Available at https://deap.readthedocs.io/en/master/api/tools.html#deap.tools.mutUniformInt. Accessed December 4, 2020.

[13] F.-A. Fortin, F.-M. de Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, *Evolutionary Tools: deap.tools.selNSGA2(individuals, k, nd='standard')*. 2009. Available

at https://deap.readthedocs.io/en/master/api/tools.html#deap.tools.selNSGA2. Accessed December 2, 2020.

[14] F.-A. Fortin, F.-M. de Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, *Evolutionary Tools: deap.tools.selTournamentDCD(individuals, k)*. 2009. Available at https://deap.readthedocs.io/en/master/api/tools.html#deap.tools.selTournamentDCD. Accessed December 2, 2020.

[15] F.-A. Fortin, F.-M. de Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, *DEAP: Evolutionary Algorithms Made Easy*, Journal of Machine Learning Research (2012), pp. 2171–2175.

[16] D.K. Grady, K.E. Bekris, and L.E. Kavraki, *Asynchronous Distributed Motion Planning with Safety Guarantees under Second-Order Dynamics*, in: *Algorithmic Foundations of Robotics IX*, B. Siciliano, O. Khatib, F. Groen, D. Hsu, V. Isler, J.-C. Latombe, and M.C. Lin, eds., Springer Tracts in Advanced Robotics Vol. 68. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 53–70.

[17] H. Ishibuchi, R. Imada, N. Masuyama, and Y. Nojima, *Comparison of Hypervolume, IGD and IGD+ from the Viewpoint of Optimal Distributions of Solutions*, in: *Evolutionary Multi-Criterion Optimization*, K. Deb, E. Goodman, C.A. Coello Coello, K. Klamroth, K. Miettinen, S. Mostaghim, and P. Reed, eds., Lecture Notes in Computer Science Vol. 11411. Springer International Publishing, Cham, 2019, pp. 332–345.

[18] R. Kala, *Multi-robot path planning using co-evolutionary genetic programming*, Expert Systems with Applications 39 (2012), pp. 3817–3831.

[19] S. Koenig, *Learn all about Multi-Agent Path Finding (MAPF): Benchmarks*. Available at http://mapf.info/index.php/Main/Benchmarks. Accessed November 21, 2020.

[20] M.H. Korayem, A.K. Hoshiar, and M. Nazarahari, *A hybrid co-evolutionary genetic algorithm for multiple nanoparticle assembly task path planning*, Int J Adv Manuf Technol 87 (2016), pp. 3527–3543.

[21] J.-H. Liang and C.-H. Lee, *Efficient collision-free path-planning of multiple mobile robots system using efficient artificial bee colony algorithm*, Advances in Engineering Software 79 (2015), pp. 47–56.

[22] H. Ma, W. Hönig, T.K.S. Kumar, N. Ayanian, and S. Koenig (eds.), *Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery*, 2019.

[23] S. Mai and S. Mostaghim, *Modeling Pathfinding for Swarm Robotics*, in: *Swarm Intelligence*, M. Dorigo, T. Stützle, M.J. Blesa, C. Blum, H. Hamann, M.K. Heinrich, and V. Strobel, eds., Lecture Notes in Computer Science Vol. 12421. Springer International Publishing, Cham, 2020, pp. 190–202.

[24] M. Muthiah and A. Saad, *Multi Robot Path Planning and Path Coordination Using Genetic Algorithms*, in: *Proceedings of the SouthEast Conference*. ACM, New York, NY, USA, 2017, pp. 112–119.

[25] G. Nagib and W. Gharieb, *Path planning for a mobile robot using genetic algorithms*, in: *International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04*, International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04, Cairo, Egypt, 5-7 Sept. 2004. IEEE, 5-7 Sept. 2004, pp. 185–189.

[26] S. Narmadha, V. Selladurai, and G. Sathish, *Multi Product Inventory Optimization using Uniform Crossover Genetic Algorithm*, International Journal of Computer Science and Information Security abs/1002.2195 (2010), pp. 170–179.

[27] G.M.B. Oliveira, R.G.O. Silva, G.B.S. Ferreira, M.S. Couceiro, L.R. do Amaral, P.A. Vargas, and L.G.A. Martins, *A Cellular Automata-Based Path-Planning for a Cooperative and Decentralized Team of Robots*, in: *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, Jun. 10-13, 2019. IEEE, 10.06.2019 - 13.06.2019, pp. 739–746.

[28] Z. Pan, Di Wang, H. Deng, and K. Li, *A Virtual Spring Method for the Multi-robot Path Planning and Formation Control*, Int. J. Control Autom. Syst. 17 (2019), pp. 1272–1282.

[29] L. Panait, R.P. Wiegand, and S. Luke, *A Sensitivity Analysis of a Cooperative Coevolutionary Algorithm Biased for Optimization*, in: *Genetic and Evolutionary Computation – GECCO 2004*, T. Kanade, J. Kittler, J.M. Kleinberg, F. Mattern, J.C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M.Y. Vardi, G. Weikum, and K. Deb, eds., Lecture Notes in Computer Science Vol. 3102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 573–584.

[30] O. Popović, *Merge Sort in Python*. Available at https://stackabuse.com/merge-sort-in-python/. Accessed December 2, 2020.

[31] M.A. Potter and K.A. Jong, *A cooperative coevolutionary approach to function optimization*, in: *Parallel Problem Solving from Nature — PPSN III*, G. Goos, J. Hartmanis, J. Leeuwen, Y. Davidor, H.-P. Schwefel, and R. Männer, eds., Lecture Notes in Computer Science Vol. 866. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994, pp. 249–257.

[32] H. Qu, K. Xing, and T. Alexander, *An improved genetic algorithm with co-evolutionary strategy for global path planning of multiple mobile robots*, Neurocomputing 120 (2013), pp. 509–517.

[33] R. Sarkar, D. Barman, and N. Chowdhury, *A Cooperative Co-evolutionary Genetic Algorithm for Multi-Robot Path Planning Having Multiple Targets*, in: *Computational Intelligence in Pattern Recognition*, A.K. Das, J. Nayak, B. Naik, S.K. Pati, and D. Pelusi, eds., Advances in Intelligent Systems and Computing Vol. 999. Springer Singapore, Singapore, 2020, pp. 727–740.

[34] D. Silver, *Cooperative Pathfinding*, in: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'05. AAAI Press, 2005, pp. 117–122.

[35] N.K. Singh, *Binary Heap*. 2020. Available at https://www.geeksforgeeks.org/binary-heap/. Accessed December 12, 2020.

[36] N. Soni and T. Kumar, *Study of Various Mutation Operators in Genetic Algorithms*, International Journal of Computer Science and Information Technologies (2014), pp. 4519–4521.

[37] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.K.S. Kumar, E. Boyarski, and R. Bartak, *MAPF Benchmark Sets*. 2019. Available at https://movingai.com/benchmarks/mapf/index.html. Accessed December 4, 2020.

[38] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.K.S. Kumar, E. Boyarski, and R. Bartak, *Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks*, in: *Proceedings of the Twelfth International Symposium on Combinatorial Search*: SOCS 2019, P. Surynek and W. Yeoh, eds. AAAI Press 2019, Napa, California, 2019, 151--158.

[39] P. Surynek, *Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving*, in: *PRICAI 2012: Trends in Artificial Intelligence*, D. Hutchison, T. Kanade, J. Kittler, J.M. Kleinberg, F. Mattern, J.C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M.Y. Vardi, G. Weikum, P. Anthony, M. Ishizuka, and D. Lukose, eds., Lecture Notes in Computer Science Vol. 7458. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 564–576.

[40] P. Surynek, A. Felner, R. Stern, and E. Boyarski, *An Empirical Comparison of the Hardness of Multi-Agent Path Finding Under the Makespan and the Sum of Costs Objectives*, in: *Proceedings of the Ninth International Symposium on Combinatorial Search (SoCS 2016)*: 6-8 July 2016, Tarrytown, New York, USA, AAAI, ed. AAAI Press, Palo Alto, California, 2016, pp. 145–146.

[41] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, and P. van Mulbregt, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature methods 17 (2020), pp. 261–272.

[42] J. Weise, S. Mai, H. Zille, and S. Mostaghim, *On the Scalable Multi-Objective Multi-Agent Pathfinding Problem*, in: *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE Congress on Evolutionary Computation, ed., 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, United Kingdom, 2020. IEEE, 2020, pp. 1–8.

[43] S. Wessing, *Hypervolume: Python*. 2010. Available at https://ls11-www.cs.tu-dortmund.de/rudolph/hypervolume/start. Accessed December 6, 2020.

[44] R.P. Wiegand, *An Analysis of Cooperative Coevolutionary Algorithms*, PhD thesis, 2003.

[45] R.P. Wiegand and M.A. Potter, *Robustness in cooperative coevolution*, in: *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, M. Cattolico, ed., the 8th annual conference, Seattle, Washington, USA, Jul. 08-12, 2006. ACM Press, New York, New York, USA, 2006, p. 369.

[46] J. Yu and S.M. LaValle, *Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs*, in: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence and the Twenty-Fifth Innovative Applications of Artificial Intelligence Conference*: 14 - 18 July 2013, Bellevue, Washington, USA ; [the Fourth Symposium on Educational Advances in Artificial Intelligence. AAAI Press, Palo Alto, Calif., 2013, pp. 1443–1449.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Kevin Kellermann Magdeburg, December 17, 2020