Masterarbeit
Intelligente Systeme
Technische Fakultät

Universität Bielefeld
AG Kognitronik & Sensorik
Prof. Dr.-Ing. Ulrich Rückert

Solving the Area Coverage Problem for Heterogeneous Robot Fleets using Decentralized Optimization

Georg Alberding

Betreuer: M.Sc. Thomas Schöpping
Prof. Dr.-Ing. Ulrich Rückert

Bielefeld, April 2021

M 58

Abstract

Die Planung von Abdeckungspfaden wird verwendet, um einen oder mehrere Roboter durch ein Zielgebiet zu navigieren mit der Aufgabe, jeden Teil des Gebietes abzufahren. Das Generieren solcher Abdeckungspfade ist ein NP-hartes Problem und erfordert ein heuristisches Optimierungsverfahren. Daher wird ein Genetischer Algorithmus entworfen, um solche Pfade zu generieren. Der Genetische Algorithmus wird mit verschiedenen Parametereinstellungen sowie unterschiedlichen Umgebungsbedingungen getestet. Die Ergebnisse zeigen, dass nahezu optimale Pfade generiert werden, die eine hohe Gesamtabdeckung der Region erreichen und gleichzeitig sich überkreuzende Pfadsegmente reduziert. Ein Nachteil ist, dass die Pfadgenerierung bis zu zehn Minuten dauern kann. Daher werden Ideen vorgestellt, die den gesamten Optimierungsprozess beschleunigen können.

Coverage path planning is utilized to navigate one or several robots through a target region with the objective of visiting each part of the area. Planning such coverage paths is a NP-Hard problem and requires a heuristic optimization procedure. Therefore a custom Genetic Algorithm is designed in order to generated a coverage path for a single robot use-case. The Genetic Algorithm is tested with several different parameter settings as well as different environment conditions. Results show that near optimal paths are generated that reach high overall coverage and simultaneously reduce the amount of crossing path segments. One major drawback is that the path generation can last up to ten minutes, thus several ideas are presented that can accelerate the overall optimization process.

Contents

1 Introduction

Mobile robots allow to automate numerous tasks that are repetitive or even dangerous for humans. One of which is the coverage path planning[1] for either one or a team of robots. In order to coordinate multiple robots in an efficient manner an appropriate coordination strategy is required. This work aims to construct such a strategy for a single robot with possible extension applicable to a team of robots.

1.1 Motivation

Performing a coverage task with a mobile robot consists of several steps with the end result that a so called coverage path is generated. This coverage path is supposed to navigate a robot across a target region and needs to ensure that the robot visits all possible area inside that region. The action that is performed at each position in the target area is highly dependent on the task the robot should perform and thus directly impacts the generated coverage path. One example are vacuum cleaning robots[2] with the goal of traveling through a target region and cleaning all reachable area. Similar techniques can also be applied in the context of agriculture path planning[3] where it is beneficial to optimize the coverage path to reduce overall costs in terms of time spent, driving and fuel consumption. Those, the time and more generally the energy consumption are important aspects that need to be considered. Their importance further move into focus when only limited resources are available such as in seabed mapping[4] where the energy constraints are the limiting factor or in search or rescue missions[5] where the time factor could potentially decide between life or death.

One general adopted strategy in order to deal with limited resources is to increase the amounts of robots in order to perform the respective task in parallel[6]. Usually homogeneous robots are employed in a respective task i.g. robots with same physical properties such as dimensions in width and height, equipped sensors and sensor range or battery capacity. Utilizing robots that are heterogeneous i.g. have different physical properties could be even more beneficial. This work mainly concerns with

the application of coordination cleaning robots, therefore the afore mentioned concept could be applied as described next.

Consider a simple coverage task like swipe or vacuum a known area with a cleaning robot. Even such a simple task involves many challenges that have to be overcome. For example, it is important to ensure that the robot does not collide with any of the existing obstacles in the room. Furthermore, is it required that as much area as possible is cleaned with respect to the robots charge. That is, the robot has only limited capacity of charge that will be decrease over time. Therefore it is desired that the robot completes the cleaning task as fast as possible in order to reduce energy usage. Hence, the cleaning path generated by the robot should be as efficient as possible. Furthermore, exploiting different shapes of robots could increase the performance even more. For example, large factory buildings with mostly accessible floor would benefit from a robot that has a large cleaning area for most effective and fast cleaning. However, such a robot would be unsuited for typical households because of spaces e.g. underneath furniture that are unreachable. Hence, a smaller robot is more suited for such a use-case. An ideal scenario for combining these two robot types are office buildings. Such buildings are often composed of small office rooms with abundant furniture connected by larger corridors. As mentioned earlier both robots could find applicable spaces suited to their specific design without hindering the other one. Coordinating such robots requires an advanced coordination strategy that additionally takes the different properties of the robots into account.

This work aims to lay the foundation of such a strategy by creating a coverage path for a single robot scenario with the help of a custom Genetic Algorithm optimization procedure and furthermore explore the capabilities of possible extension to multi robot coordination.

1.2 The Area Coverage Problem

In general trying to cover a target region is referred to as the *Area Coverage Problem* (ACP). Numerous works concern with solving problems that are related to the ACP[7, 8, 6] usually by applying the so called Coverage Path Planning (CPP). The CPP is used as a guideline which states the general objectives that should be accomplished when generating a coverage path.

The following shows the major goals of the CPP[1]:

1. Move through all points in the target area

2. Fill region minimally overlapping paths
3. Continuous and sequential operation without any repetition of paths
4. Obstacle avoidance
5. Use of simple motion trajectories (e.g. straight lines)
6. Computed path should be “optimal” under given conditions

Furthermore, the CPP in general is composed of two steps[6]. First, so-called Viewpoints (VPs) are generated. Those are simply two dimensional positions in the target area that needs to be visited by the robot. In the second step those VPs are arrange in an optimal visiting order to optimize the overall coverage generated by the resulting path.

The overall path generation procedure is NP-Hard as further explained in Section 2.1.4. Therefore some kind of optimization procedure needs to be utilized. Literature suggests several approaches as further described in the next chapter however, most of those approaches restrict the overall path generation in terms of a visiting neighborhood or require a prior segmentation of the map in order to assign a region to a robot. That is, without getting into too much detail, visiting neighborhood simply refers to the movement directions a robot can take on a given map representation. Usually those directions are restricted to four or eight neighbors meaning that the robot can turn at a VP in 90 or 40 degree angles. This work creates a custom Genetic Algorithm that does not restrict the robot in its rotation movements and furthermore takes the complete map into account when generating the coverage path. A later extension shows that during a possible negotiation task, the coverage path can be adapted without prior segmenting the given map into regions for each robot.

1.3 Outline

This work is organized in the following chapters. First Chapter 2 presents examples for multi robot CPP. Furthermore, Genetic Algorithms are introduced as well as the most common operators and how they work. Chapter 3 presents the custom GA approach together with some possible extensions as well as an experiment for testing the capabilities of adapting to changing environment conditions. Chapter 4 presents the overall parameter setting for the GA as well as the test results that are generated by a grid search. The results are discussed in Chapter 5, followed by a short conclusion in Chapter 6.

2 Related Work

The first part of this chapter presents approaches that are designed to generate coverage paths on a known map and can be extended to a multi-robot scenario. Consecutively the general principle and mechanisms of Genetic Algorithms (GA) are described.

2.1 Coverage Path Planning

A navigation task usually consists of three tightly coupled components[9]: localization, map representation and path planning. In a real world application localization is essential because it assigns a robot to a position and pose inside a given map. Getting the positioning right is not a trivial task because one needs to utilize sensors and compensate for their error and noise during measurements[10]. The second component in a navigation task is the map representation. Usually the map is represented as a discrete grid where each cell represents one part of the target area[11] but complete continuous representations based on polygons are also frequently used[12].

The path planning approach can be classified as model or non-model based. Non-model based approaches have no prior information about the environment and perform simultaneously mapping and path planning[9]. Model based approaches usually have complete information about all environment features like obstacles or walls. A non model based approach needs to continuously construct its environment and therefore plan the next move after each iteration. Such a process is called online. In contrast the model based approaches have all needed information to calculate a complete path. Thus the path planning is performed before the operation, which is referred to as offline[6].

In the following different coverage path planning procedures are presented.

2.1.1 Movement Pattern

One possible approach in coordinating robots is pattern based. For example, [13] utilizes several coverage pattern to steer different robots on a map where each robot follows a certain movement pattern for a fixed amount of steps. The map or environment the robot is supposed to cover is composed of disk shaped cells. The arrangement of those disks is illustrated in Figure 2.1. A disk is covered when the robot passes through the center. Furthermore, disks are arranged in an overlapping manner such that there is no redundant or free area between. The robot is able to visit all eight adjacent disks when following the disk-prioritization pattern. Disks representing obstacles are marked in gray and block the robots movement direction.

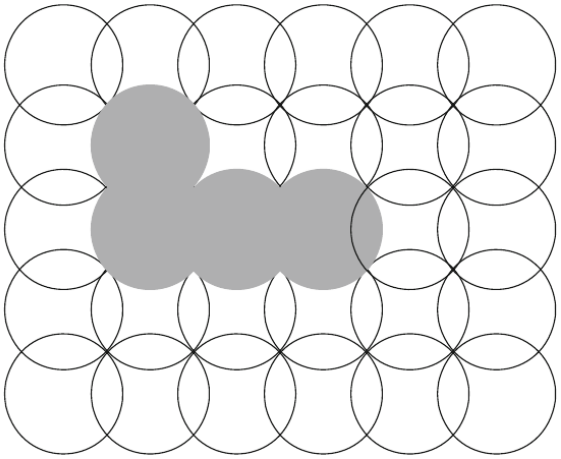


Figure 2.1: Environment represented by disks, each disk corresponds to robot sensor range. Obstacles are marked with gray.

Examples for those patterns are presented in Figure 2.2 and are structured as follows: The cell in the center marked with PX is the current robot location where X corresponds to the current pattern number. Movement direction is determined by the priority value of the neighbored cells. The cell with highest priority is selected where lower values indicate higher priority. When the highest priority cell is occupied or blocked the movement direction is determined by the second highest priority and so forth. Figure 2.3b illustrates the movement dependent on a given pattern. For example, the circle represents the first robot that is encoded in the genome representation 2.3a with the name “Robot#1”. Furthermore, the encoding contains the specific robot id (*1*) as well as the movement pattern (*P3*) and amount (*17*) of steps state how many steps P3 shall be applied. Overall the given example allows two movement pattern for

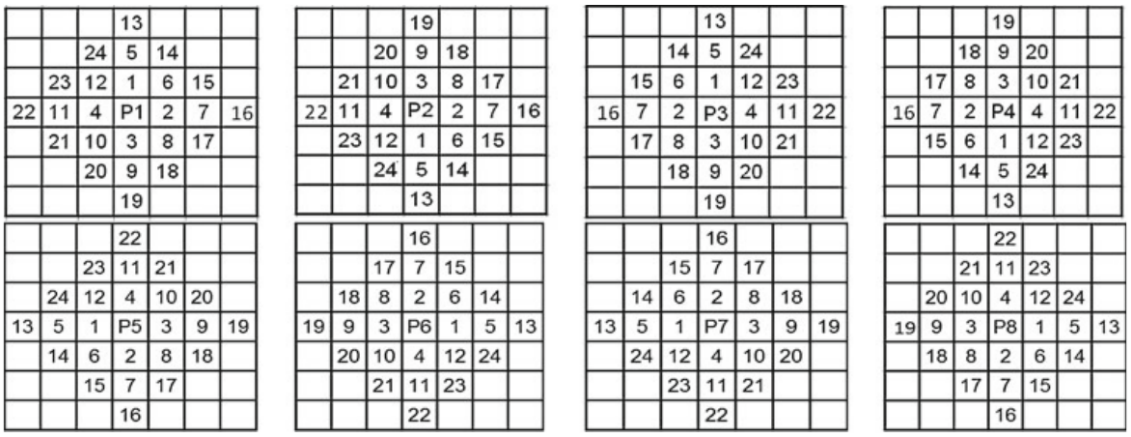


Figure 2.2: Examples of neighbor-prioritization patterns.

each robot, consequently the next two fields also describe movement pattern (*P2*) and corresponding steps (*19*).

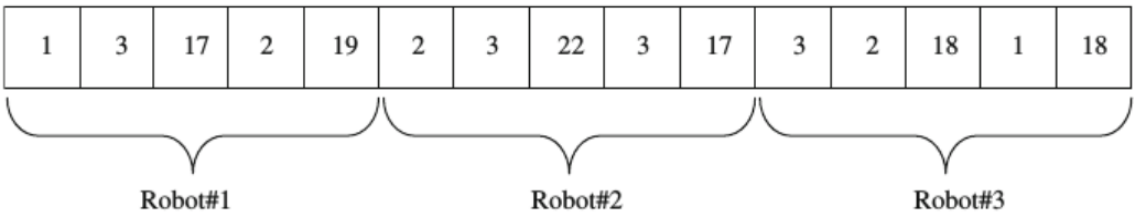
Following the path of "Robot#1" results in an impasse, on the map marked with a circle labeled 2. The first one is the start and the second one the end position that will result in repeated coverage. Impasses are resolved with the movement patterns by exploring neighbors that are not directly adjacent to the current position. In order to continue with the coverage task a path from the current to the next free disk is planned that navigates over already covered area.

The goal of the genetic algorithm is to arrange the movement pattern and the amount of steps such that minimal repeated coverage occurs and the overall paths are as short as possible but simultaneously cover all disks.

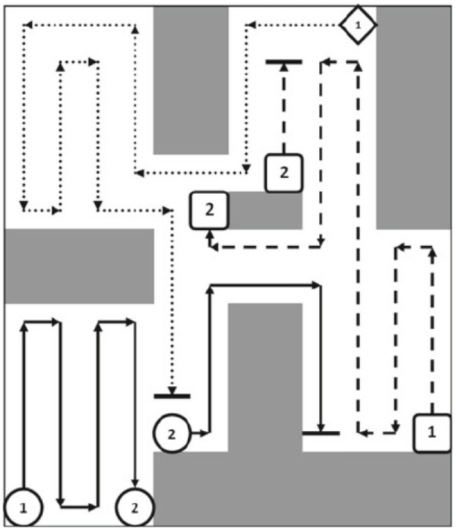
2.1.2 Cell Decompositon

Utilizing coverage patterns is a well established approach. In fact one of the most popular approaches the cell decomposition utilizes coverage pattern, too. The cell decomposition procedure divides desired area into cells and applies a coverage pattern to it. Furthermore, the coverage pattern is fairly simple and consists of basic back and forth motion. Additionally, the simplistic path that is generated is convenient to determine if the target area is optimally covered.

Two variations of the cell decomposition are illustrated in Figure 2.4. Trapezoidal decomposition Figure 2.4a aims to generate convex cells whereas the boustrophedon Figure 2.4b relaxes this assumption because covering non-convex cells works with



(a) Representation contains three types of information: Robot ID (first position), pattern ID and associated steps following the given pattern.



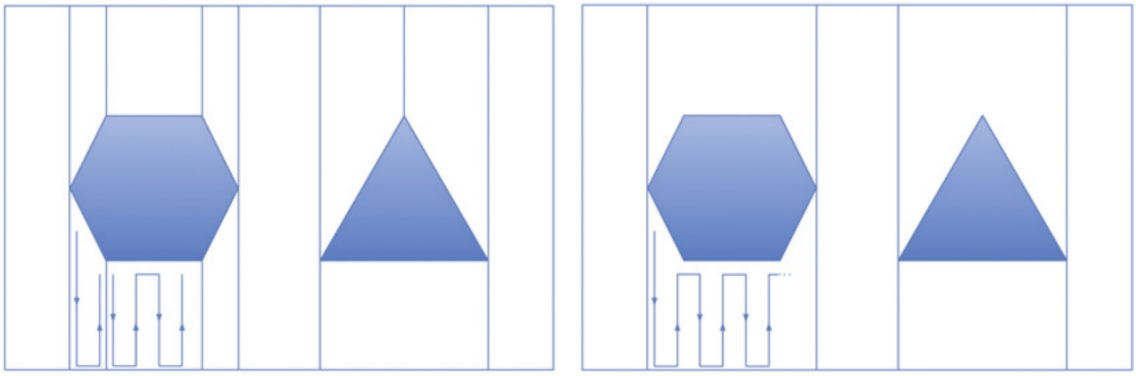
(b) Each robot is represented by a differently shaped node (circle, square, dimond) and line type. Nodes labeled with 1 are starting positions. Nodes greater 1 are start and end positions associated with repeated coverage.

Figure 2.3: Genome represenation (a) that generate the path (b) for three different robots.

simple motion. Therefore cells are directly created around obstacles hence reducing the amount of generated cells and therefore the overall length of the path[14].

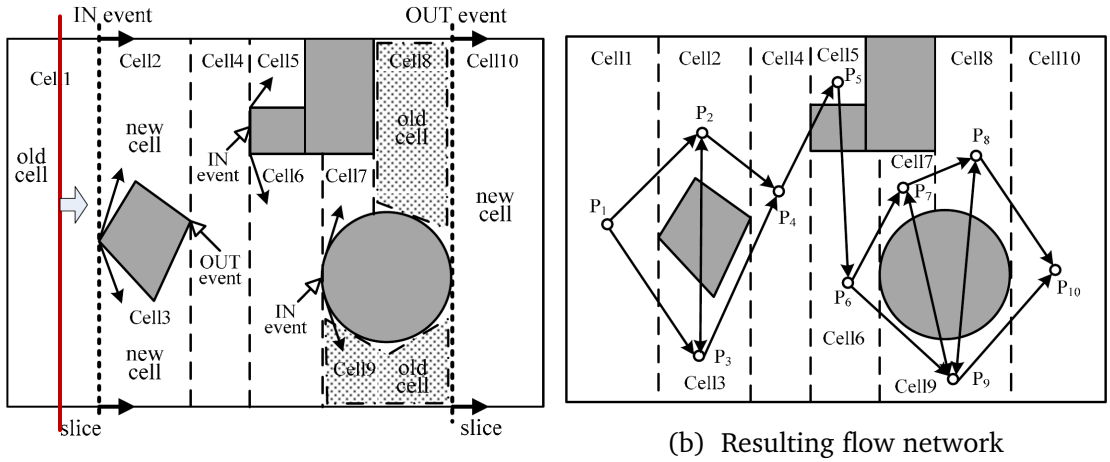
Similarly Janchiv, Dugarjav, Kim, et al. demonstrate that boustrophedon decomposition can be used to construct a flow network that is designed to generate a coverage path for one or more robots. The previously mentioned cell generation is illustrated more detailed in Figure 2.5a.

Origin for the procedure is a known map with obstacles in it. Obstacles are present in different shapes and locations. Generating cells with the decomposition works by scanning the map on the vertical axis, marked in red as “slice” in horizontal direction. As soon as an obstacle is detected a so called IN event closes the current cell e.g. Cell1



(a) Trapezoidal decomposition (b) Boustrophedon decomposition

Figure 2.4: Different versions of cell decompositon[14].



(a) Cell decomposition (b) Resulting flow network

Figure 2.5

and create two new ones above and below the obstacle e.g. Cell2 and Cell3. If the end of it is reached an OUT event merges both cells into one e.g. Cell4. Furthermore, the cells are represented as nodes of the flow network and can be connected according to the spacial arrangement on the horizontal axis. The flow network in Figure 2.5b illustrates all possible movements inside the network.

The goal of the search algorithm is to determin the minimum cost path from a given start node to a final node where each is visited exactly once. Therefore all possible spanning paths are identified and where the path with the minimum costs is selected. In order to calculate the costs for each cell, the movement templates in Figure 2.6 are utilized to connect to the next following cell. The chosen template determines the time

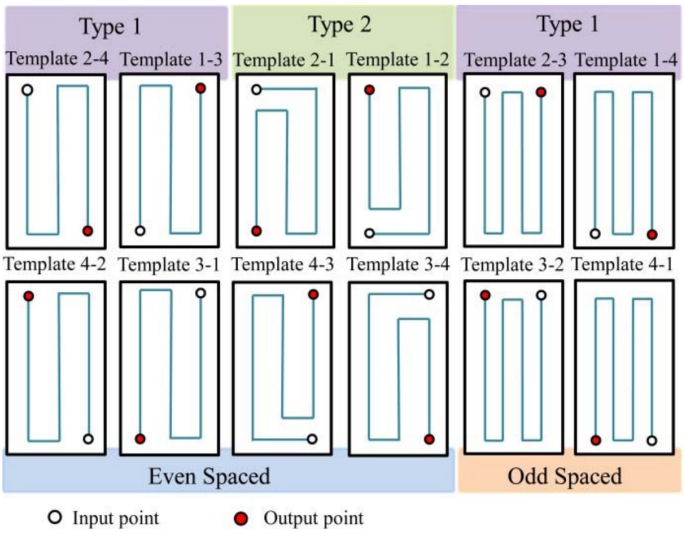
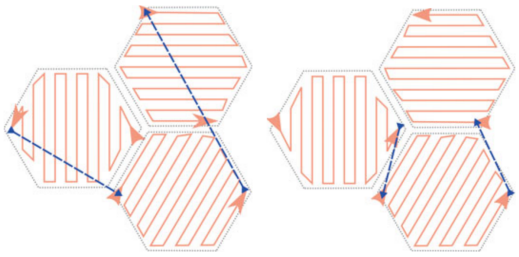


Figure 2.6: Movement templates designed to match the input point with the output point of the previous cell. Positions of the points are numbered clockwise starting at bottom left

needed to cover a cell and represent the cost of the edge from one node to another in the flow network[15].

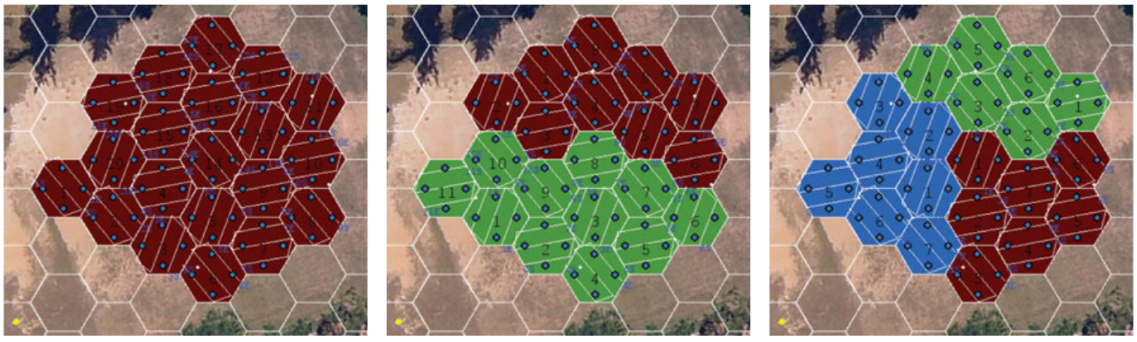
As mentioned before there are numerous other approaches utilizing cell decomposition where many properties of the approach vary dependent on the current task. For example the area is decomposed into hexagonal instead of rectangular cells[16]. Hence a coverage path is generated by applying some coverage pattern inside the cells. Furthermore those cells have IN and OUT positions that mark the beginning, connection to another cell and the end. The benefit of using hexagonal cells is the property that the distance from the center to each of the neighbored cells is the same. Therefore hexagonal cells can be rotated while simultaneously preserving the neighborhood. Rotations are used to minimize the travel path between cells as illustrated in 2.7. Figure 2.7a shows the cells after the coverage pattern is applied and cells are in a non optimal rotation state. After optimizing orientation of the cells the minimal coverage path is generated as illustrated in Figure 2.7b.

Additionally this approach is applicable to multiple robots. Each robot is assigned one ore more neighbored cells and consecutive the cells are rotated to minimize the overall coverage path. Figure 2.8 illustrates the multi-robot application.



(a) (b)

Figure 2.7: Hexagonal cells with lawnmower[16] pattern. (a) initial coverage path (b) rotated path with now optimized transitions between the cells



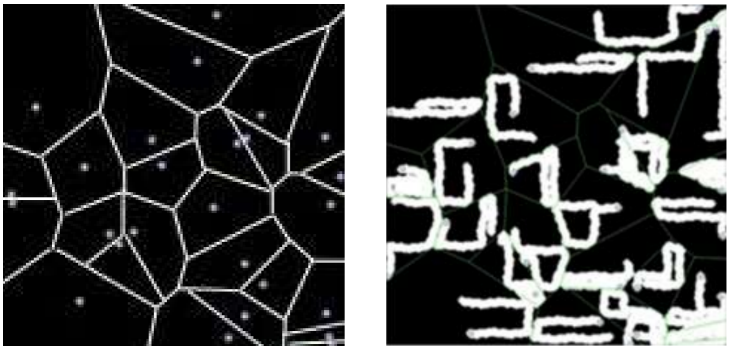
(a) (b) (c)

Figure 2.8: Example of unmanned aerial vehicles (AUVs) covering decomposed region with hexagonal cells. (a) one AUV (b) two AUVs (c) three AUVs

2.1.3 Region Clustering

Despite utilizing coverage patterns another option is to apply an optimization procedure that generates a path directly for a given region. Yang, Ding, and Hao present a multi robot coverage path planning approach by dividing the given area into cells with a Voronoi diagram. The diagram is constructed utilizing the inital start positions of robots in the team. Subsequently, a Modified Bacteria Foraging Optimization (MBFO) algorithm is utilized to navigate the robots inside their respective cells. That is, BFO is inspired by the natural foraging behavior of the *E. coli* bacteria that consumes nutrients in the intestines. Interestingly those bacteria are always found where the highest nutrients concentration is. This strategy is adapted to the robot by emulating uncovered cells as high nutrient concentration which is reduced when the robot crosses over the respective area. The MBFO takes the different cells generated by the Voronoi diagram into consideration and enables multi robot coverage.

Figure 2.9a illustrates the initial step where each dot represents the start point of the respective robot as well as the initial seed for the Voronoi diagram. Each robot has its desired cell. Next, comes the optimization procedure where each robot is dragged towards uncovered cells while simultaneously staying in their cell bounds and avoiding collisions with other robots[17].



(a) (b)

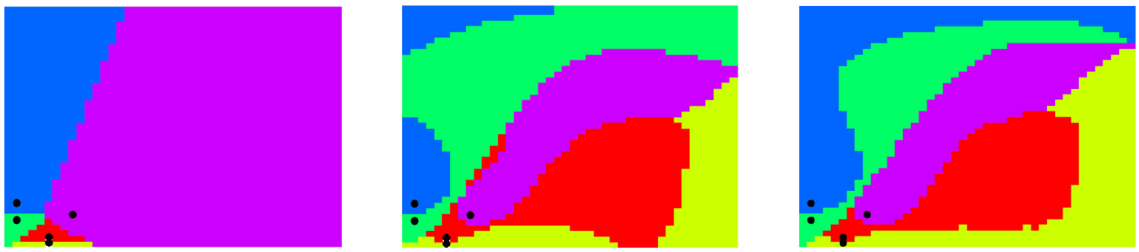
Figure 2.9: MBFGO at initial stage (a) and during the optimization (b)

Dividing the area such that each robot has its own terrain it can work on reduces the complexity of the path generation. This has the effect that robots do not interfere with each other. However, while generating the cells it is a challenging task to assign the same amount of space to each robot, especially if their initial positions are close to each other. Kapoutsis, Chatzichristofis, and Kosmatopoulos tackle that problem by optimizing each robots region size. This ensures that each robot gets a "fair share" $f = l/n_r$ of the map where l is the amount of uncovered cells and n_r the amount of robots. During the procedure *Divide Areas Based on initial Robot Position* (DARP) the fair share error J is minimized with *Gradient Descent*:

$$J = \frac{1}{2}\sum_{i}^{n_r}(k_i - f)$$

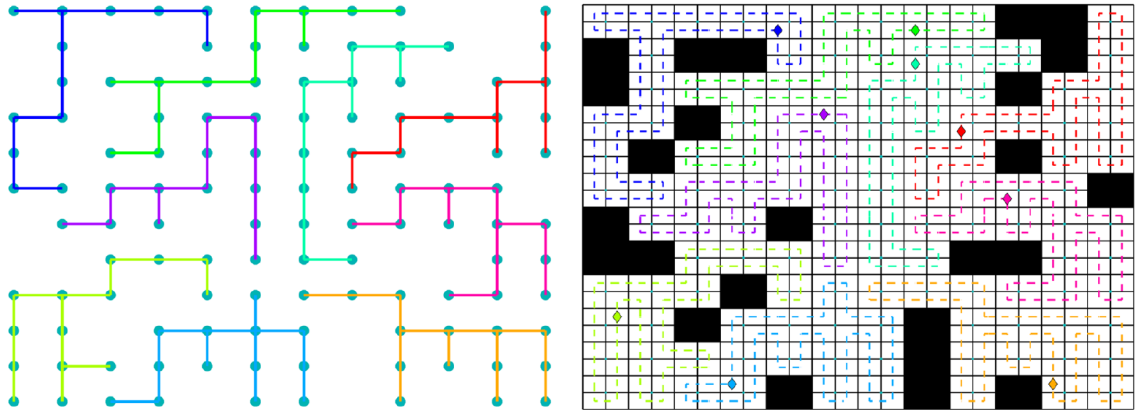
where k_i describing cells assigned to robot i. The optimization procedure is illustrated in Figure 2.10.

In the final step the Spanning Tree Coverage[18] (STC) algorithm is applied to the clustered regions. This constructs a spanning tree with the goal to cover each cell exactly once. Figure 2.11 visualizes the spanning tree for a team of robots in 2.11a and the resulting coverage paths 2.11b.



(a) T=0 (b) T=80 (c) T=260

Figure 2.10: DARP optimizing procedure at different time steps T



(a) Minimum spanning tree (b) Resulting coverage path

Figure 2.11: Visualization of STC.

2.1.4 State of the Art

The short literature review showed that there are different interesting approaches in order to solve the ACP with coverage path planning. Due to the huge quantity of available approaches only a fraction is selected to present the most common components involving a coverage path planning task. For the interested reader the survey [7] describes different variants of the cell decomposition. More recent approaches involving the 2D as well as the 3D case of single robot coverage path planning are collected in [14]. Finally [6] resumes the most recent works in multi-robot coverage path planning.

Solving CPP is closely related to solving the Coverage Salesman Problem (CSP). This problem is a generalization of the Traveling Salesman Problem (TSP)[19]. In the CSP the salesman tries to visit as much cities as possible but instead of visiting a city directly the salesman is only required to visit the city in some defined neighborhood. This is

related to the CPP where the coverage that is generated by a path defined by viewpoints is the range for the neighborhood and the viewpoint the position that is visited by the salesman. Furthermore, the TSP is a well investigated problem which is a special case of the Vehicle Routing Problem (VRP)[20]. The VRP takes several vehicles into account and tries to find the shortest route for each vehicle and simultaneously visits all given nodes. Because VRP and CSP are generalizations of the TSP which is NP-Hard and so are VRP and CSP. Therefore generating viewpoints and trying to connect them in order to achieve complete coverage (CSP) and furthermore coordinate several vehicles in order to minimize the travel path for each vehicle while covering the area (VRP) means that the complete CPP process is also NP-Hard.

In order to solve such a complex problem a good heuristic optimization procedure needs to be utilized. The procedure of choice for this work are Genetic Algorithms (GAs) that have proven them self to be efficient in large and complicate search spaces[21]. In the next section the concept of Genetic Algorithms as well as their working principle is presented.

2.2 Genetic Algorithm

The Genetic Algorithm (GA) is a meta-heuristic optimization approach, inspired by natural evolution such as selection and recombination and is inspired by Darwin's principle "Survival of the fittest"[22]. Each GA works with a population of individuals that are called genomes. Genomes contains one or more chromosomes that are composed of different gens.

Genome representation describes the structure and composition of the information encoded in the chromosomes. Holland first described a binary encoding where the chromosomes contain long strings of zeros and ones in order to simulate the natural DNA sequence[23]. Moreover, different types of representations are possible such as real valued or even vectored encodings. As long as it is possible to define a fitness function and implement the GA operators various representations are feasible. This makes the GA very flexible and adaptable to a wide range of problems.

Fitness Function is utilized to describe the quality of such a solution or genome. Ultimately the fitness function is the most important part because the generated fitness value is used to steer the search process. Furthermore, the fitness gives a formal description of the optimization problem and is therefore also called the objective function. The search process is conceptually divided into two parts, the exploration or global search where new solutions in the search space are randomly explored and the exploitation or local search that recombines already known solution to new ones.

GA operators are utilized to select, recombine and alter the genomes in the population in order to generate offspring that replaces or competes against individuals from the previous generation. Each iteration of the GA procedure emulates one generation where the most fit individuals are determined by a selection procedure. The selected individuals are combined with a crossover method that generates two offspring individuals from two parents. Subsequently, the offspring undergoes a mutation procedure that randomly alters gens in the chromosome. Finally, the offspring builds the new population for the next iteration.

GA challenges include building a suited genome representation, finding a good fitness function for the problem and balancing all parameters for the operators correctly to optimize the search process. Furthermore, it is desired to keep a good diversity in the

population to ensure that the solution space is adequately searched. The GA operators have direct influence on the diversity. Selection has a negative impact on it by applying a so called selection pressure that removes unfit individuals from the population. On the other hand the mutation operator inserts diversity by randomly altering already known solutions and thus exploring the search space. This results in new individuals which increases the diversity. A reduction in diversity is normal especially at the end of the optimization when a good individual cannot be outperformed. However, losing too much diversity in early generations traps the population in an local optima where one individual prevents the population from exploring the search space. This is called premature convergence and needs to be avoided in order to properly explore the search space[24].

2.2.1 Selection Heuristic

The selection procedure selects individuals from the population. It utilizes the genomes fitness to calculate a selection probability or performs a tournament between different genomes in order to select the best individuals.

2.2.1.1 Tournament Selection

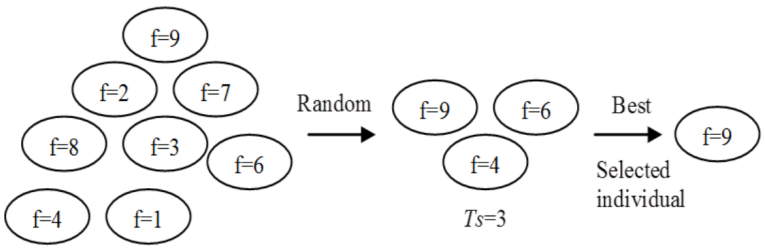


Figure 2.12: Illustration of the tournament selection [25]

In the tournament selection procedure, random individuals will be drawn uniformly from the gen-pool and compete for producing the next generation. Usually two individuals will be selected which is called binary tournament, whereas the number of individuals selected for tournament is referred to as tournament size TS. The individual with the highest fitness will be selected for the next population (or crossover). Figure 2.12 Illustrates the process.

First individuals are randomly selected from the population. Here TS is set to three. Thus three individuals compete against each other. The one with the highest fitness (f=9) wins and is selected as one individual for reproduction (crossover).

This procedure is quite popular because of the simplicity as well as the direct influence on the selection pressure. Increase TS increases selection pressure where $TS = \#Population$ means elite and $TS = 1$ is the same as random selection. However, a few drawbacks need to be taken into consideration when applying this procedure. Especially the selection pressure is a double edged sword because choosing the pressure to high could lead to premature convergence because less fit individuals are dominated by already converged ones and therefore hinder the exploration process. On the other hand, choosing selection pressure to low could prevent successful convergence, thus hinder sufficient exploitation. Additionally, the random selection of individuals from the population for tournament could cause that some individuals in the population are never selected for the tournament and therefore do net get a chance of reproduction.

2.2.1.2 Proportionate Roulette-Wheel Selection

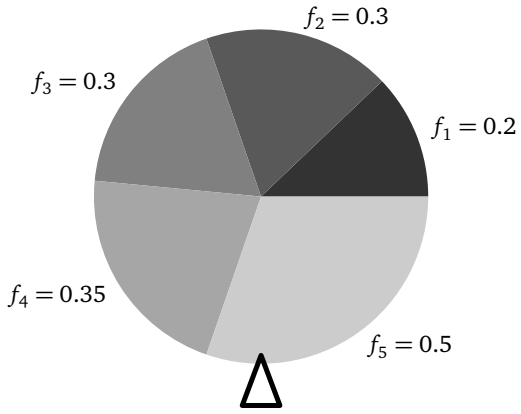
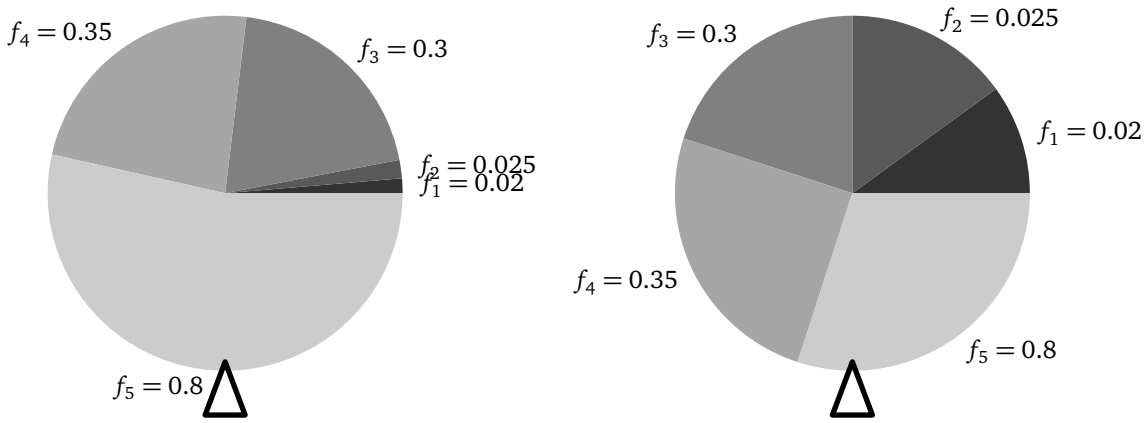


Figure 2.13: Proportionate Roulett-Wheel Selection

In this selection scheme, genomes are projected by some mapping function around the circumference of a roulett-wheel. Genomes are selected by spinning the wheel and choosing the one, the pointer on the wheel points to. The performance of this scheme is entirely determined by the projection function. In the Proportionate Roulette-Wheel Selection (PRWS) the probability p_i is calculated by using the genomes fitness values: $p_i = \frac{f_i}{\sum_{n=1}^{N} f_n}$ Where f_i is the fitness and N is the amount of individuals that are projected to the wheel. Each genome is then projected to one segment of the wheel. Higher fitness values result in bigger segments on the wheel and thus in higher selection probability. An illustration of the projection is shown in Figure 2.13



(a) Proportionate RWS

(b) Ranked RWS

Figure 2.14: Behaviour of proportionate and ranked Roulette-Wheel Selection (RWS) when fitness values are widely spread

This process is repeated until a predefined amount of individuals is selected. Furthermore individuals with high fitness are more likely to be selected which produces selection pressure that improves the overall solutions. One major advantage of this selection is that each genome has a chance in contributing to the next generation.

However, the projection function performs poorly if individuals with high fitness values appear in int population as illustrated in Figure 2.14a. The high fitness is projected to a too wide segment on the wheel which drastically reduces the selection probability for all other individuals. Such individuals tend to quickly take over the population. Especially in early stages of the optimization this quickly leads to premature convergence. Another drawback appears when all individuals in the population have very similar fitness values. This hinders the population to move forward and thus prevents convergence towards an optimal solution.

2.2.1.3 Rank-based Roulette-Wheel Selection

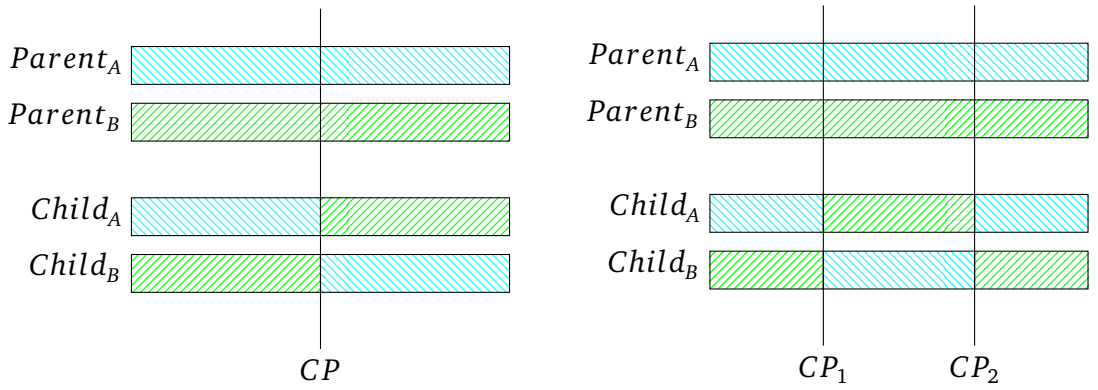
In order to tackle the two major drawbacks of the PRWS, one can use a projection function that is not directly dependent on the fitness value. Instead each genome is assigned a rank. The higher the fitness value the higher the rank. The position is

assigned can be used to calculate a selection probability. This is also referred to as rank selection. Most prominent is the linear rank selection:

$$Rank(Pos) = 2 - SP + \left(2 \cdot (SP-1)\frac{Pos-1}{n-1}\right)$$

The position (Pos) is projected to a value in the interval $[2-SP, SP]$ where SP is the selection pressure in range $1 \leq SP \leq 2$. Next, the rank value is used to calculate the selection probability p_i. As consequence, highly fit individuals will not supersede less fit individuals and therefore increase the selection probability of less fit ones. This behavior is illustrated in Figure2.14. 2.14b shows the PRWS where one individuals f_5 features a very high fitness value. The same individuals, projected with the Ranked Roulette-Wheel Selection (RRWS) is illustrated on the left. Hence the ranked projection results in a more evenly distribution of individuals on the wheel without the drawback of individuals with too high fitness taking over. This behavior comes with the cost of selecting the correct bias which is directly controlled by SP. The higher SP the greater the selection pressure[25].

2.2.2 Genome Recombination



(a) Single-Point Crossover

(b) Dual-Point Crossover

Figure 2.15: Illustration of different genome recombination strategies

The recombination or crossover of two chromosomes is usually applied after the selection process. It follows the natural crossover procedure where combining two fit individuals will result in equally fit or better ones. During the Selection, the best individuals are selected and serve as parents for the new offspring. This offspring will either replace or compete against the parents and therefore produce even fitter

individuals. The general idea behind recombining two individuals is to construct offspring by using parts of the parental genetic material. Each parent either equally or partially contribute to the gen material. How much and what genetic material is shared depends on the utilized strategy. After the crossover two offspring individuals are generated.

Single-Point Crossover is designed to share equal amounts of genetic material between the parents. Therefore, a cross point CP is selected to divide the parents chromosome at a specific position. The actual generation of CP remains application dependent but usually it is located at the center of the parents chromosomes. Figure 2.15a illustrates the process where two parents $Parent_A$, $Parent_B$ share genetic material in order to generate the offspring $Child_A$, $Child_B$.

Dual-Point Crossover utilizes two instead of one cross point in order to determine a region of gens that should be shared. Usually the size of the region is randomly chosen as well as the start positions. This enables more possibilities that single point crossover for recombination of two individuals. Figure 2.15b shows the general setup of the crossover where the cross points CP_1, CP_2 mark the region that is shared in the offspring individuals.[26]

2.2.3 Parameter Setting

Genetic Algorithms utilize different operators to explore the given search space. Those operators select good and reject bad solutions based on a fitness value. The solutions are generated by recombining and randomly mutating already existing solutions. Each of the selection, crossover and mutation operators utilize some probability that guide the search process over a population of a specific size. Those values that contribute in directing the search process of the GA are called the parameter setting.

Before diving into tuning those settings there are correlations and known working combinations that should be taken into consideration before tuning those parameters.

The population size directly impacts the quality and duration of the GA. The major goal of a GA is to generate a good or optimal solution in the population that meets the expectations for a given problem. How quickly such an individual is found directly depends on the size of the population. Small population sizes usually require more generations to converge but due to the lower computational load it is much quicker

than larger population sizes. However, those populations suffer from small diversity and thus are in danger to premature converge because the search space is not sampled broadly enough.

On the other site larger populations are computationally more expensive but feature more diversity as well as a higher probability of already good solutions in the initial population. This results in higher quality of the resulting solutions. Finally, one needs to find a good trade of on the population size which is often in range $Pop_{min} \in [50, 100]$.

Crossover and mutation needs to be adapted accordingly where good values for the crossover probability are $P_c \in [0.6,0.95]$. On the other hand the mutation operator should only be applied in a small manner $P_m \in [0, 0.5]$. In the concept of global and local search, it is beneficial to utilize small values for the crossover and higher values for the mutation when dealing with small population sizes because the global search is preferred. The opposite is true for larger sizes there the local search should be encourage. Thus crossover promotes exploitation or local search where mutation performs the exploration or global search[27].

Selection plays an important role in determining how fast the population converges to an optimal solution. It applies the so called selection pressure by promoting good and neglecting bad individuals. The selection pressure can be indirectly applied as for example in the PRWS or actively by the tournament size TS in Tournament or ranking probability which is also referred to as selection pressure SP in the RRWS Selecting the right parameters for the pressure is a delicate procedure because too low selection pressure could prevent the population from reaching an optimum but too high pressure could result in premature convergence. For TS the following range is suggested $[1, \frac{Pop_{min}}{6}]$ where $TS = 1$ is no pressure at all and just a random individual is selected and $\frac{Pop_{min}}{6}$ causes the population to almost instantly loose diversity[28].

2.2.3.1 Parameter Variation

The majority of parameters contained in the parameter setting are described in the previous section. Eiben, Hinterding, and Michalewicz investigated the influence of different parameter settings and showed that the optimal parameter setting varies during the optimization[29]. This is mainly caused by 1) Incorrect application of selection pressure in a way that strong individuals quickly take over the population, 2) Too low mutation rate such that too much diversity is lost during selection and

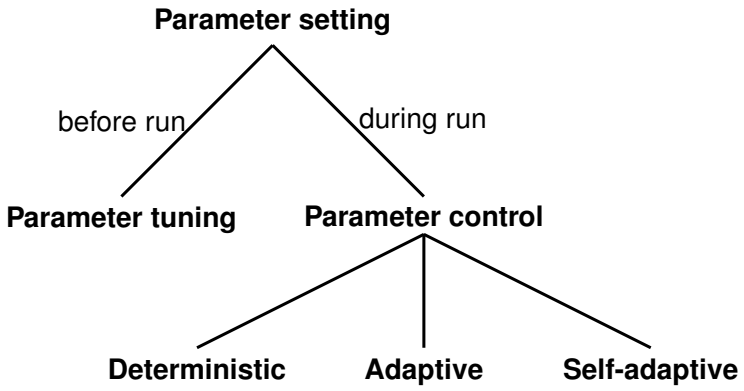


Figure 2.16: Global taxonomy of parameter setting[29]

crossover that cannot be recovered and 3) Ineffective crossover when individuals in the population are too similar to each other such that the recombination generates very similar individuals[28].

Therefore, varying the parameter during the optimization process could yield better results in the overall performance and solution generation. Figure 2.16 illustrates the different variations on how the parameter setting can be selected. The first and most common strategy is to select the parameter setting before the run and tune them until a sufficient solution is reached. This is referred to as parameter tuning. Parameter control on the other hand adapts the initial parameter setting during the run. Deterministic control works by varying the parameters e.g. dependent on the iteration. It is intended to work without any direct feedback from the performance of the population. This has the benefit, that those parameters always behave the same. One example of deterministic adaptation of crossover and mutation rate is presented in [27]. The crossover $P_{c,i}$ and mutation probability $P_{m,i}$ are calculated by taking the current iteration step i into account.

$$P_{m,i} = \frac{Iter_i}{Iter_{max}}, Iter_n = \sum_{i=1}^{n} i \tag{2.1}$$

$Iter_n$ gives the value of the actual and $Iter_{max}$ the maximal amount of iterations. The following holds true for the crossover probability:

$$P_{c,i} = 1 - P_{m,i} \tag{2.2}$$

Therefore when $P_{m,i}$ increases, $P_{c,i}$ decreases[27].

Adaptive and self-adaptive approaches work by taking the current state of the population into account and adapt the parameter setting accordingly. Furthermore self-adaptive

means that instead of utilizing a global parameter e.g. mutation probability, each individual carries its own factor that determines how it is treated by different GA operators. [28] presents an adaptive approach that utilizes the standard population diversity (SPD) and healthy population diversity (HPD) to adapt selection pressure as well as mutation and crossover probabilities. The crossover probability is calculated as follows:

$$P_c = \left[\left(\frac{SPD}{SPD_{max}} * (K_2 - K_1)\right) + K_1\right] \tag{2.3}$$

Subsequently P_c varies between K_1 and K_2 where $K_1 := 0.4$ and $K_2 := 0.8$. Furthermore the mutation probability is determined by two factors, the diversity (SPD) and the fitness.

$$P_m^{Diversity} = \frac{SPD_{max} - SPD}{SPD_{max}} * K \tag{2.4}$$

$$P_m^{Fitness} = K * \left(\frac{f_{max} - f}{f_{max} - f_{min}}\right) \tag{2.5}$$

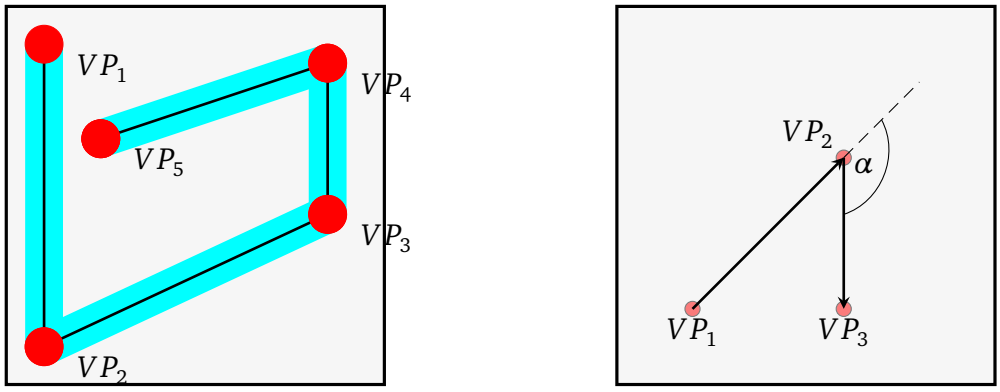
$$P_m = \frac{P_m^{Diversity} + P_m^{Fitness}}{2} \tag{2.6}$$

Where $K = 0.5$ gives the upper bound for $P_m^{Diversity}$.

The calculation steps for SPD and HPD are omitted because the diversity calculation is based on a mean individual. However, the next chapter presents a GA architecture that utilizes a custom diversity calculation in Section 3.2.3 which is used instead of SPD.

3 Viewpoint and Path Generation

The CPP is composed of two steps, the Viewpoint VPs generation and optimization of the visiting order. With help of Genetic Algorithms the generation procedure and combination of VPs is combined in the genome encoding and thus enables simultaneous optimization by only relying on the information of the environment. Therefore, this chapter concerns with the environment components and how to estimate them in order to calculate the desired fitness function. Furthermore, several different functions are presented as well as the encoding of the environment with the help of genomes. Finally four different GA architectures are presented to handle the actual optimization.



(a) Path representation in continuous space

(b) Rotation angle α

Figure 3.1: Visualization of path described by five VPs . Coverage is marked blue and VPs red.

3.1 Environment Conditions

Viewpoints are the essential component for generating a path. In the application scenario of a cleaning robot they describe coordinations on a plane, $VP \in \mathbb{R}^2$. A robot

in a straight line from one VP to the next until all VPs are visited. Grouping several VPs together in a specific order yields the path PA:

$$PA = (VP_1, VP_2, \ldots, VP_N)$$

where N is the maximal amount of VPs . The robot R is described by the following attributes: cleaning width R_{width}, drive speed R_{speed} and rotation speed R_{rot}.

One example of a path described by VPs is illustrated in Figure 3.1a. The red dots are the VPs with the index indicating their visiting order. The blue marked region is the coverage that is produced by following the VPs. In order to assess the quality of a path its length PA_{len} needs to be calculated. This is easily done by summing up the distances between the VPs :

$$PA_{len} = \sum_{i=1}^{N-1} \|PA_{i+1} - PA_i\| \tag{3.1}$$

Covered area PA_{cov} and time PA_t needed by the robot can be estimated accordingly:

$$PA_{cov} = PA_{len} \cdot R_{width} \qquad PA_t = \frac{PA_{len}}{R_{speed}}$$

The last component is the rotation angle between the VPs . As illustrated in Figure 3.1b α describes the rotation angle between $V_1 = PA_{i+1} - PA_i$ and $V_2 = PA_{i+2} - PA_{i+1}$. In order to calculate α, the respective angle between the vectors and X-Axis is calculated:

$$\beta(V) = atan2(\frac{V}{\|V\|}) \cdot \frac{180}{\pi}$$

Subsequently the rotation angle is given by $\alpha = rotAngle(\beta(V_1) - \beta(V_2))$ where

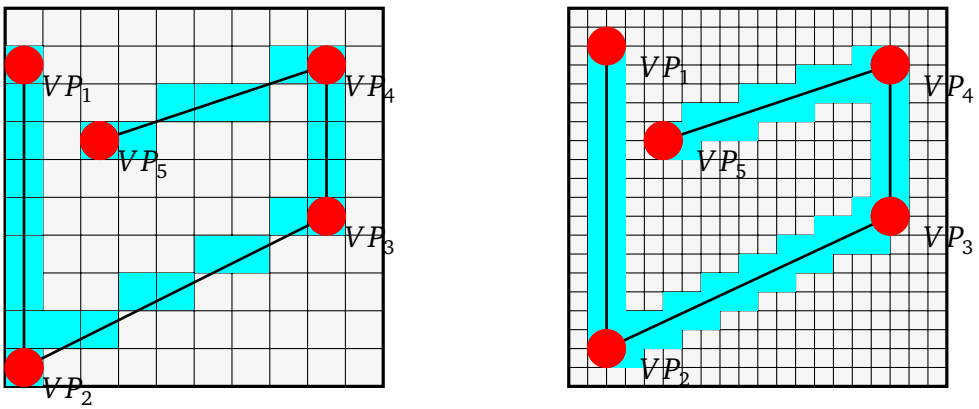
$$rotAngle(d) = \begin{cases} d > 180, & d - 360 \\ d < -180, & d + 360 \\ else & d \end{cases} \tag{3.2}$$

3.1.1 Grid Based Model

The previous representation works purely in continuous space and is accurate in terms of measuring travel time and allows free rotation angles without any restrictions to a specific neighborhood. Still a few drawbacks remain in terms of measuring crossing paths and detecting obstacles. In order to calculate those parameters instead of choosing a continuous map a discrete grid map is utilized to represent the environment. That is,

map M is represented as a grid where each cell of the grid represents a part from the environment of the size R^2_{width}.

Figure 3.2a shows the same VPs as before but this time represented inside M . Cells marked with blue represents covered cells and the red dots mark VPs . This discrete model of the environment makes it fairly easy to detect crossing path segments and also obstacles. Furthermore the model is easily shareable between different robots, e.g. to share detected objects or already covered area. However this model comes with a major drawback that originates in the rasterizuation itself[30]. Through the process of rasterization the path in continuous or also called vector space (Figure 3.1a) is projected to the discrete grid map (Figure 3.2a). This procedure is error-prone because grid-cells can only be in one state at a time. For example, they can either be covered or uncovered. This makes it hard to represent partially occluded cells in a single map. One solution to compensate this effect is to increase the resolution of the map M_{res} which is traded for with higher computational costs and memory usage. Figure 3.2b shows the effect of increasing the grid resolution. The presented approach combined with the increased resolution delivers enough accuracy to evaluate the overall optimization procedure. Furthermore, the approach is designed to be as modular as possible so the strategies for estimating the coverage are easily exchangeable if more accuracy is desired.



(a) $M_{res} = R_{width}$ (b) $M_{res} = R_{width}/2$

Figure 3.2: Path with increased resolution

3.1.2 Path Performance

First it needs to be clarified what performance measurements are available and what information has to be processed before it can be used for the optimization. As known from the previous two Sections the grid based model allows access to the overall covered region cov_{pix}, the crossing segments on the path $cross_{pix}$, overall path length PA_{len}, intersections with obstacles obj_{pix} and the rotation angle between two consecutive robot movements as described in equation 3.2. The pseudo code in Listing 3.1 abstracts the data collection process the robot preforms when driving over the map. The first line means iteration over each pixel that the path contains. This is a little bit more complex that just iterating over a list of pixels. In the actual application a rectangle between two VPs is constructed with the with equal to R_{width}. Afterwards all pixels that are covered by the rectangle are selected. This is similar as presented in Figure 3.2b.

Furthermore, several maps are utilized with different purposes. $M_{obstacle}$ provides information about obstacles and walls whereas $M_{traveled}$ is used to incrementally mark the covered regions and to capture if one pixel was covered multiple times.

Both maps are equal in size and resolution. Pixel that are not accessible are marked with a value greater zero. Counting all pixels that are accessible corresponds the free area of the environment A_{pix}. If the robot encounters such a pixel obj_{pix} is incremented. If the pixel is accessible the robot checks if it was seen before. If so $cross_{pix}$ is increased together with cov_{pix} and the $M_{traveled}$ index.

Listing 3.1: Pseudocode for parameter calculation

```
1 for p in Path:
2     if M_obstacle(p) > 0:
3         obj_pix++
4     else:
5         if M_traveled(p) > 0:
6             cross_pix++
7         cov_pix++
8         M_traveled(p)++
```

Subsequently the rotation time needs to be estimated. This is done afterwards because the robot can change the position of VPs e.g. if they are outside the map boundaries. The calculation for T_{rot} is described in equation 3.3. That is, rotation angle between

two consecutive viewpoint transition where each transition is simply a robot move from PA_i to PA_{i+1}.

$$T_{rot} = \begin{cases} R_{rot} > 0, & \sum_{i=1}^{N-2} \frac{rotAngle(PA_{i+1} - PA_i, PA_{i+2} - PA_{i+1})}{R_{rot}} \\ else & 0 \end{cases} \tag{3.3}$$

The only thing left is to convert the collected pixel data into the required metric scale. This is quickly illustrated in the following equations

$$cov_{[m^2]} = cov_{pix} \cdot M_{res}^2 \tag{3.4}$$

$$cross_{[m^2]} = cross_{pix} \cdot M_{res}^2 \tag{3.5}$$

$$A_{[m^2]} = A_{pix} \cdot M_{res}^2 \tag{3.6}$$

The previously described steps are the essential points of the path evaluation. It gathers information that describe the path length, time to follow the path, rotation time, approximation of covered area and the corresponding crossing area.

3.2 Genome Representation

The genome representation is tightly coupled to the environment conditions where it encodes the path that a robot can follow. Furthermore, to clarify some biological nomenclature: A genome is composed of one or several chromosomes where each one holds one or more gens.

Therefore the VPs needs to be encoded somehow in the chromosome. One example that motivates the following genome representation is presented in [31]. Instead of creating a coverage path they utilize a chromosome composed of VPs that is intended to generate the shortest path from a given start to an end point. Furthermore the chromosome is not constrained in length and thus operations from the GA can randomly insert or remove VPs .

In this work a modified version of this representation is created. Instead of modeling gens directly as VPs they are modeled as actions. Actions follow the basic concept of describing the position of the next VPrelative to the current position. Thus each action contains a distance factor and direction. This representation is basically exchangeable by a sequence of VPs because one action is described by the distance AC_{dist}, direction AC_α between two VPs and a start position AC_{pos}. The direction is given in degree

relative to the X-Axis and the distance in meter. Each genome contains only one chromosome, therefore both are used interchangeably to describe the sequence of actions of the chromosome: $G = (S, AC_2, \ldots, AC_{G_{len}-1}, E)$ where G_{len} is the current length of the chromosome, S describes the start action, E the end action and AC_i the transition at i-th position in the chromosome. The start and end actions are special cases of transitions because they contain positions that are fixed from the beginning. For example the start action cold correspond to a docking station where the robot is placed in idle. The end action could be the position of another or the same station and therefore ensures that the robot again ends at a predefined location.

The inference procedure of generating VPs for the robot to follow is accomplished by first calculating the direction vector d^t of the current action

$$d^T = (cos(AC_{i,\alpha} \cdot \frac{\pi}{180}), sin(AC_{i,\alpha} \cdot \frac{\pi}{180})) \tag{3.7}$$

and subsequently updating the position of the consecutive action.

$$AC_{i+1,pos} = AC_{i,pos} + \vec{d} \cdot AC_{i,dist} \tag{3.8}$$

This is propagated through all actions until all positions are updated. The VPs are then contained int eh respective AC_{pos}.

In summary, the presented genome representation G consists of a chromosome that is not limited in length and contains so-called actions. Each action is composed of a starting position, a direction and a distance. Two special actions, located at the beginning and at the end of each genome, mark the start and end positions, which are not changed during optimization.

3.2.1 Fitness

The fitness function is the essential part of a genetic algorithm because it describes the optimization problem in form of an equation. The genome encoding provides a solution or enables to derive a solution that solves the equation. Furthermore, genomes are ranked by the fitness function and can either preform in a “good” or “bad” manner. Depending on the fitness value good individual are preferred in the next optimization step. Hence the goal of the genetic algorithm is to maximize the fitness function.

3.2.1.1 Optimization Goal

The previous Section describes genomes as a composition of action that generate a coverage path. Furthermore, through evaluating that path environment parameter

like all visited area $cov_{[m^2]}$, the free area $A_{[m^2]}$, crossing path segments $cross_{[m^2]}$, and the rotation time T_{rot} are exposed and can be utilized for the fitness calculation. The fitness formalize two objectives that should be optimized, the overall coverage, the path length and therefore the time needed to complete the coverage task and the obstacle avoidance.

The first condition can be approached by calculating the actual coverage cov_{actual} and then create the relation between it and $A_{[m^2]}$.

$$cov_{actual} = cov_{[m^2]} - cross_{[m^2]} \tag{3.9}$$

$$cov_{final} = \frac{cov_{actual}}{A_{[m^2]}} \tag{3.10}$$

The final coverage relation factor $cov_{final} \in [0,1]$ states how much area is actually covered without crossing segments. Zero means no area is covered at all and one that the complete region was covered.

Determine the optimal value for the coverage is straight forward. However this is not the case for the time condition which is contrary to the first because generating more coverage requires to increase PA_{len} and therefore increases the time that is needed to follow the path and vice versa. Therefore, this condition is implicitly modeled by minimizing the $cross_{[m^2]}$ parameter. This would exclude the redundant parts of the path and therefore reduce the overall path length and increase quality. The general principle is to estimate an optimal duration that is needed to cover the region described by $cov_{[m^2]}$ which is referred to as $t_{optimal}$

$$t_{optimal} = \frac{cov_{actual}}{R_{speed} \cdot R_{width}} \tag{3.11}$$

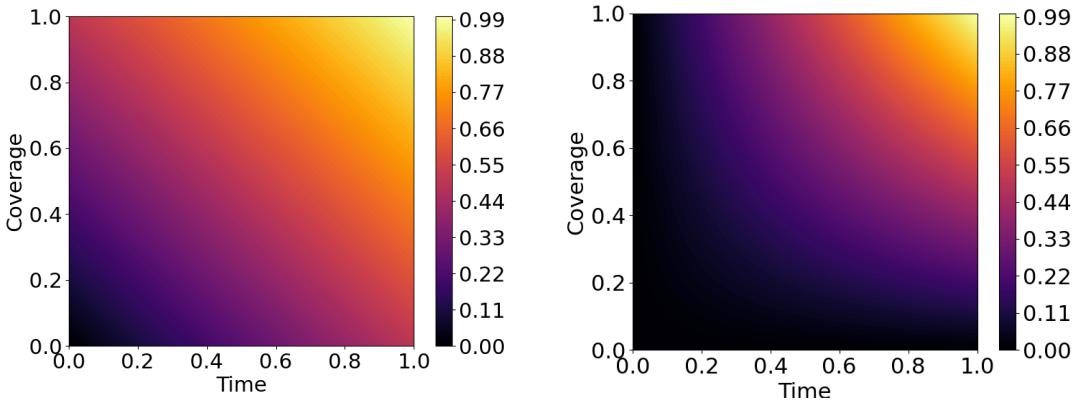
Similar for the actual time value t_{actual} where the crossing segments are considered.

$$t_{actual} = \frac{cov_{[m^2]}}{R_{speed} \cdot R_{width}} \tag{3.12}$$

Subsequently the final time value t_{final} is calculated from the relation between actual and optimal time.

$$t_{final} = \frac{t_{optimal}}{t_{actual}}$$

The final time value t_{final} ranges between zero and one where one means no redundant area was covered whereas zero means that no area was covered at all. In fact if a path has maximal or infinite redundant parts t_{final} converges to zero because the t_{actual} increases.



(a) Fitness projection f_1

(b) Fitness projection f_2

Figure 3.3: Proposed fitness projection functions. X-axis gives the time and y-axis the coverage factor and the color indicates the fitness level

In order to further minimize the coverage time the robot rotation time is included in the final time calculation.

$$t_{final} = \frac{t_{optimal}}{t_{actual} + T_{rot}} \tag{3.13}$$

Including the rotation time results in more realistic paths where excessive rotations and turns are reduced.

The coverage and time factor are used in the next section in order to generate a single fitness value. Subsequently a penalty is introduced that reduces the fitness drastically if intersection parts with obstacles occur.

3.2.1.2 Fitness Projection

In the final step a good fitness function is needed that projects the time and coverage values to fitness value. Two different functions are proposed in order to control the search process:

$$f_1(x,y) = 0.5x + 0.5y \tag{3.14}$$

$$f_2(x,y) = x \cdot y \tag{3.15}$$

All functions project the fitness to a value between zero as one. Same holds true for the input x and y that correspond to the final time and coverage. The co-domain of the linear function f_1 is presented in Figure 3.3a. The coverage is given on the y-axis

and the time on the x-axis. The nature of f_1 is that it behaves in a linear manner. Therefore, individuals with very high time value and very low coverage can gain a reasonable fitness and vice versa. The idea behind this is that individuals are widely spread over the fitness surface and therefore achieve overall better performance in the path generation process. However, the second non-linear function f_2 does not promote wide spreading individuals but instead encourages them to improve both in their time and coverage value at the same time. This should result in individuals that are more centered and furthermore prevents the individuals from only optimizing the time or the coverage part. The co-domain is illustrated in Figure 3.3b

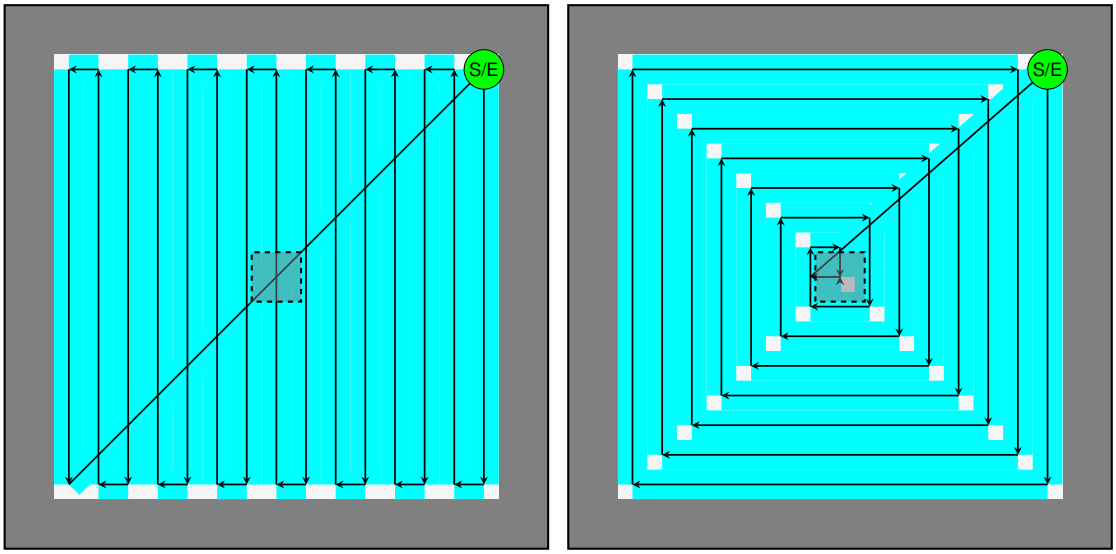
Obstacle Avoidance is achieved by penalizing paths that intersect with obstacles. When a path crosses an obstacle the obj_{pix} counter is increased. Crossing an obstacle results in a collision path and should be avoided. However at the early stages of the optimization nearly all paths collide with objects due to random initialization of the action. Therefore the penalty should encourage the optimization to prefer paths that do not collide with obstacles.

$$f_{obstacle}(f,x,y,obj_{pix}) = \frac{f(x,y)}{2 * obj_{pix}} \tag{3.16}$$

3.2.2 Fitness Calculation

This section illustrates the evaluation procedure of a genome and presents some output values for a given example pattern and obstacles in order to investigate the behavior of $f_{obstacle}$ the final fitness function used in the optimization

The genome evaluation decodes the chromosome to a coverage path and projects it to a map in order to evaluate the environment parameters. Subsequently, the environment parameters are used to calculate the fitness. Two well established coverage patterns are utilized[32] such as the boustrophedon movement and spiral pattern. Both are illustrated in Figure 3.4 where blue shows the covered and white the uncover regions, arrows mark the movement direction and the green circle shows start and endpoint. Furthermore the dashed rectangular shape in the center of each map represents an obstacle. In the small test scenario the fitness is first calculated without and then with the obstacle placed on the map in order to emulate an intersection with an object. One important thing to know is that during evaluation the robot only marks a region as covered when it moves and not when it rotates. Therefore, small rectangular artifacts of uncovered area appear.



(a) Boustrophedon Pattern (b) Spiral Pattern

Figure 3.4: Two different coverage pattern[32] as they are generated by the genome evaluation with obstacle place in the middle of the map

Table 3.1 notes the results for the boustrophedon and 3.2 results for the spiral pattern. In general utilizing the f_1 always results in higher fitness values ($f_{obstacle}$) than f_2 for same genome which is simply caused by the non-linearity of f_2. Furthremore both pattern reach nearly full coverage except for the uncovered artifacts which makes it hard to reach complete coverage. Still the cov_{final} component expresses the coverage as expected.

Placing an obstacle also reduces the overall coverage because as intended path segments that intersect with an obstale should not contribute to the final coverage. Furthermore $f_{obstacle}$ is reduced immensely because of the detected intersecting pixel in the path obj_{pix}. One thing that is not taken into consideration is that the actual coverage should not change when an obstacle is detected. This is done here for pure illustration purposes to show, that obstacle regions do not contribute to the coverage. In the actual application those obstacle are known beforehand and thus excluded in $A_{[m^2]}$ such that intersecting an obstacle only impacts the final $f_{obstacle}$ and not cov_{final}.

The time factor t_{final} is directly impacted by the cross coverage and R_{rot}. When the rotation is weighted thus the R_{rot} is greater zero, t_{final} is reduced and therefore $f_{obstacle}$, too. One important reason for the not optimal t_{final} is again caused by the coverage generation on the map. The rectangle that is drawn from a start position e.g. beginning

Table 3.1: Results for the final fitness $f_{obstacle}$ and variable conditions with and boustrophedon coverage pattern

obj_{pix}	Function	R_{rot}	$f_{obstacle}$	cov_{final}	t_{final}
0	f_1	0	0.922019	0.96	0.884038
0	f_1	6.28319	0.91871	0.96	0.87742
0	f_2	0	0.848677	0.96	0.884038
0	f_2	6.28319	0.842323	0.96	0.87742
44	f_1	0	0.010444	0.947654	0.890487
44	f_1	6.28319	0.0104053	0.947654	0.883685
44	f_2	0	0.00958948	0.947654	0.890487
44	f_2	6.28319	0.00951623	0.947654	0.883685

Table 3.2: Results for the final fitness $f_{obstacle}$ and variable conditions with and spiral coverage pattern

obj_{pix}	Function	R_{rot}	$f_{obstacle}$	cov_{final}	t_{final}
0	f_1	0	0.953991	0.973827	0.934154
0	f_1	6.28319	0.950108	0.973827	0.92639
0	f_2	0	0.909705	0.973827	0.934154
0	f_2	6.28319	0.902144	0.973827	0.92639
36	f_1	0	0.013216	0.962469	0.940637
36	f_1	6.28319	0.0131625	0.962469	0.932929
36	f_2	0	0.0125741	0.962469	0.940637
36	f_2	6.28319	0.0124711	0.962469	0.932929

of an arrow, intersects with the end of the previous coverage rectangle generate by the predecessor action and causes crossing segments.

To summarize, $f_{obstacle}$ is the final fitness function that utilizes a fitness projection f_1 or f_2 to generate a scalar value from cov_{final} and t_{final}. Furthermore, the rotation speed of the robot R_{rot} and the obj_{pix} are taken into consideration. The rotation intends to create more realistic path with overall smaller rotation angles where obj_{pix} is utilized to determine if a path intersects with an object and therefore penalizes the resulting final fitness.

3.2.3 Diversity

The diversity is distance measure of similarity between two genomes. High diversity in a population means that the individuals or genomes differ from each other. Low diversity on the other hand states that the individuals are similar.

Usually the fixed size of the chromosomes enables to perform a comparison between gens with a distance measure like euclidean or hamming distance[33]. However this is not the case for variable genome length where structural similarities are shrouded. This is caused by the crossover that places gens in a random order in the chromosome in order to explore different path variations. Therefore, even if two chromosomes have a very similar path that is generated, their chromosome structure can differ greatly. Consequently a distance measure is required that captures the similarity between the paths and not the chromosomes.

In order to perform such a comparison, each genome needs to have a representation of the path that is encoded. Luckily this is the case because during the evaluation process when the genomes is evaluated, the path is stored in form of a grid map in M_{travel}. Thus the outcome of the evaluation provides the required information to calculate a distance between two genomes based on their encoded paths. The following distance measure D is the proposed measure that utilizes the euclidean distance to compare two paths with each other.

$$Dist(M^1, M^2) = \sum_{i=1}^{\#rows} \sum_{j=1}^{\#cols} \|M^1_{i,j} - M^2_{i,j}\| \tag{3.17}$$

$Dist$ is utilized to construct a distance matrix that D with $N \times N$ and N describes the amount of genomes that should be compared.

$$D_{i,j} = Dist(M^i, M^j) \tag{3.18}$$

As a result of the symmetry of A, the upper triangle values are used to calculate the diversity Div which is simply the mean of all distance values.

$$Div = \frac{2}{N \cdot (N+1)} \sum_{i=1}^{N} \sum_{j=i}^{N-1} D_{i,j} \tag{3.19}$$

3.3 Genome Modification

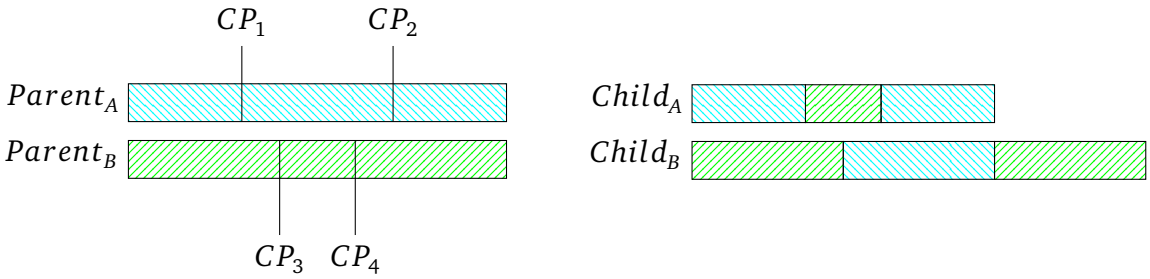
The genome modification is used to alter the genomes chromosomes. Modification operators are the essential part of a genetic algorithm because only by modifying

genomes the search space can be explored. Three different procedure, the Initialization, Crossover for combining two existing individuals and the mutations for randomly altering a small part of the chromosome are presented. Furthermore all modifications applied are analyzed and applied by the correction procedures where the *mending* applies modifications inside the gen and the *zero action removal* removes useless and dead gens from the chromosome.

3.3.1 Initialization

The initialization is required in order to start the search procedure. Goal is to generate a population with size $Init_I$ where each genome has the same chromosome length or amount of actions $Init_{AC}$. During the genome creation each genome has the start action S at the first position. Subsequently the parameter AC_α is randomly sampled from [0,360] and AC_{dist} set to a positive value between [0,50]. This is repeated for each action until all are initialized and the end action E is appended. Finally AC_{pos} is inferred by applying the method described with equation 3.8 and the resulting path is evaluated in order to assign each genome a fitness value.

3.3.2 Crossover



(a) Illustration of crosspoints CP for each parent

(b) Children after crossover

Figure 3.5: Modified crossover procedure

The recombination process, also called crossover, takes two genomes, recombines their chromosomes, and creates two new offspring. Various techniques are available for the recombination process, such as single-point, dual-point, and n-point crossover[21]. For

this work, dual-point crossover is chosen because it has the ability to exchange small pieces at arbitrary positions in the chromosome. This could be beneficial in rearranging pathway segments of the coverage path.

However, the standard procedure as illustrated in Figure 2.15b is designed to preserve the order and length of the offspring chromosomes. This is not intended in the path generation process because the GA optimization needs to be able to explore and cover the map which is only possible if new VPs or actions can be inserted and removed in the chromosome.

One possibility to accomplish that is the mutation operator. However the mutation can only insert random gens and this only to a low percentage. In order to prevent random disruptions by the mutation operator the crossover could be utilized in order to insert already, potentially good gen sequences. This would minimize the disruption and random insertions or length variations could appear more frequently, especially in the early stages of the optimization procedure when the amount of VPs is not optimal yet.

The following proposed modified Dual-Point crossover utilizes random generation of start and end indices namely cross points CP to generate offspring with variation in length.

3.3.2.1 Modified Dual-Point Crossover

The modified crossover is controlled by a possible information sharing value C_{len} that is used to calculate the max amount of gens that can be shared from genome $Parent_A$ to the offspring $Child_B$. The remaining part which includes the start and end action is passed on to $Child_A$. The opposite is true for $Parent_B$. This is illustrated in Figure 3.5. The goal of this procedure is to determine the cross points (CP_1, CP_2,CP_3,CP_4) such that the shared information between the offspring varies in length. For illustration purposes the following description focuses on the generation of CP_1 and CP_2. The process is analog for $Parent_B$.

First the absolute amount of actions $absShare$ that can be shared is determined by C_{len} and the current size of the genome G_{len}.

$$absShare = G_{len} \cdot C_{len}, \text{ where } C_{len} \in [0,1] \tag{3.20}$$

C_{len} describes the probability of how much information should be shared and therefore the max distance that is possible between CP_1 and CP_2. CP_1 selected randomly from a

uniform distribution and the uniform distribution can draw from the following interval:

$$CP_1 \in]1, G_{len} - absShare[\tag{3.21}$$

The interval is selected such that start and end action are excluded and maximal possible shared information length can still be reached. Furthermore CP_2 is equally sampled but CP_1 is used to create the sample interval:

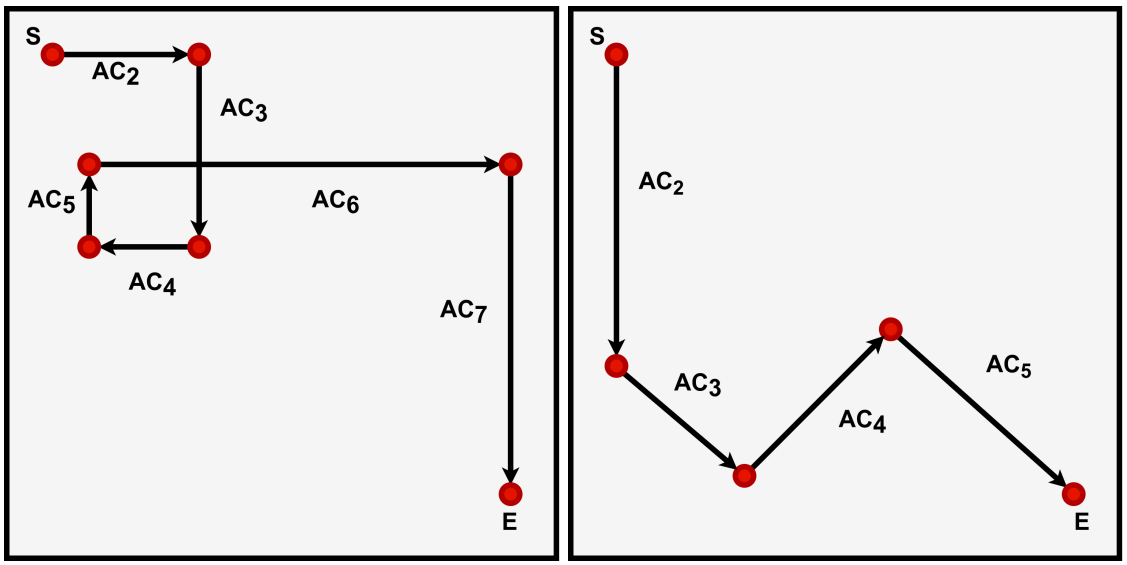
$$CP_2 \in]CP_1, CP_1 + absShare] \tag{3.22}$$

For example, given a genome G with length $G_{len} = 30$ and $C_{len} = 0.5$. Therefore $absShare = 15$ and CP_1 is sampled from the open interval $]1, 15[$. Assuming the sampling resulted in $CP_1 = 3$ then CP_2 is sampled from the half open interval $]3, 18[$. If $CP_2 = 8$ the total amount of shared information or gens shared with $Child_B$ is 5, including the gen at position CP_1 and excluding the gen at CP_2.

The information sharing value C_{len} should have an impact on the diversity in the GA population. Assuming a high value with $C_{len} = 0.8$ is chosen that the randomly selected parts and differences in the actual shared information can differ greatly between $Parent_A$ and $Parent_B$ and therefore introduce more diversity. If on the contrary a small value is selected $C_{len} = 0.2$ only few variations of length and amount of gens that can be shared are possible and therefore the resulting offspring does only differ in two or three gens which should decrease diversity. The most reasonable interval would be in range $[0.1, 0.5]$ because sharing too little or too much information would result in copying the either only one gen or almost the complete genome and thus neglecting the dual point crossover strategy where one tries to share equal amount of information between the offspring. Overall the proposed procedure utilizes ranges of possible positions and a maximum length for the shared information to determine the range of gens that are shared with the offspring.

3.3.2.2 Locality Preservation

The previous section covered the first part of the crossover procedure that describes the selection of the respective genomes that are selected for sharing with the offsprings. Despite varying the sizes of the shared it is also possible to decide how the shared information should be interpreted when it comes to the path generation. The two different integration strategies for genetic material are possible because two different kinds of information, the VPs and action are available. Information sharing based on the VPs is described as *locality preserving* crossover and the action based *locality changing* crossover.



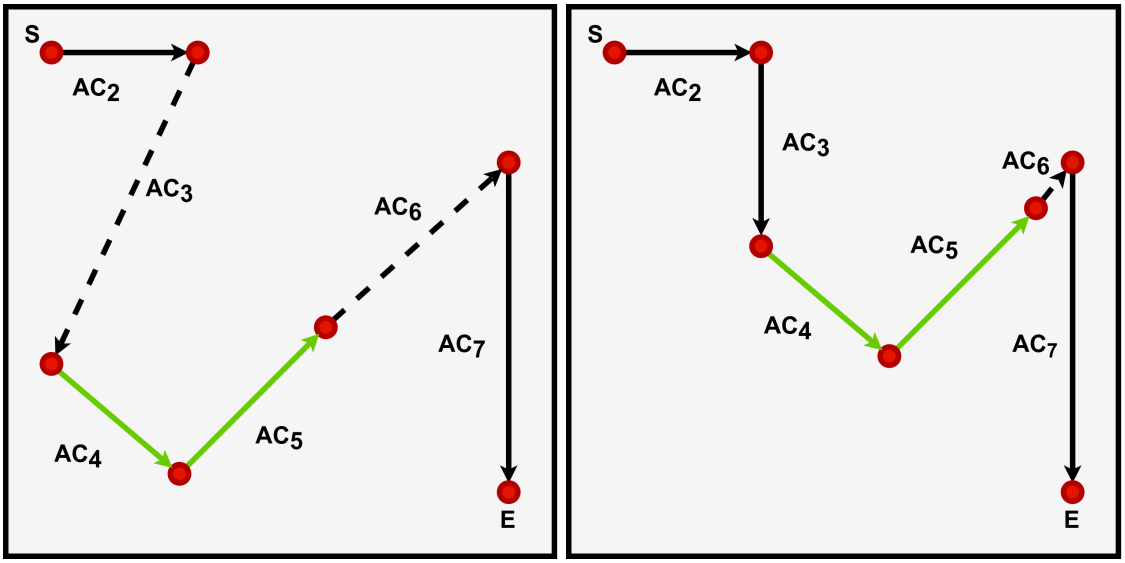
(a) Path generated by G_1 (b) Path generated by G_2

Figure 3.6: Parents selected for crossover

Easiest to explain both strategies is with an example. Assuming two genomes G_1 with $G_{1,len} = 9$ and G_2 with $G_{2,len} = 6$. The path generated by G_1 is visualized in Figure 3.6a and G_2 in 3.6b where red dots are the VPs , the black arrows the connecting actions, S the start and E the end positions. Furthermore it is desired to create the offspring from G_1 with respective cross points $CP_1 = 3, CP_2 = 7$ and $CP_3 = 3, CP_4 = 5$ for G_2. The generated offspring is based on G_1 therefore the start and end part are the same whereas gens in between CP_1 and CP_2 are exchanged with gens between CP_3 and CP_4. The actual sharing procedure is illustrated in Figure 3.6.

In the *locality preserving* crossover illustrated in Figure 3.7a two action from G_2, AC_3 and AC_4 are combined with the offspring of G_1. Preserving the location basically means that the actions before and after the newly inserted ones are adapted such that the positions remain the same. The consequence of this is that AC_3 and AC_6 are adapted in order to integrate the gens correctly in the chromosome. Black dotted arrows in the illustrations are marked as altered actions from the parent, where green lines illustrate the new action sequence.

Changing the locality as shown in Figure 3.7b results into recalculating all positions for the newly inserted action and consecutively connecting the new position at the end of the inserted sequence at AC_5 with the rest of the path. This is accomplished by modifying AC_6.



(a) Locality preserved, VP based (b) Locality changed, action based

Figure 3.7: Offspring generation strategy

Both procedures are presented for one side of the crossover (G_1). Same is applied to the offspring of G_2. Furthermore this opens the possibility of expanding the search space of the GA during the crossover by introducing the two variants with and without locality preservation. The resulting GA procedure allows to select between tree different crossover strategies. First preserving location, second, changing it and third using both where the third and combined strategy generate four instead of two offspring individuals. This is expected to yield better results in the optimization because more individuals means more samples of the search space and therefore better chances in getting good individuals.

3.3.3 Mutation

After crossover the offspring is exposed to the mutation operator. Mutations only affect small parts of the chromosome, benefit the exploration process and increase the diversity when applied. All mutations follow the same principle. First it needs to be decided if the mutation is applied to a genome. This is controlled by the mutation probability P_{mA}. Subsequently, if the mutation is applied a gen in the chromosome is randomly selected. In the following procedures tree types of mutations are available

that randomly alter the selected gen. The first mutation randomly scales the distance $randScale(gen)$ of one action or restores the action if distance is zero:

$$randScale(AC) = \begin{cases} AC_{dist} > 0, & AC_{dist} \cdot r, \text{ where } r \in [0.5, 1.5] \\ AC_{dist} = 0, & \frac{1}{G_{len}} \sum_{i=1}^{G_{len}} AC_{i,dist} \end{cases} \tag{3.23}$$

r is randomly sampled form the given interval and promotes equally shrinking and increasing of action distance. The second mutation randomly modifies the angle:

$$randAngleOffset(AC) = AC_{\alpha} + r, \text{ where } r \in [0,360] \tag{3.24}$$

AC_{α} is given in degree. r enables free rotation in all directions.

The last action is contradicting to the mutation philosophy because the complete chromosome is changed by $randInit(G)$. All actions are reinitialized including their positions, angle and distance. This type of mutations is strongly disruptive and negatively influences the performance if applied too frequently. Therefore it has its own probability P_{mI}.

Despite the disruptive behaviour, such a mutation is beneficial of the population is trapped in a local optima. Randomly seeding new individuals can therfore introduce more diverity and benefit the overall search process.

3.3.4 Correction Procedure

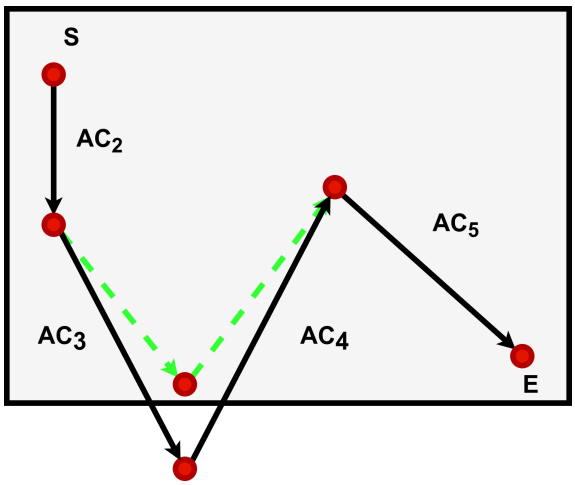


Figure 3.8: Out of bounds correction during evaluation

Correction procedures are applied in the context of a chromosome and population. During the optimization two problematic cases needs to be corrected in order to prevent erroneous or computationally expensive behavior.

When a genome is initialized or modified by the crossover or mutation it needs to be validated that all VPs are still inside the map boundaries. The correction for this happens during the evaluation when the robot executes each action in the genome. If a VP is outside the map boundary, the closest position to the map in the map is located and used as new start position for the next action. In fact even the consecutive action is altered to match the previously stored end position. This process is illustrated in Figure 3.8. First angle and distance of AC_3 is is adapted to steer to the new point in the map and subsequently AC_4 is adapted in all its parameters to preserve the start position of AC_5.

The second problem concerns so called zero actions. When an action has the distance factor of zero it is considered to be a zero action. Such actions do not contribute to the fitness in a positive way and only generate coverage if distance is injected by some mutation procedure. Even worse if two consecutive actions have the same direction there is no effect on the fitness at all. Furthermore such action impact the computational performance in a negative way because they tend to increase the length of the overall chromosome. This is time consuming for the evaluation process, hence such actions are partially removed. There is one condition for this, a zero action is only removed if it is placed between two other zero actions. The reason for this is that the general concept of zero actions is noting undesirable because those actions can be utilized later in the process. However, with the zero action removal a “healthy” amount still remains in the chromosome but overall length is decreased such that the computational load is reduced.

The last correction procedure deals with so called dead genomes. Those are genomes that only consist of zero actions, have no coverage at all but a very high time value. However, those individuals only contribute in a negative way by sharing zero actions. Therefore as soon as a dead genome is detected after evaluation, it is completely reinitialized. This prevents the population from converging towards such dead individuals which was a critical problem in preliminary tests.

3.4 Genome Selection

This section presents all used methods to select genomes based on their fitness value. Four methods are chosen including the Roulette-Wheel selection in two different

variants as well as the Tournament selection. Furthermore a custom version of the elite selection similar to [34] is proposed.

3.4.1 Elite Selection

The elitist selection scenario is partially based on the niche procedure where parents should compete against their offspring[34]. In order to achieve such a behaviour a two staged selection process is developed that first groups all individuals randomly into a so called family. A family is simply a population consisting of two individuals. Families are generated by drawing without replacement from the population until no individual is left. This process is guided by the $Init_I$ parameter where the amount of families is stated by $\#families = Init_I/2$. If $Init_I$ is uneven, the remaining individual will be ignored in the family generation process.

After the generation, crossover is applied to each family depending on the crossover probability P_c. Thereby is a family excluded when crossover is not applied such that they are further ignored by the upcoming elite selection. For all other families offspring is generated such that at least four individuals are in each family. Subsequently the children are exposed to mutation and fitness is calculated whereas the parents remain unchanged. Finally the elite selection is applied where only the two best individuals inside a family are selected and passed on the next generation.

The overall benefit of this method is that it does not require any type of meta parameter. Furthermore all individuals get a chance in reproduction and none is left out. However this comes with the drawback of high selection pressure which may result in dominant individuals quickly taking over the population.

3.4.2 Tournament Selection

The Tournament Selection behaves exactly as described in Section 2.2.1.1 where TS dictates how many individuals participate the tournament process. Despite the tournament size, it is also important to determine the amount of offspring that is generated by the selection. This is implicitly given by the amount of individuals that are determined by the tournament process, the $select$ parameter. Thereby setting $select = 1$ selects two individuals that generate two offspring individuals.

One major benefit is the adaptability of the selection pressure through SP. This is investigated in more detail in the Section 3.5.2 where the selection pressure is dynamically adapted in order to accelerate the optimization process.

3.4.3 Roulette-Wheel Selection

Two different versions of the Roulette-Wheel Selection (RWS) are used, the proportionate RWS and the ranked RWS. The PRWS is described in Section 2.2.1.2 and only exposes the $select$ parameter that behaves similar to the tournament selection.

The same holds true for the RRWS that is described in Section 2.2.1.3. Despite the $select$ parameter, the selection pressure is adaptable by changing the ranking that calculates the selection probability. This is achieved by the $selPressure$.

3.5 Genome Optimization

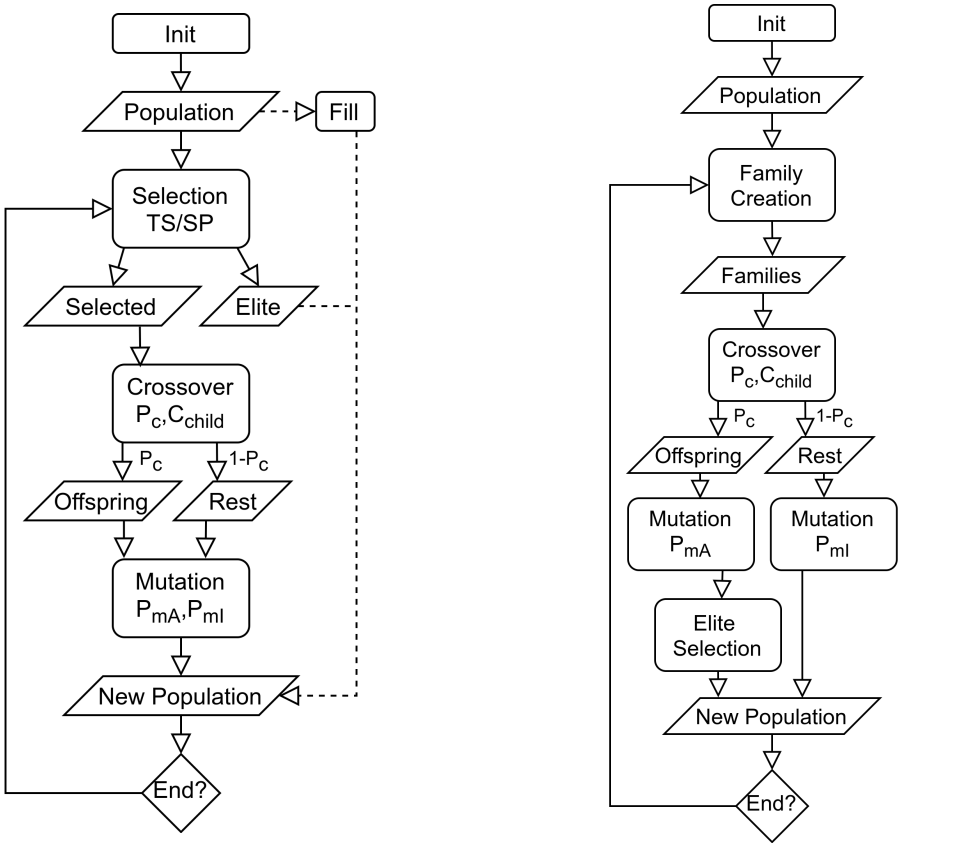
The previous sections presented all necessary components and operators to build a GA optimization. This section brings all components together and explains the general setup of the GA architecture. Subsequently, two parameter control methods are presented as well as an experiment that should elaborate on the multi-robot capabilities.

3.5.1 Architecture Design

This section describes the general architecture in its two variations as illustrated in Figure 3.9. Main focus lies on the population behavior and how the population is divided and modified throughout one optimization step.

The population modification directly depends on the applied selection strategy. Those strategies and their parameters are described in Section 3.4 where RRWS, PRWS and Tournament selection ($Turn$) behave similar in light of how individuals are selected from the population. However, the custom $Elite$ selection does not because all individuals are competing against their own small population. This needs to be taken into consideration when designing the overall algorithm. Figure 3.9 presents the two different architectures where the first one 3.9a is used when RWS or $Turn$ is employed and 3.9b for the $Elite$ procedure.

Starting with illustration 3.9a the population is initialized as described in Section 3.3.1 and passed on to the selection operator. The tournament size TS or the selection pressure SP are utilized to select individuals based on their fitness value which results in the “Selected” population. Furthermore, the best individuals with highest fitness are saved in the “Elite” population. The size is determined by the $keep$ parameter. It is important to notice that individuals in the “Elite” population can also be part of



(a) Locality preserved, VP based

(b) Locality changed, action based

Figure 3.9: Offspring generation strategy

the "Selected" population. Only the "Selected" individuals are passed to the crossover procedure where the crossover probability P_c states on which individuals to apply the crossover which results in "Offspring" and on which not. All individuals not part of the crossover will remain unchanged and are stored in the "Rest" population. Furthermore the C_{child} parameter stated what strategy is utilized when generating the offspring (see Section 3.3.2.1). $C_{child} = 0$ generates two offspring individuals where locality is changed, $C_{child} = 1$ where locality is preserved. A combination of both which creates four offspring individuals from two given parents is applied when $C_{child} = 2$.

On both the "Offspring" and "Rest" mutation operators are applied. There are two different types of mutations, one to reinitialize complete chromosomes where the probability is P_{mI} and the action modifications that are less destructive and only alter one gen. This probability is given with P_{mA}. The resulting individuals are stored in the "New Population" If the population is of size less that Pop_{min} random individuals from "Population" of the previous generation are inserted to the "New Population".

Finally the end condition is checked which can either be a convergence criteria or a fixed amount of iterations $Iter_{max}$. If the end condition is not fulfilled the process is repeated with the "New Population".

The second architecture illustrated in Figure 3.9b shows some modifications, especially after the initialization when the "Population" is split into families. Each family consists of two individuals. After the crossover the "Offspring" contains families of four or six individuals depending on the C_{child} parameter. The "Rest" is not organized in families anymore and only there the P_{mI} mutation is applied. The reason for this is that newly initialized individuals are almost always worse that already optimized ones. Thus completely re-initializing an offspring individual would effectively destroy it without any benefit. However, applying this mutation to the "Rest" gives the random individual a chance to share genetic material in the next round of crossover and therefore insert diversity to the process. Furthermore the P_{mA} is only applied to the offspring individuals, not the parents in order to preserve the best individuals from the previous generation. Finally the elite selection is applied to the families in the "Offspring" population such that the "New population" consists of the best overall individuals and potentially re-initialized once.

3.5.2 Parameter Variation

General principles and strategies on how variation of the parameter setting works is explained in Section 2.2.3.1. This section utilizes the two presented approaches to propose two different parameter variation strategies.

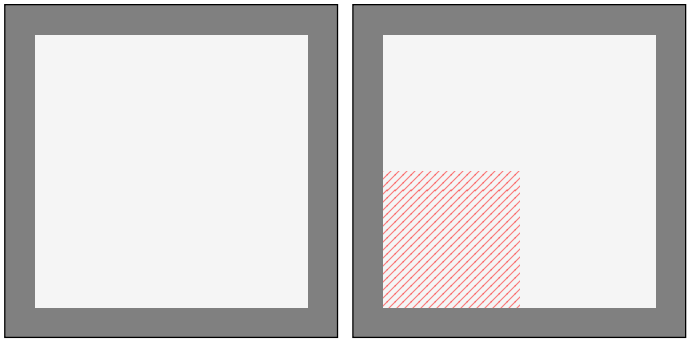
Deterministic Parameter can be used to steadily increase the selection pressure such that at the beginning of the optimization the selection pressure is very low in order to promote diversity and prevent premature covergence. While the optimization progresses the selection pressure increases which results in faster overall convergence. In order to accomplish such behavior, equation 2.3 is adapted as follows:

$$SP_i = \left[\left(\frac{Iter_i}{Iter_{max}} \cdot (SP_{max} - SP_{min}) \right) + SP_{min} \right] \tag{3.25}$$

Where SP states the selection pressure and can either be applied to the RRWS or used as tournament size TS where the SP_{min} and SP_{max} values needs to be adapted accordingly.

Adaptive Parameter utilize the diversity to steer the search direction towards global or local search in order to promote diversity or better fitness values. This concept is copied from [28] and except the diversity calculation the exact same equations are utilized to calculate the crossover, with equation 2.3 and mutation probability 2.6. SPD is substituted with diversity Div calculated in equation 3.19. Furthermore SPD_{max} is set to the maximum value of the distance matrix D used for the diversity calculation in equation 3.18.

3.5.3 Changing Environment Conditions



(a) (b)

Figure 3.10: Two maps are shown. (a) is used to perform the prior coverage path optimization. Subsequently a region is marked as covered (b) and the optimization continues

Up until now the coverage path generation is designed to optimize a path for a single robot. In order to enable multi-robot applications, the optimization procedure needs to adapt the already generated path according to the incoming information. Imagine the following scenario: Multiple robots are positioned in their respective charging station and receive the command to start cleaning a given area that is also transmitted. In the first step each robot optimizes its own path for the complete region and generates a near optimal coverage path. Next, the robots start cleaning and begin to negotiate with each other in order to determine which regions should be covered by which robot and what regions are already cleaned. Furthermore, obstacles on the map that are not known in advance should be included in the optimization. The coverage paths of each robot have to be adapted until no further improvement is available or changing environment conditions are detected.

In order to test if the GA procedure can react sufficiently to changes inside the environment the following test is conducted. After the coverage path is generated, a part of the transmitted map is marked as already covered region. This is interpreted by the robot as a reduction in the overall area that can be covered, thus $A_{[m^2]}$ is reduced accordingly and every path segment that intersects with the covered region is automatically interpreted as already covered area such that this area is not desirable to be visited. An illustration of that principle is shown in Figure 3.10. First the path generation is performed on the basic map that is initially transmitted, containing information about the environment (3.10a). Subsequently, the negotiation between the robots begins and as a result the red marked region as shown in Figure 3.10b should be avoided.

A more detailed description on the map and also different map types as well as the test results of this process is described in the next chapter.

4 Designspace Exploration

The previous chapter presented, the genome representation with the modification techniques as well as different selection methods. Ultimately all components where connected to build the final genetic algorithm architecture as well as a few variations in parameter control and architectural wise population behavior. Goal of this chapter is to first get an overview of the complete design space and thus present all possible parameter settings. Subsequently, with the help of prior knowledge and empirical tests a subset of the parameter space is selected and used to perform a gridsearch on the proposed GA architecture. Finally a subset of the best individuals is selected for the retrain procedure described in Section 3.5.3.

Table 4.1: All parameter that can be varied are given in the table with their respective range

Parameter	Value Range	Comment
M_{type}	$\{1, 2\}$	
Pop_{min}	$[1, \infty]$	Elite
$Init_{AC}$	$[0, \infty]$	
$Init_I$	$[Pop_{min}, \infty]$	
keep	$[0, Pop_{min}]$	Turn, RRWS, PRWS
select	$[1, Pop_{min}]$	Turn, RRWS, PRWS
TS	$[1, Pop_{min}]$	Turn
SP	[1, 2]	RRWS
P_c	[0, 1]	
C_{len}	[0, 1]	
C_{child}	$\{0, 1, 2\}$	
P_{mA}	[0, 1]	
P_{mI}	[0, 1]	

Table 4.2: All fixed parameters and environment conditions

Parameter	Value
M_{res}	$0.2[m]$
M_{width}	$11[m]$
M_{height}	$11[m]$
R_{speed}	$0.2[\frac{m}{s}]$
R_{width}	$0.6[m]$
$Iter_{max}$	2000
$Init_{AC}$	50
$Init_{I}$	100/1000
Pop_{min}	100
P_c	0.8

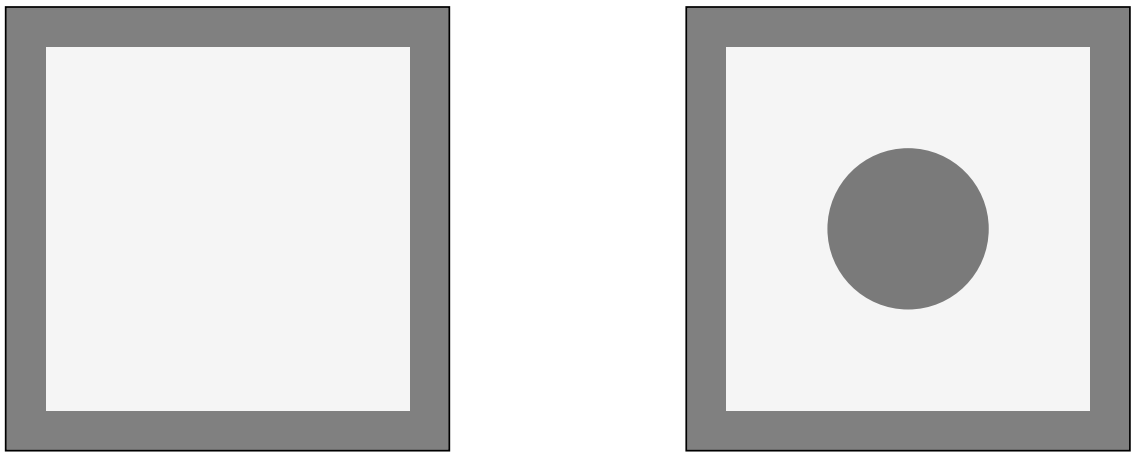
4.1 Parameter Search

The parameter search is composed of three parts. First all environment conditions in Table 4.2 are explained. Furthermore the possible parameter ranges listed in Table 4.1 are broken down to more reasonable samples that can be used to perform a grid search. The resulting values used by the search are listed in Table 4.3. Subsequently, the results of the grid search are presented in Section 4.1.2. A subset of the best performing parameter settings is used to test the behavior when environment conditions change as suggested in 3.5.3. Finally those results are presented in Section 4.1.3.

4.1.1 Preliminary Parameter Setting

Table 4.2 describes the general parameter setting for the GA architecture. However, performing a grid search on those parameters would be an impossible task because some are not properly ranged in order to test them. Therefore, this sections concerns with a predetermination of those parameters in order to reduce the overall design space. First, all environment components concerning the map M and robot R are determined. Subsequently, reasonable samples for the parameters presented in Table 4.1 are determined in order to perform a grid search.

During the search two versions of the map M are used which are referred to with $M_{type} \in \{1, 2\}$. The two maps are visualized in Figure4.1. Walls and inaccessible area is marked with gray and bright one is the area that has to be covered. Both maps have the same dimensions with $M_{width} = 11$, $M_{height} = 11$ but in the second map ($M_{type} = 2$) a



(a) $M_{type} = 1$ (b) $M_{type} = 2$

Figure 4.1: Illustration of different map types used for the path generation process.

circular shaped obstacle is place in the center. The robot R that is utilized to evaluate the genome encoding has the following properties: $R_{speed} = 0.2$, $R_{width} = 0.6$, $R_{rot} = 6.28$. The rotation speed R_{rot} corresponds to 60 rounds per minute and is altered during the search to emulate a path generation with rotation penalty and without by setting $R_{rot} = 0$.

Choosing proper ranges for the GA parameter setting is accomplished by empirical knowledge and prior tests as well as using values that are already accepted in other works. The duration of one GA iteration can take from 20ms up to 1000ms. This depends on many factors but ultimately the maximum amount of iterations $Iter_{max}$ is set to 2000 with a time limit of 20 minutes for the duration of the optimization. Furthermore a maximum chromosome length is set to 300.

For the initialization three parameters need to be adapted to generate an initial population. The minimal population size $Pop_{min} = 100$, amount of initial individuals exposed to the GA in the first iteration $Init_I = 1000$ and the amount of actions each genome should feature $Init_{AC} = 50$. Those parameters where empirically determined by previous tests and yielded good results in prior tests. Note that the $Init_I$ parameter is overwritten with Pop_{min} when using the $Elite$ selection because each individual is first processed by crossover and mutation in order to create a family.

Selection procedures, except for $Elite$ are adjusted with the amount of pairs of individuals that are selected by the procedure $select$ and best individuals that should be part of the new population $keep$. $select$ is set to {10,20} and $keep$ to {1,10}. For the

Table 4.3: All parameter ranges used by the grid search

Parameter	Elite	Turn	PRWS	RRWS
$f_{obstacle}$	f_1, f_2	f_1, f_2	f_1, f_2	f_1, f_2
R_{rot}	0, 6.28	0, 6.28	0, 6.28	0, 6.28
M_{type}	1, 2	1, 2	1, 2	1, 2
$keep$	0	1, 10	1, 10	1, 10
$select$	0	10, 20	10, 20	10, 20
TS	0	3, 7	0	0
SP	0	0	0	1.4, 2
C_{len}	0.2, 0.4, 0.6, 0.8	0.2, 0.4, 0.6, 0.8	0.2, 0.4, 0.6, 0.8	0.2, 0.4, 0.6, 0.8
C_{child}	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2
P_{mA}	0.0, 0.1, 0.01	0.0, 0.1, 0.01	0.0, 0.1, 0.01	0.0, 0.1, 0.01
P_{mI}	0.0, 0.1, 0.01	0.0, 0.1, 0.01	0.0, 0.1, 0.01	0.0, 0.1, 0.01

$Turn$ and $RRWS$ the tournament size TS is set to {3, 7} and selection pressure value SP to {1.4, 2} respectively.

The general crossover probability P_c is fixed at 0.8 because typical crossover probabilities range from 0.6 to 0.95[21] and 0.8 showed promising results in prior tests. For the proposed Dual-Point crossover, the possible range for C_{len} was sampled in 0.2 steps such that $C_{len} \in \{0.2, 0.4, 0.6\ 0.8\}$. Furthermore all variations of the child selector C_{child} are utilized: $C_{child} \in \{0, 1, 2\}$. Where $C_{child} = 0$ means locality is not preserved, $C_{child} = 1$ preserves it and $C_{child} = 2$ utilizes both.

Two types of mutation probability are available, the action modification P_{mA} and re-initialization probability P_{mI}. Both are set to very low values because of their destructive nature[21] such that $P_{mI}, P_{mA} \in \{0, 0.01, 0.1\}$.

A complete overview of all parameter ranges is given in Table 4.3. Furthermore, Table 4.2 features all parameters that stay fixed during the search process.

4.1.2 Parameter Grid Search

During the grid search all parameters listed in Table 4.3 are utilized to generate different parameter settings. Each parameter setting is used to perform a GA optimization that is former described as run. Table 4.4 gives a concise overview of how many runs are performed for each selection procedure as well as their respective average, minimal and maximal duration time.

Table 4.4: Statistics shows overall amount of tests that where conducted for each selection, in addition the min, max and average duration for one run in the specific selection is presented

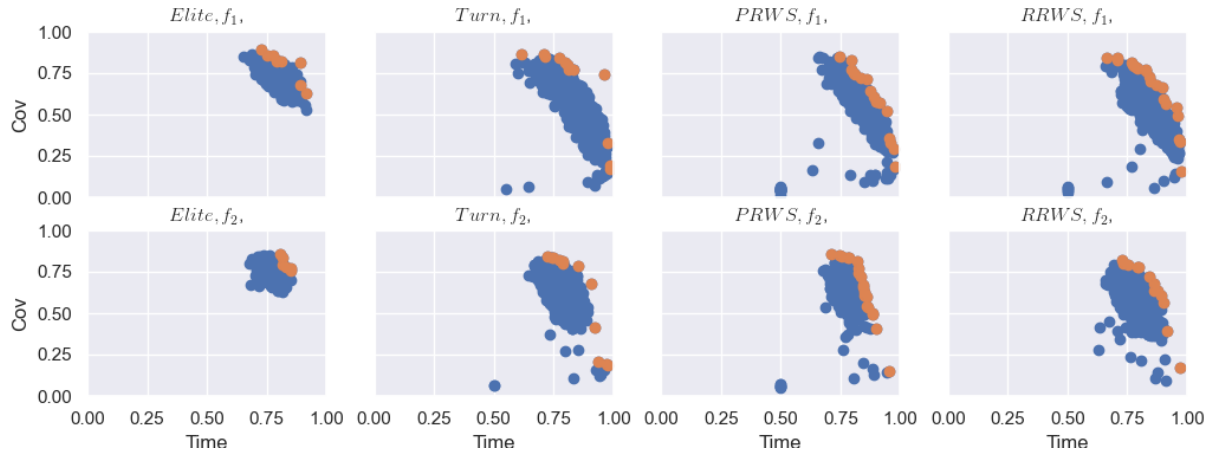
Selection	$Elite$	$Turn$	$PRWS$	$RRWS$
$\#Runs$	5201	20779	10392	17318
Avg Duration	269s	123s	106s	143s
Max Duration	846s	715s	497s	620s
Min Duration	26s	21s	17s	25s

Two major aspects play a role when evaluating the behavior of GA operators, M_{type} and R_{rot}. In order to keep the amount of visual illustrations to a reasonable amount, only results for $M_{type} = 1$ and $R_{rot} = 0$ are described. However, a complete collection of the results is presented in Chapter A.

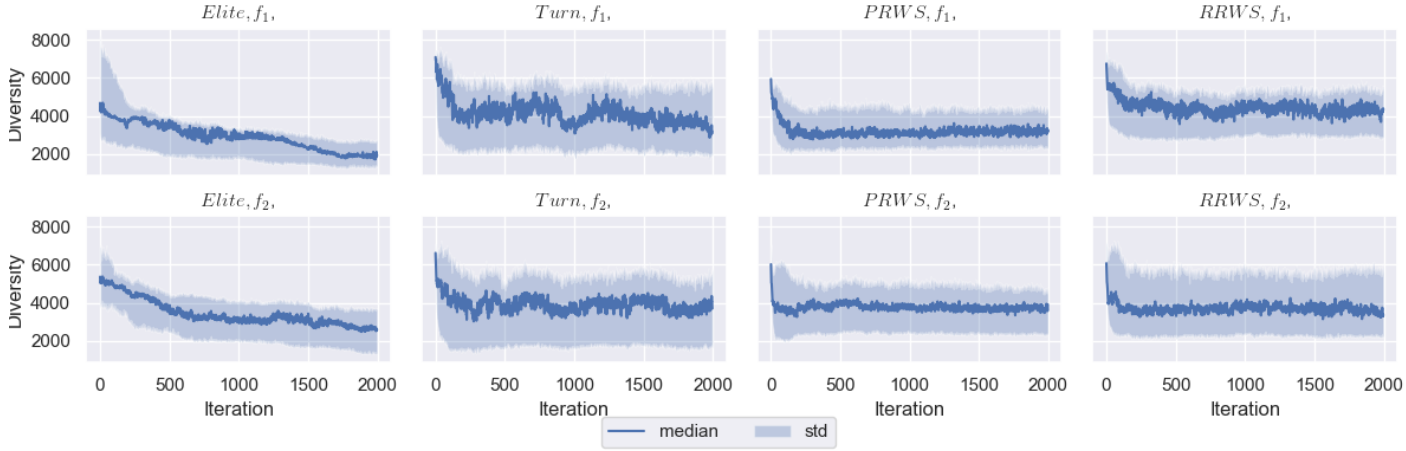
Each run optimizes a population where the best individual in terms of fitness is stored. In order to compare individuals across runs the cov_{final} and t_{final} are utilized. This has the benefit, that the comparison between different fitness functions is possible.

The following visualizations alway feature the same principle. First a scatter plot show all individuals, i.g. one point represents the best individual out of the respective run. Additionally the individuals with highest cov_{final} and t_{final} are plotted in a different color. This corresponds to the Pareto front. Secondly, the runs generating the individuals in the Pareto front are used to generate the diversity plots.

The first illustration in Figure4.2 visualizes the different selection procedures with their respective fitness function.



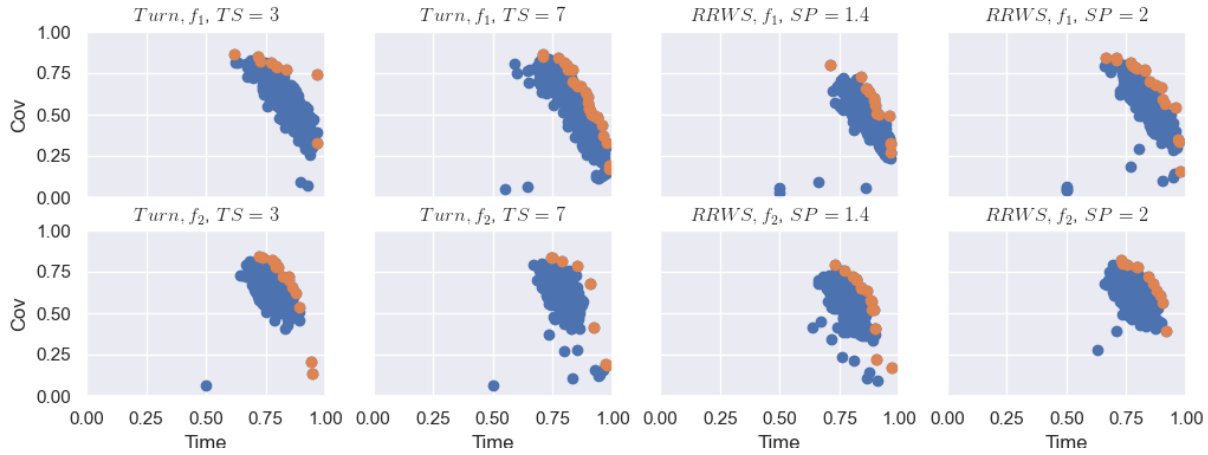
(a) Pareto front



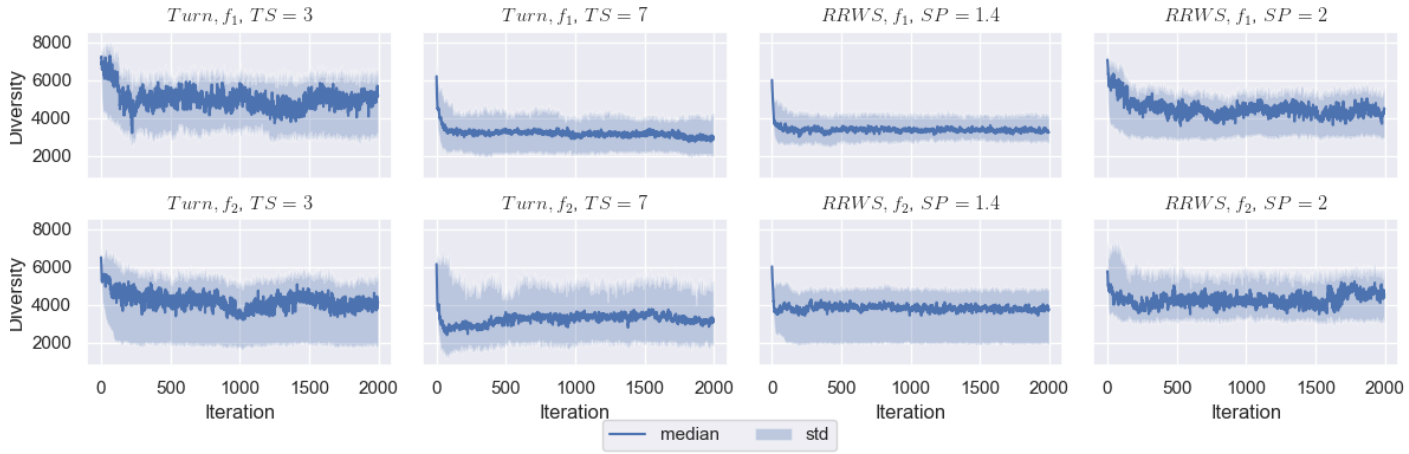
(b) Diversity

Figure 4.2: Pareto front and diversity for selection procedures and fitness functions

The different kinds of selection pressure with the *Turn* and *RRWS* are visualized the same way in Figure4.3.



(a) Pareto front

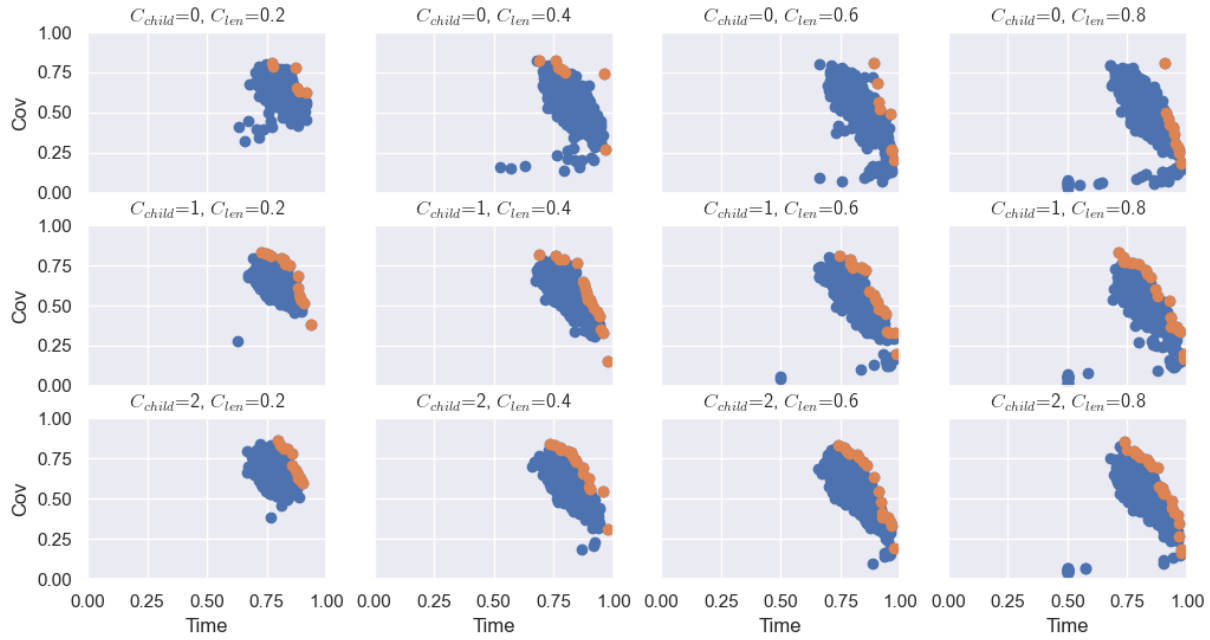


(b) Diversity

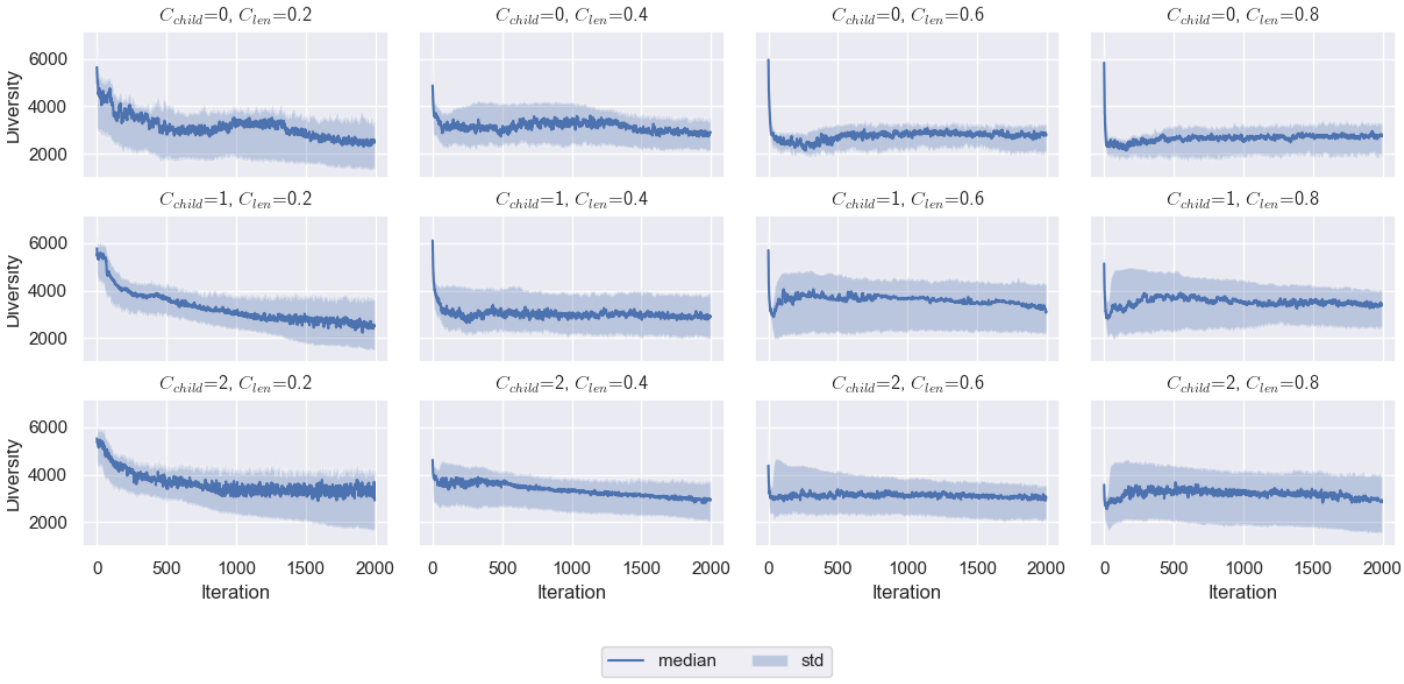
Figure 4.3: Pareto front and diversity for selection pressure and fitness functions

The Crossover is evaluated in terms of C_{len} and C_{child}. All combinations are listed in Figure4.4.

Finally, the mutation operator is evaluated according to the effect of P_{mI} and P_{mA} which is illustrated in Figure 4.5

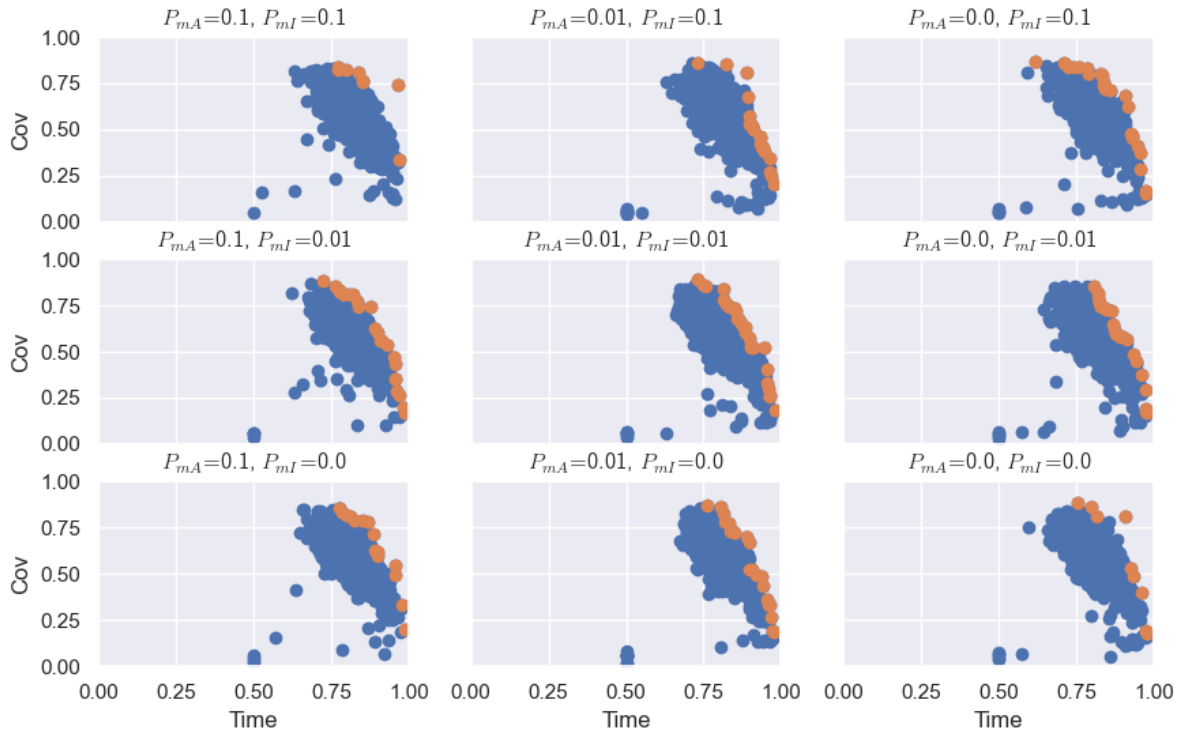


(a) Pareto front

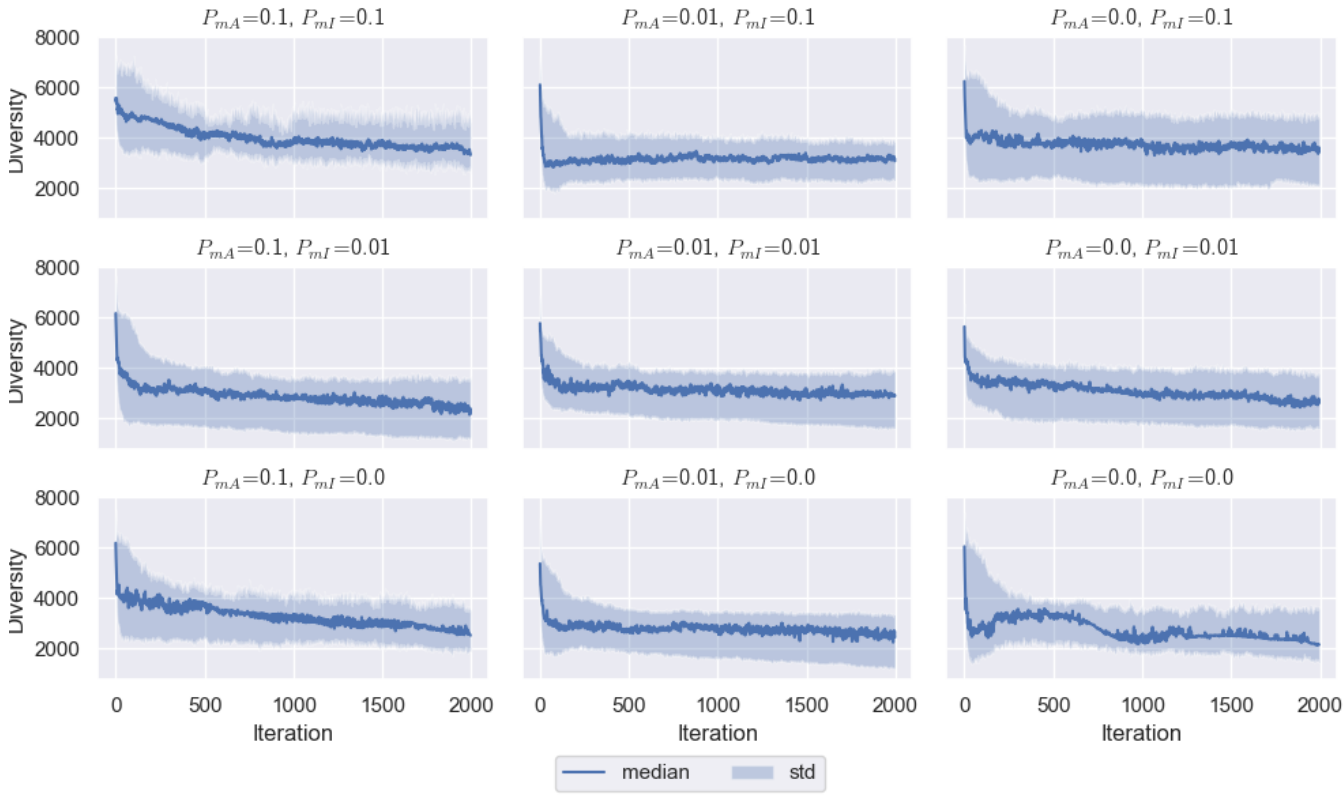


(b) Diversity

Figure 4.4: Pareto front and diversity for C_{len} and C_{child}



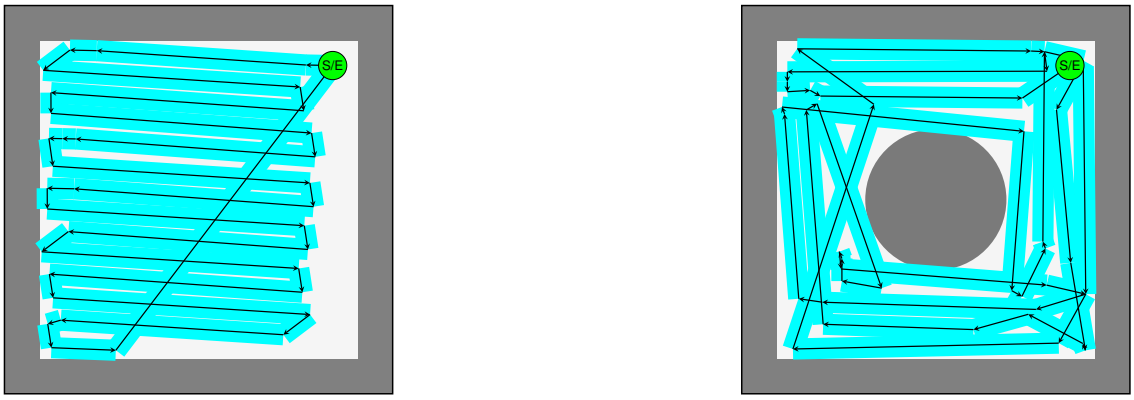
(a) Pareto front



(b) Diversity

Figure 4.5: Pareto front and diversity for mutation probabilities P_{mI} and P_{mA}

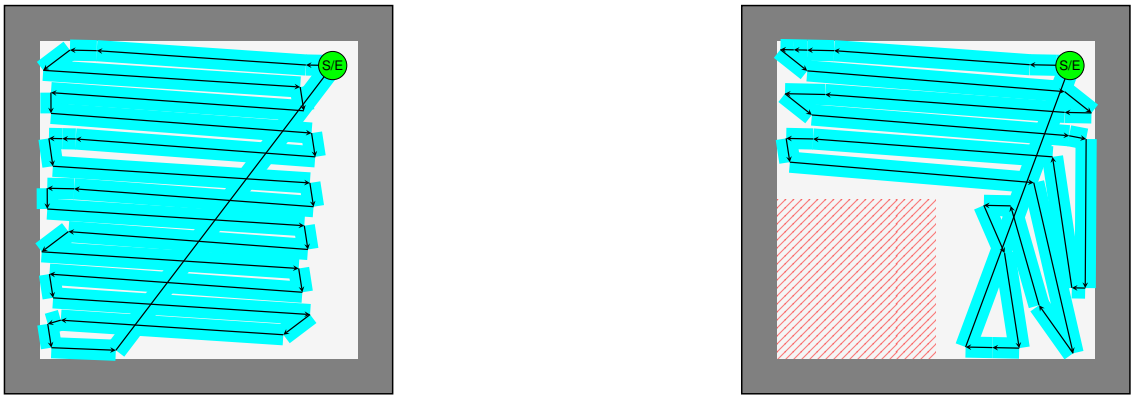
In preparation for the next section, the best individuals for each selection strategy, fitness function, R_{rot} and map type are selected. A complete list of the best individuals for $M_{type} = 1$ is presented in Table A.1 and for $M_{type} = 2$ in TableA.2. Furthermore all generated paths for each configuration are visualized in FigureA.17 and A.18. Two examples, one for each map type are illustrated in Figure4.6.



(a) $M_{type} = 1$ (b) $M_{type} = 2$

Figure 4.6: Illustration of the resulting coverage path

4.1.3 Changing Environment Conditions



(a) (b)

Figure 4.7: Effect of different environment conditions

The best 32 runs that selected by fitness, map type, scenario and rotation factor are used for the retrain procedure. End results are shown in Table A.3 and A.4 for their respective map type. Again, all generated coverage paths are also referenced there.

More interesting is the behavior of the run itself when the environment condition changes. The path before and afterwards for run (j) in Table A.3 is illustrated in Figure 4.7.

Furthermore the internal performance values during the run are presented in Figure4.8

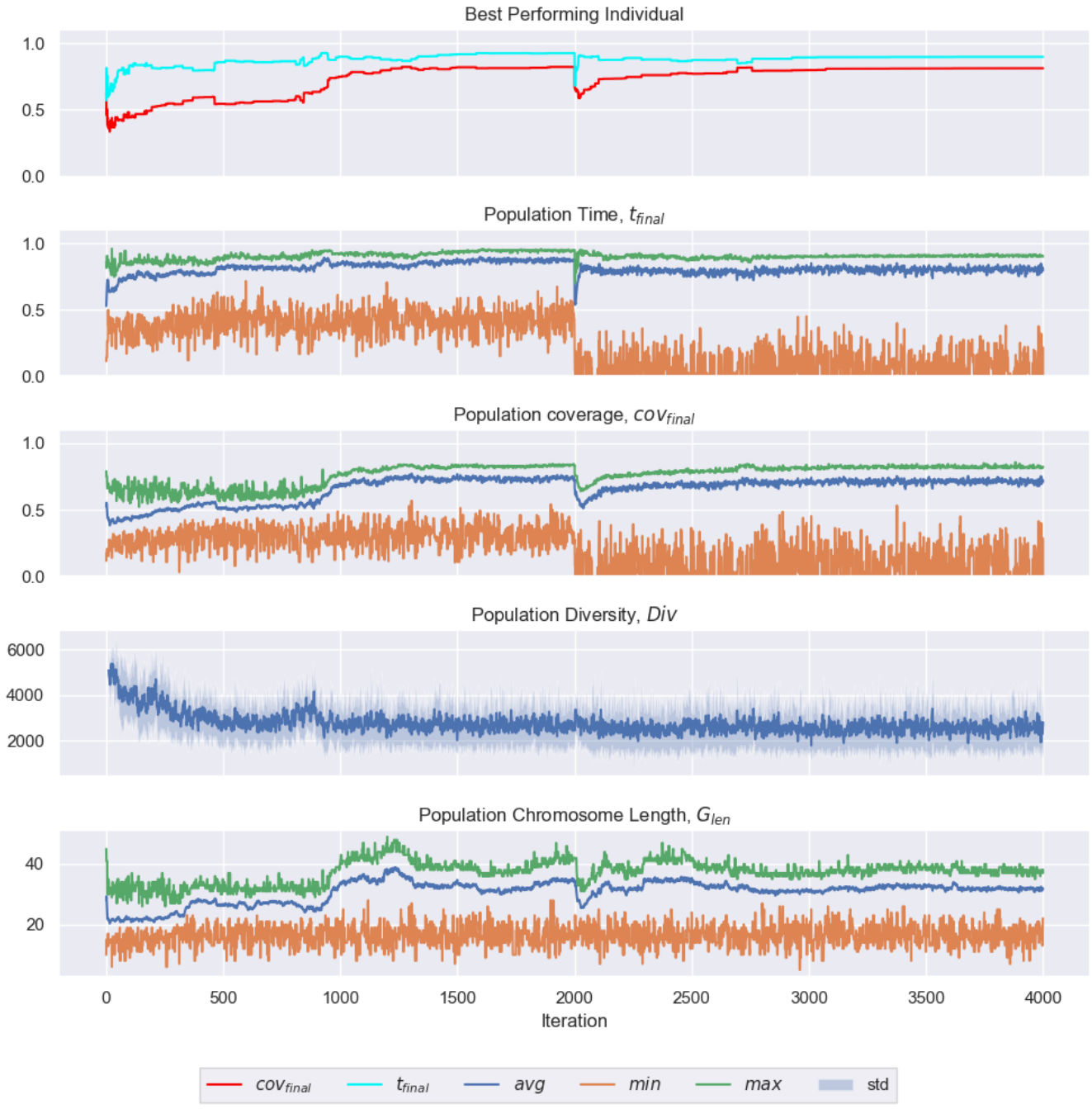


Figure 4.8: Run first optimized with normal map. After 2000 iteration environment conditions change such that the robot is supposed to avoid the newly covered region

5 Discussion

The previous chapter presented a subset of the overall results generated by the grid search procedure. This includes the distribution of the best individuals and their respective Pareto front which are the topic of interest of this chapter.

First the behavior of the different operators is investigated by inspecting the distribution of the Pareto optima as well as the respective diversity. Subsequently, the path generation itself and the reaction to changing environment conditions is analyzed. Finally, ideas for future work are presented.

5.1 GA Operators

In Figure 4.2 the different selection procedures with their respective fitness functions are visualized. Considering f_1, except for the $Elite$ selection, all individuals are distributed along a line or at least slightly curved shaped distribution. Interestingly, individuals never seem to reach the full coverage value which means that there is no optimal coverage path generated at all. However, the time value is frequently reached to a full extend but almost always with a coverage value below 0.5. This can be correlated with the calculation of the time factor because when there is less region covered it is also easier to get less crossing path segments. Furthermore, this seems to have a negative impact on the overall population behavior and traps the individual in a local optima where the coverage value cannot be improved because the time is already to high. This on the other hand seems to be compensated when using f_2 where the population tends to be more distributed towards the diagonal between time and coverage. Furthermore, $Elite$ generates individuals that are in a small distribution within a reasonable range for f_1 and almost centered in the upper right corner for f_2. Additionally, no outliers are generated in the $Elite$ selection which is cause by the strong selection scheme. This however is treated for with strong loss of diversity as shown in Figure 4.3b. In fact $Elite$ looses the most diversity caused by its restrictive offspring generation policy. The loss of diversity does not necessary need to be bad. Actually the path generation samples presented in TableA.1 suggest, that the procedure simply converges to good solution

which is supported by the small distribution of individuals. However, neither optimal coverage nor time is reached thus suggesting that there is still room for improvement. The other three procedure behave similar in terms of distribution of individuals as well as diversity with the exception of slightly higher variation in $Turn$ diversity. Otherwise they are roughly on the same level.

The comparison between selection pressure of $Turn$ and $RRWS$ is illustrated in Figure 4.3. Expected behavior is that the diversity reduces with an increase in selection pressure. This can be seen for the $Turn$ selection where the diversity is lower when TS is increased. An immediate effect on the population distribution is not observable except for higher TS and F_2 where more outliers are generated. This indicates that the high pressure causes premature convergence in some cases. The opposite seems to be the case when increasing SP when f_2 is applied. Less outliers are observed. However a decrease in diversity is not visible, which indicates that $RRWS$ is in fact good in preserving diversity by simultaneously applying higher pressure.

Next is the crossover presented in Figure4.4 where different combinations between the child selector C_{child} and crossover length C_{len} are tested. The initial assumption is that small length of shared information reduces the overall diversity because only small parts of information are shared which generates similar offspring to the parents. Observing the distribution of individuals supports this assumption that small C_{len} introduces fewer variations because such individuals are less spread than those with high C_{len} values. The child selector does not have an obvious effect on the spread of the population distribution but an overall decrease in outliers is observed supporting the assumption that utilizing both child types lead to more robust offspring. It is hard to draw any conclusion from the diversity except that $C_{len} = 0.2$ causes diversity loss whereas the diversity of longer information length remain steady.

The different mutation parameters presented in Figure 4.5 compare the two mutation probabilities with each other where P_{mA} alters single gens and P_{mI} completely re-initializes a genome. It is expected that diversity is higher when mutations have higher probabilities, too, because they favor the global search and inject diversity by randomly altering individuals. This is supported by the corresponding diversity plots that show a general higher diversity between 4000-6000 when both mutations are at 0.1 meaning that an individual is affected by the mutation by a 10% chance. Furthermore it can be observed that the diversity is initially high but decreases rapidly during approximately 100 iterations until settles down to a steady value between 2000-4000. The highest diversity values are reached when $P_{mI} = 0.1$ thus showing that re-initialization injects high amount of diversity. However, such type of mutation needs to be applied with care because injecting diversity comes with the price of loosing already good performing

individuals. One last interesting aspect that apperas when setting both mutation probabilities to zero is that the diversity starts to fluctuate. The prior increase could be cause by the crossover itself and support the fact that longer amounts of shared information increase variation of offspring and thus the diversity. However, the diversity quickly reduces over time suggests that diversity is only injected in the eraly stages when new chromosome length are found that perform better. As soon as the length is settled the diversity starts to decrease. The population distribution does not give further insights that support or disprove the described behavior.

The results suggest that the overall GA procedure runs as expected where the random population is optimized in order to generate better coverage and time properties. This is accompanied by some exceptions or outliers that occur under some of the given parameter settings. Two aspects have not been discussed yet concerning different R_{rot} and M_{type}. Even though there are obvious deviations when it come to the map type, the additional results suggest similar conclusions as just presented. All generated data, including the ones already presented and those with rotation factor and other map type are included in the appendix.

In the next section the results of the coverage path and the altering of the environment are finally discussed.

5.2 Coverage Path Generation

Before taking a closer look to the retrain procedure, the general capabilities of the path generation for the two map types need to be discussed. Both types with generated coverage paths are shown in Figure4.6. The paths are extracted from the best 32 individuals selected by the grid search. Interestingly the coverage path for 4.6a is similar to the boustrophedon coverage pattern that is used in cell decomposition[7]. However, it is far from being optimal because plenty of region is not covered yet and the movement back towards the start position creates a lot of unnecessary crossing path segments. Furthermore, 4.6b show a coverage path that is considered good in terms of fitness. It is still far from optimal because of the different inefficient turns as well as multiple coverage segments. However, this paths show an important principle that states that the GA is able to optimize and generate a coverage path for a complex region by taking obstacles successfully into account.

Next, the adaptability of an already generated coverage path is elaborated. Figure 4.7 shows the path before 4.7a and after the repeated optimization with changed conditions 4.7b. Insights on the complete procedure and behavior of the population

is presented in Figure 4.8. Illustrated are the two stages of the optimization where the first 2000 iterations a normal coverage path is generated and subsequently the conditions are changed and the optimization continues with the current population. Furthermore, five different components are visualized for each iteration. The “Best Performing Individual” describes the coverage and time value. The GA constantly varies its solutions to get better results in those components. Of course each individual in the population features those values and the mean, min and max for them are visualized in “Population Time” and “Population Coverage”. Additionally the overall chromosome length and diversity is listed. As soon as the retrain starts the effect is viable in all time series except the diversity. All other properties suffer from a drop in their respective values cased by the changed conditions. Interestingly, the best individual reaches a new optimum after roughly 1000 iterations. This shows that the presented GA architecture is able to further adapt an already optimized paths when environment conditions change. Again the result is shown in Figure 4.7b where the red marked area is ignored by the coverage path. Furthermore, the existing solution is not destroyed but adapted such that the first part of the coverage path is still preserved and only the part intersecting with the covered region is optimized.

5.3 Future Work

The previous section analyzed the general behavior of the GA as well as the capability of retraining an already optimized population with positive results in terms of adapting to new environment conditions. However, a problem remains in the analysis method especially on how the diversity and the foundation for all presented results. The most significant problem is that all tests where based on the same random seed. Thus another random seed could mean different behavior. Furthermore, the analysis presented was on a pure observation level where only the distribution of the best individuals was examined. In order to validate the presented conclusions some statistical tests are appropriate to investigate if the observed impact was in fact statistically relevant. For example an analysis of variance (ANOVA) could be conducted after averaging out the runs with different random seeds to double check the validity and influence of each parameter in the parameter setting.

Furthermore the parameters Pop_{min}, $select$ and $keep$ where not included in the evaluation. Further tests should investigate if different variations of those opens up new possibilities to reduce the population size by simultaneously gain similar results. This highlights the issue of the overall duration that it takes to generate a coverage path as presented in Table 4.4. Generating a coverage path with as little constrains as possible

such as free selection of the neighborhood comes with computational complexity that in worst case take up to ten minutes to generate a coverage path and double the amount of time when a population is retrained. Another aspect is the duration of each run. Preliminary tests showed that the duration of a run increases with the length of the chromosome. This is caused by the evaluation process of the genome. During the evaluation each pixel in the map that is part of the path needs to be visited, possibly several times when crossing segments appear. The time complexity even increases quadratically with the resolution when more accuracy is required. This quickly gets intractable for large scale maps, not to mention an extension for the three dimensional planning task.

In order to reduce the computationally load a new method of calculating the coverage and crossing paths is required. For example utilizing bounding boxes and checking for or using a ray tracing method could yield further improvements. Such method could be furthermore optimized by hardware acceleration.

Despite the coverage calculation, there are more possibilities in order to improve the optimization. The $Elite$ procedure could be combined with another selection procedure such as the $Turn$ selection. This could be employed to create the different family pools that then determine the new population instead of randomly creating families and possibly even converge faster.

Two other extensions that could not be tested sufficiently are presented in Section 3.5.2 that describe adaptive and deterministic parameter variation during the optimization. This could improve the convergence speed and promote better divergence throughout the optimization and further reduce the overall parameter space. In this context it needs to be investigated if a higher diversity has also a positive impact when it comes to changing environment conditions.

One other possible extension concerns the genome representation. As suggested by the general principles of the CPP[1] only simple movements are allowed such as straight movement along a line or rotations. The presented action representation already enables to optimize different kinds of actions such as the end action that introduces a penalty when the robot does not reach its start position. Similar actions could be implemented that enable movement along a radius or utilize spline curves to better navigate around obstacles. Furthermore, different kinds of speeds could be utilized, e.g. when the robot cleans it drives slower in comparison to its normal movement speed. This could be utilized to drive more quickly through already covered region.

One last aspect considering the fitness that only features the bare minimum of constraints such as coverage minimal, path crossing segments and obstacle avoidance.

Other approaches also consider energy constraints when covering given area[35, 4]. Even in the simple scenario of a cleaning robot this can be from great importance in order to prevent it from running out of energy along the coverage path. The current fitness function only tries to minimize the path length by reducing the crossing path segments. This results in a time constraint that is directly dependent on the amount of area that is already covered. A penalty on the energy could introduce an additional more independent approach that reduces the path length by taking the current battery charge into account.

6 Conclusion

This work presented a coverage path planning approach for a single robot scenario. Main goal is to generate a coverage path that is nearly optimal in its coverage and furthermore minimal in the time required to follow it. Genetic Algorithms are utilized with a custom genome representation that consists of variable chromosome lengths. Furthermore, a special crossover is implemented that promotes the dynamic length adaptation of the chromosome. Changes in length of the chromosome corresponds to adding or removing new path segments to the coverage path. Additionally, during the crossover different properties can be preserved and promote the overall search process by generating offspring individuals that preserve or change the locality of the shared path segments. Several correction procedures are introduced that help to guide the overall search process. Subsequently different selection schemes are presented where the custom implementation of the $Elite$ selection shows the most promising results followed by the $Turn$ selection. A grid search is performed on the presented parameter setting in order to elaborate the overall path generation capabilities. The results of the generated coverage paths are promising and even show patterns that are similar to already known coverage patterns. Furthermore, the capability of adapting already optimized populations to new environment conditions was tested. Results show that the proposed GA architecture is able to further adapt such a population. Finally some issues of the overall time complexity are discussed that needs to be addressed before extending the procedure to a multi robot scenario.

In summary, the presented approach showed great capability in generating a coverage path for a single robot scenario with a customized GA architecture and operators. Furthermore, the capabilities of adapting to altering environment conditions enable a further extension to a multi robot scenario.

Bibliography

[1] Zuo Cao, Yuyu Huang, and Ernest Hall. “Region filling operations with random obstacle avoidance for mobile robots”. In: *Journal of Robotic Systems* 5 (Apr. 1988), pp. 87–102. DOI: 10.1002/rob.4620050202.

[2] Md Habib, Mohammad Alam, and Nazmul Siddique. “Optimizing Coverage Performance of Multiple Random Path-planning Robots”. In: *Paladyn* 3 (Jan. 2012). DOI: 10.2478/s13230-012-0012-5.

[3] Timo Oksanen and Arto Visala. “Coverage path planning algorithms for agricultural field machines”. In: *Journal of Field Robotics* 26.8 (2009), pp. 651–668. DOI: https://doi.org/10.1002/rob.20300. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20300. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20300.

[4] B. Sun, D. Zhu, C. Tian, and C. Luo. “Complete Coverage Autonomous Underwater Vehicles Path Planning Based on Glasius Bio-Inspired Neural Network Algorithm for Discrete and Centralized Programming”. In: *IEEE Transactions on Cognitive and Developmental Systems* 11.1 (2019), pp. 73–84. DOI: 10.1109/TCDS.2018.2810235.

[5] "*Search and Rescue Robots – Current Applications on Land, Sea, and Air*". https://emerj.com/ai-sector-overviews/search-and-rescue-robots-current-applications/. [Online; accessed 16-03-2021].

[6] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, and Yahya Zweiri. “A survey on multi-robot coverage path planning for model reconstruction and mapping”. In: *SN Applied Sciences* 1 (Aug. 2019). DOI: 10.1007/s42452-019-0872-y.

[7] Howie Choset. “Coverage for robotics–a survey of recent results”. In: *Annals of mathematics and artificial intelligence* 31.1 (2001), pp. 113–126.

[8] Enric Galceran and Marc Carreras. “A survey on coverage path planning for robotics”. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2013.09.004. URL: http://www.sciencedirect.com/science/article/pii/S092188901300167X.

[9] Raja Sankar Dileep Muddu, Dan Wu, and Libing Wu. “A frontier based multi-robot approach for coverage of unknown environments”. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2015, pp. 72–77.

[10] Shoudong Huang and Gamini Dissanayake. “Robot Localization: An Introduction”. In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. American Cancer Society, 2016, pp. 1–10. ISBN: 9780471346081. DOI: https://doi.org/10.1002/047134608X.W8318. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047134608X.W8318. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W8318.

[11] Athanasios Kapoutsis, Savvas Chatzichristofis, and Elias Kosmatopoulos. “DARP: Divide Areas Algorithm for Optimal Multi-Robot Coverage Path Planning”. In: *Journal of Intelligent & Robotic Systems* 86 (June 2017). DOI: 10.1007/s10846-016-0461-x.

[12] Gustavo SC Avellar, Guilherme AS Pereira, Luciano CA Pimenta, and Paulo Iscold. “Multi-UAV routing for area coverage and remote sensing with minimum time”. In: *Sensors* 15.11 (2015), pp. 27783–27803.

[13] Muzaffer Kapanoglu, Mete Alikalfa, Metin Ozkan, Osman Parlaktuna, et al. “A pattern-based genetic algorithm for multi-robot coverage path planning minimizing completion time”. In: *Journal of intelligent manufacturing* 23.4 (2012), pp. 1035–1045.

[14] Enric Galceran and Marc Carreras. “A survey on coverage path planning for robotics”. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2013.09.004. URL: http://www.sciencedirect.com/science/article/pii/S092188901300167X.

[15] Adiyabaatar Janchiv, Batsaikhan Dugarjav, Byungsoo Kim, et al. “Time-Efficient and Complete Coverage Path Planning Based on Flow Networks for Multi-Robots”. In: *International Journal of Control, Automation and Systems* 11 (Apr. 2013). DOI: 10.1007/s12555-011-0184-5.

[16] Héctor Azpúrua, Gustavo M Freitas, Douglas G Macharet, and Mario FM Campos. “Multi-robot coverage path planning using hexagonal segmentation for geophysical surveys”. In: *Robotica* 36.8 (2018), pp. 1144–1166.

[17] B. Yang, Y. Ding, and K. Hao. “Area coverage searching for swarm robots using dynamic Voronoi-based method”. In: *2015 34th Chinese Control Conference (CCC)*. 2015, pp. 6090–6094. DOI: 10.1109/ChiCC.2015.7260592.

[18] Y. Gabriely and E. Rimon. “Spanning-tree based coverage of continuous areas by a mobile robot”. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, 1927–1933 vol.2. DOI: 10.1109/ROBOT.2001.932890.

[19] JOHN R. CURRENT and DAVID A. SCHILLING. “The Covering Salesman Problem”. In: *Transportation Science* 23.3 (1989), pp. 208–213. ISSN: 00411655, 15265447. URL: http://www.jstor.org/stable/25768381.

[20] Gilbert Laporte and Yves Nobert. “Exact Algorithms for the Vehicle Routing Problem**The authors are grateful to the Canadian Natural Sciences and Engineering Research Council (grants A4747 and A5486) and to the Quebec Government (FCAC grant 80EQ04228) for their financial support.” In: *Surveys in Combinatorial Optimization*. Ed. by Silvano Martello, Gilbert Laporte, Michel Minoux, and Celso Ribeiro. Vol. 132. North-Holland Mathematics Studies. North-Holland, 1987, pp. 147–184. DOI: https://doi.org/10.1016/S0304-0208(08)73235-3. URL: https://www.sciencedirect.com/science/article/pii/S0304020808732353.

[21] Ahmad Hassanat, Khalid Almohammadi, Esra’a Alkafaween, et al. “Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach”. In: *Information* 10.12 (2019). ISSN: 2078-2489. DOI: 10.3390/info10120390. URL: https://www.mdpi.com/2078-2489/10/12/390.

[22] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[23] John H. Holland. “Genetic Algorithms”. In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939139.

[24] Deepti Gupta and Shabina Ghafir. *An Overview of methods maintaining Diversity in Genetic Algorithms*. 2012.

[25] Noraini Razali and John Geraghty. “Genetic Algorithm Performance with Different Selection Strategies in Solving TSP”. In: vol. 2. Jan. 2011.

[26] Padmavathi Kora and Priyanka Yadlapalli. “Crossover operators in genetic algorithms: A review”. In: *International Journal of Computer Applications* 162.10 (2017).

[27] Heinz Muhlenbein and Dirk Schlierkamp-Voosen. “Optimal interaction of mutation and crossover in the breeder genetic algorithm”. In: *Proceedings of the Fifth International Conference on Genetic Algorithms*. Citeseer. 1993.

[28] B. McGinley, J. Maher, C. O'Riordan, and F. Morgan. "Maintaining Healthy Population Diversity Using Adaptive Crossover, Mutation, and Selection". In: *IEEE Transactions on Evolutionary Computation* 15.5 (2011), pp. 692–714. DOI: 10.1109/TEVC.2010.2046173.

[29] Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. "Parameter control in evolutionary algorithms". In: *IEEE Transactions on evolutionary computation* 3.2 (1999), pp. 124–141.

[30] Arnold Bregt, J. DENNEBOOM, and Huynh Van Y. "Determination of rasterizing error: a case study with the soil map of The Netherlands". In: *International Journal of Geographical Information Science* 5 (July 1991), pp. 361–367. DOI: 10.1080/02693799108927861.

[31] Yanrong Hu and S. X. Yang. "A knowledge based genetic algorithm for path planning of a mobile robot". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. Vol. 5. 2004, 4350–4355 Vol.5. DOI: 10.1109/ROBOT.2004.1302402.

[32] Amna Khan, Iram Noreen, and Z. Habib. "On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges". In: *J. Inf. Sci. Eng.* 33 (2017), pp. 101–121.

[33] Edmund Burke, Steven Gustafson, and Graham Kendall. "A Survey and Analysis of Diversity Measures in Genetic Programming". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. GECCO'02. New York City, New York: Morgan Kaufmann Publishers Inc., 2002, pp. 716–723. ISBN: 1558608788.

[34] David E. Goldberg and Kalyanmoy Deb. "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms". In: ed. by GREGORY J.E. RAWLINS. Vol. 1. Foundations of Genetic Algorithms. Elsevier, 1991, pp. 69–93. DOI: https://doi.org/10.1016/B978-0-08-050684-5.50008-2. URL: http://www.sciencedirect.com/science/article/pii/B9780080506845500082.

[35] K. R. Jensen-Nau, T. Hermans, and K. K. Leang. "Near-Optimal Area-Coverage Path Planning of Energy-Constrained Aerial Robots With Application in Autonomous Environmental Monitoring". In: *IEEE Transactions on Automation Science and Engineering* (2020), pp. 1–16. DOI: 10.1109/TASE.2020.3016276.

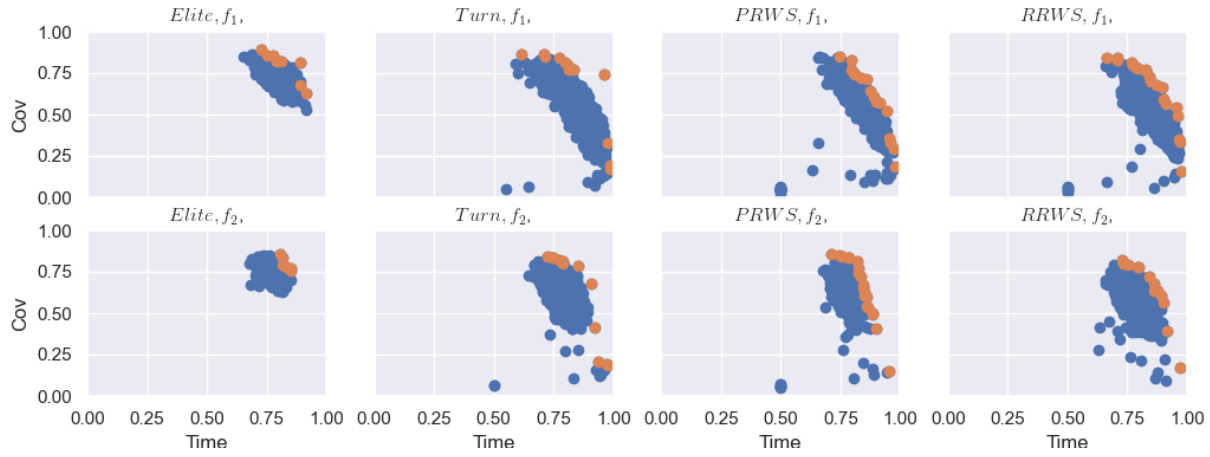
A Appendix

A.1 Evaluation Results

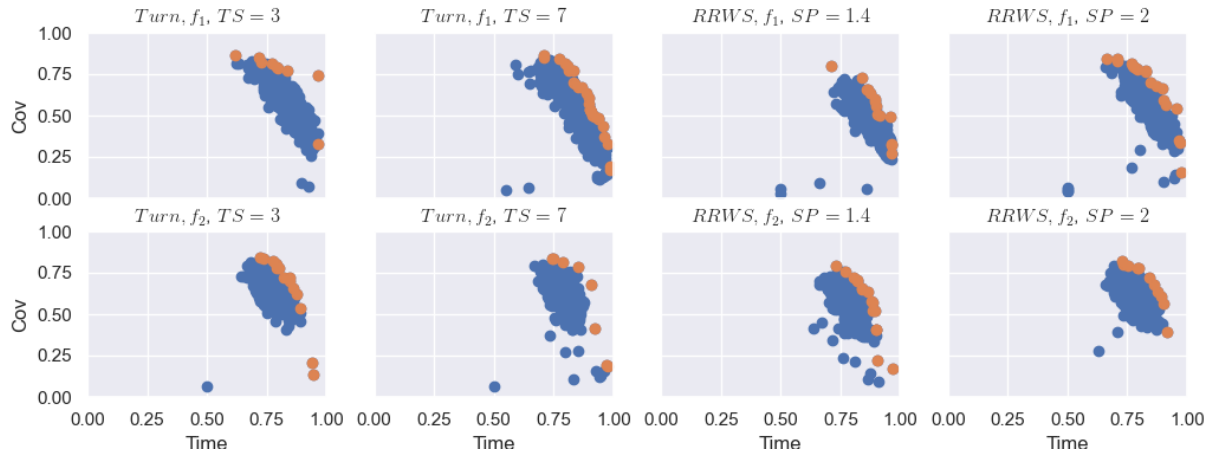
The following section shows the general results of the grid search with parameters presented in Table 4.3. The Results are grouped by map type and with and without rotation parameter.

One point in a scatter plot always refers to the best individual in the complete population of one run. Test ref A.2 and next woth A.6

A.1.1 Grid Search Results

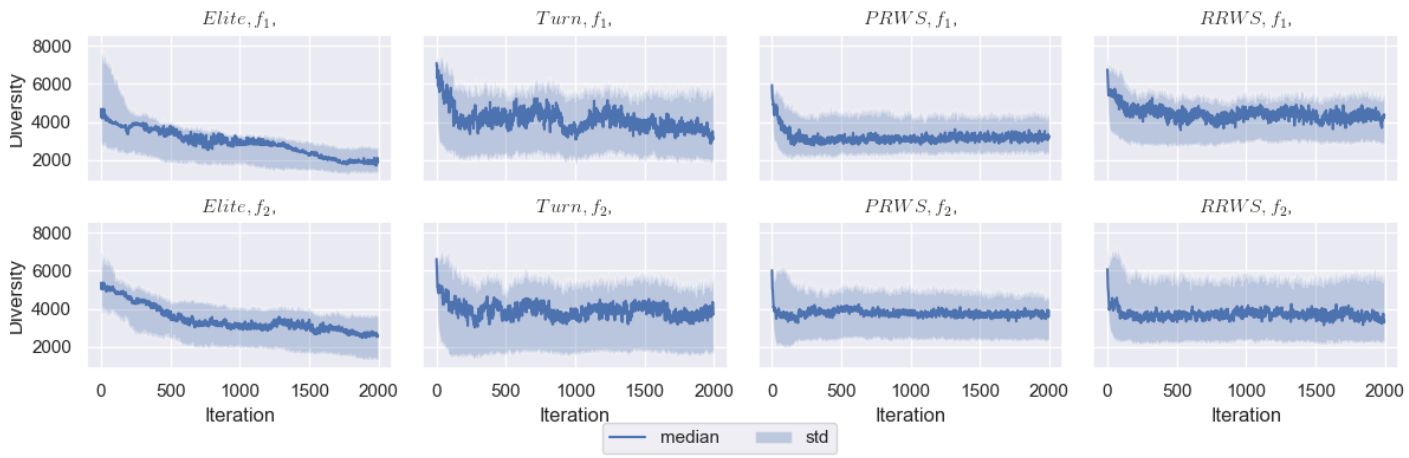


(a) Selection

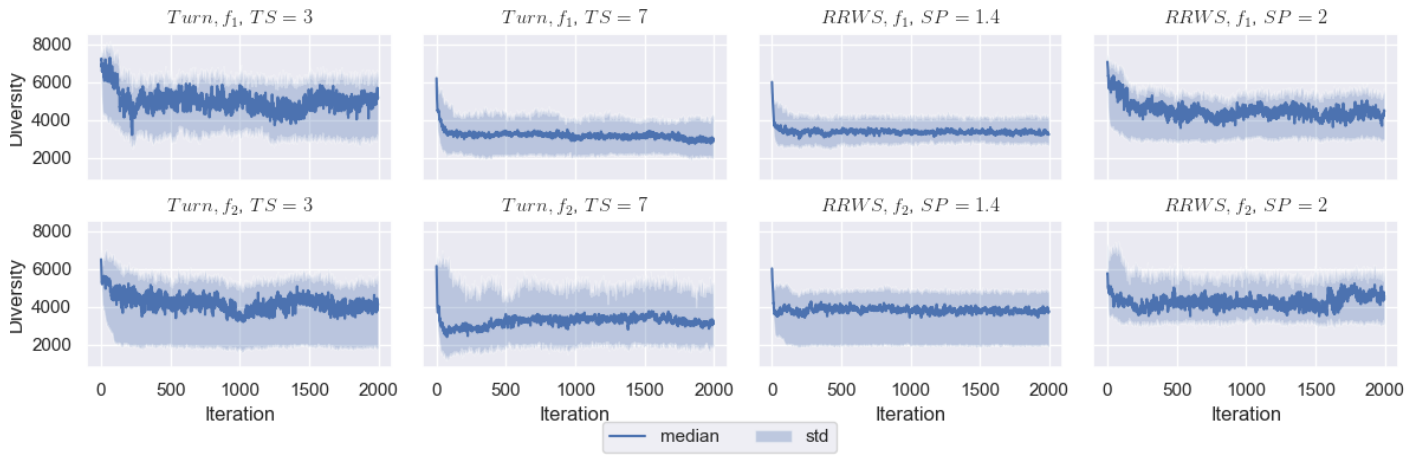


(b) Selection Pressure

Figure A.1: Population distribution for Selection, $M_{type} = 1, R_{rot} = 0$

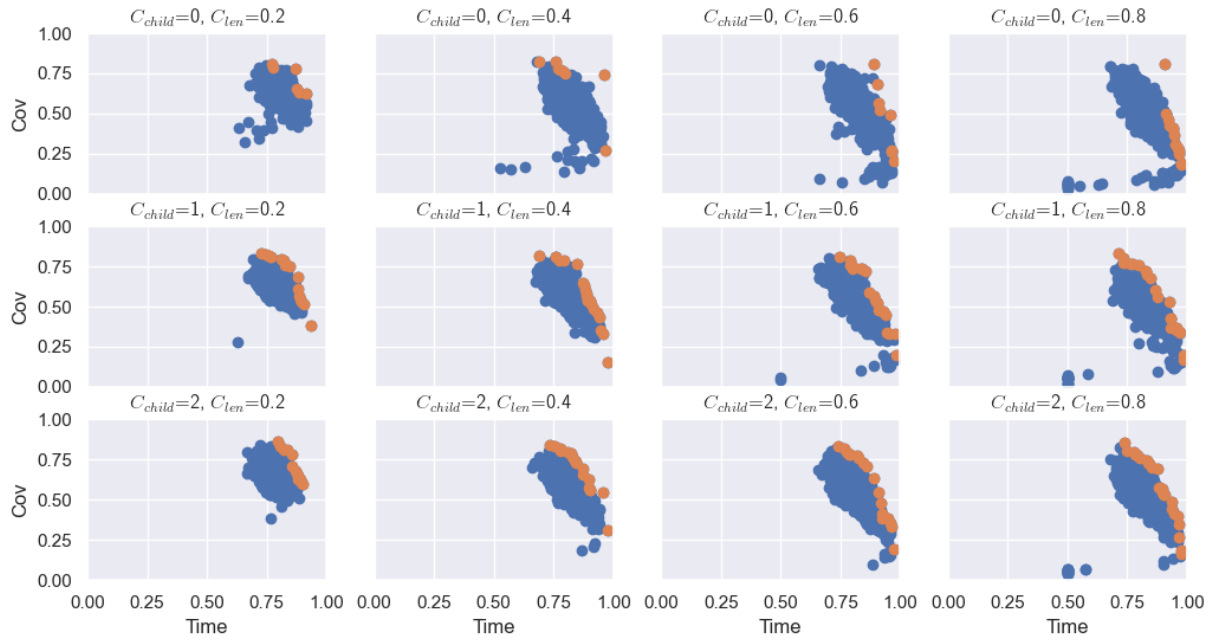


(a) Selection

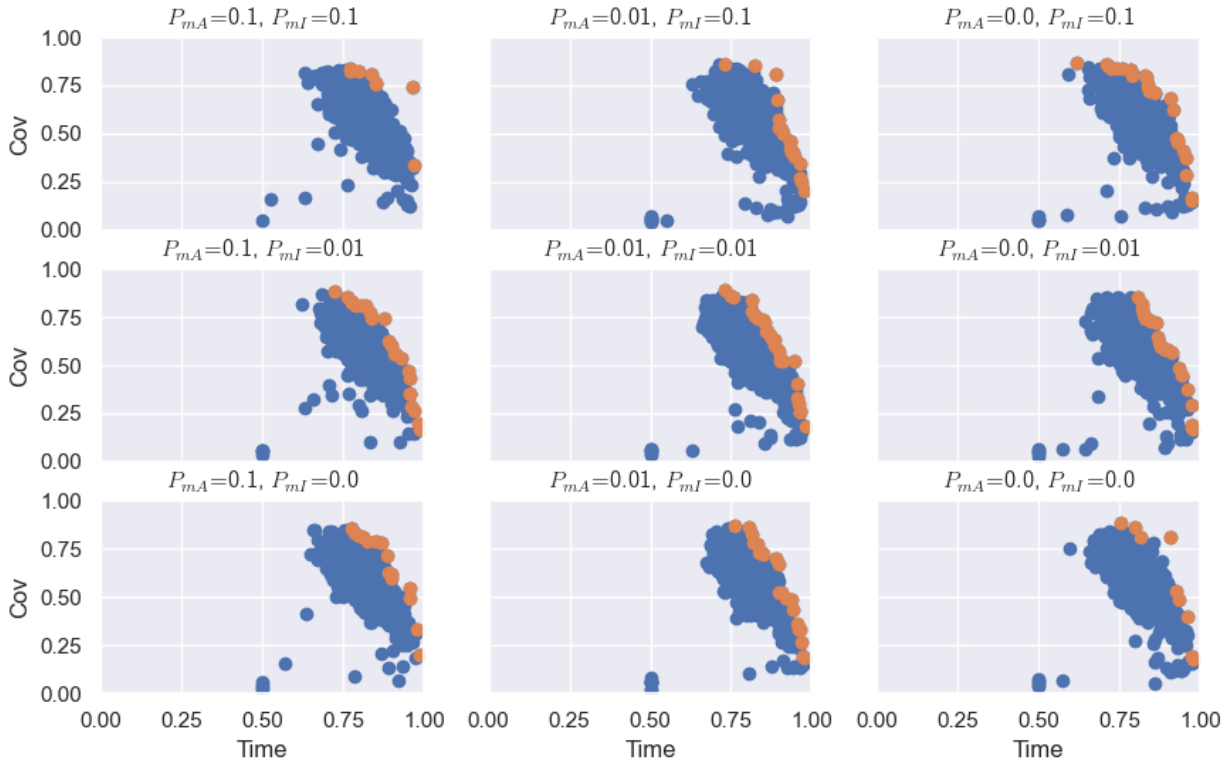


(b) Selection Pressure

Figure A.2: Diversity for Selection, $M_{type} = 1, R_{rot} = 0$

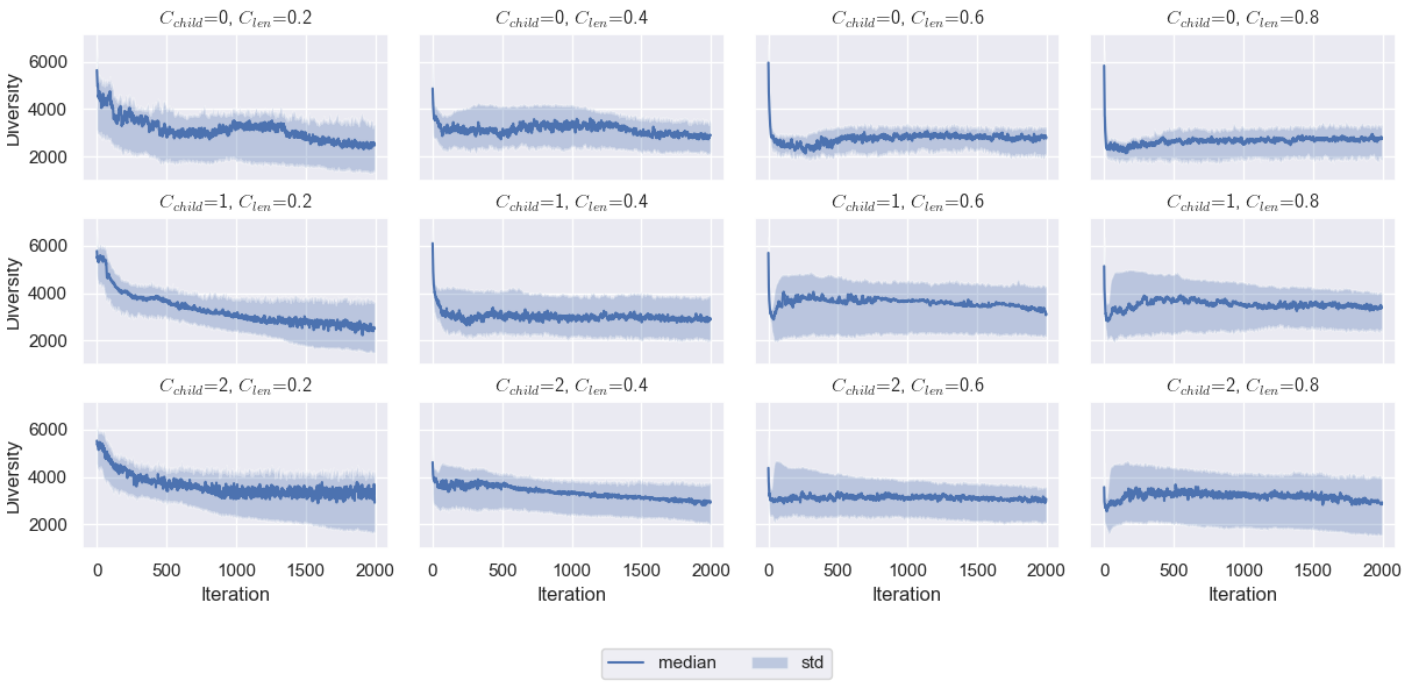


(a) Crossover

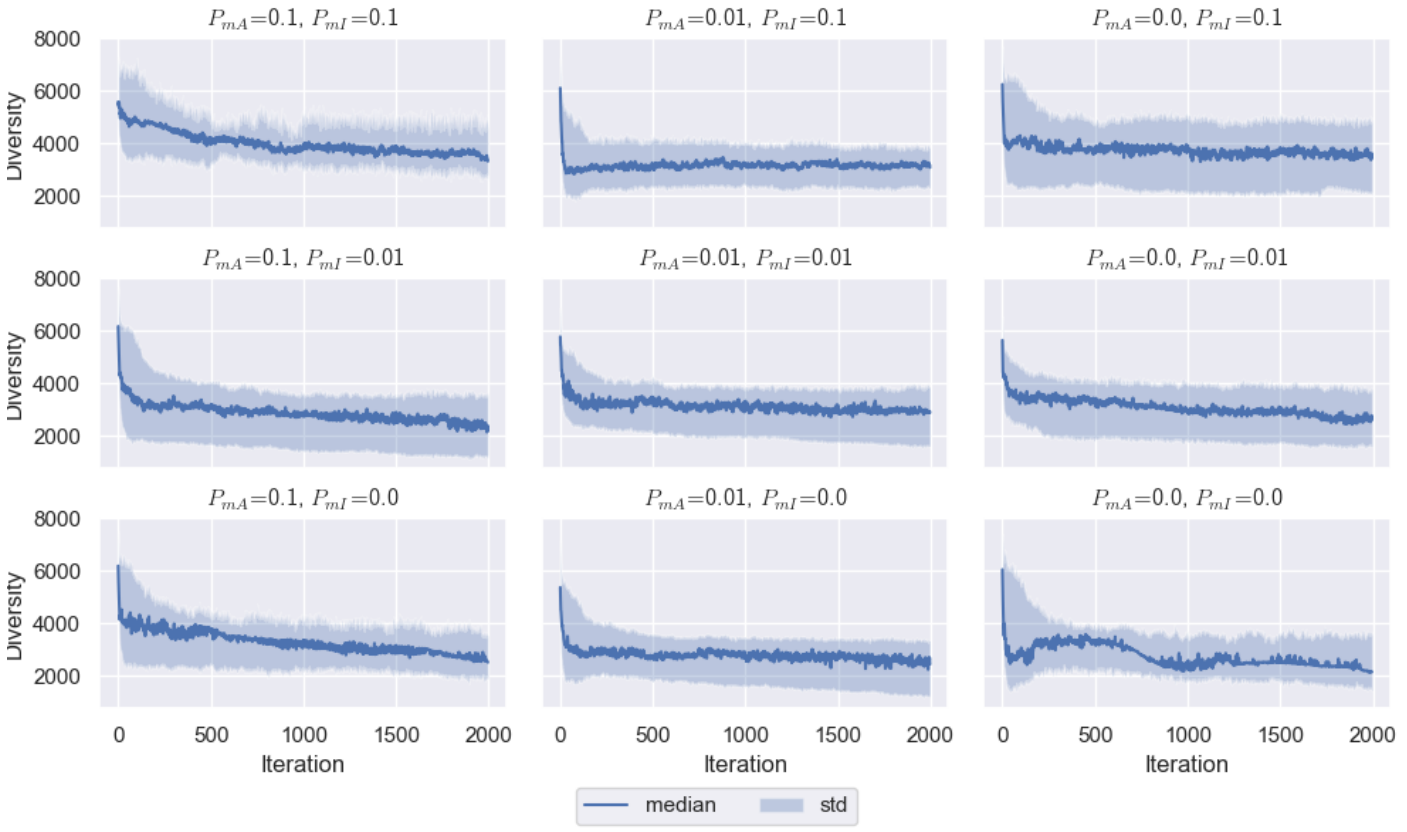


(b) Mutation

Figure A.3: Population distribution for crossover and mutation, $M_{type} = 1$, $R_{rot} = 0$

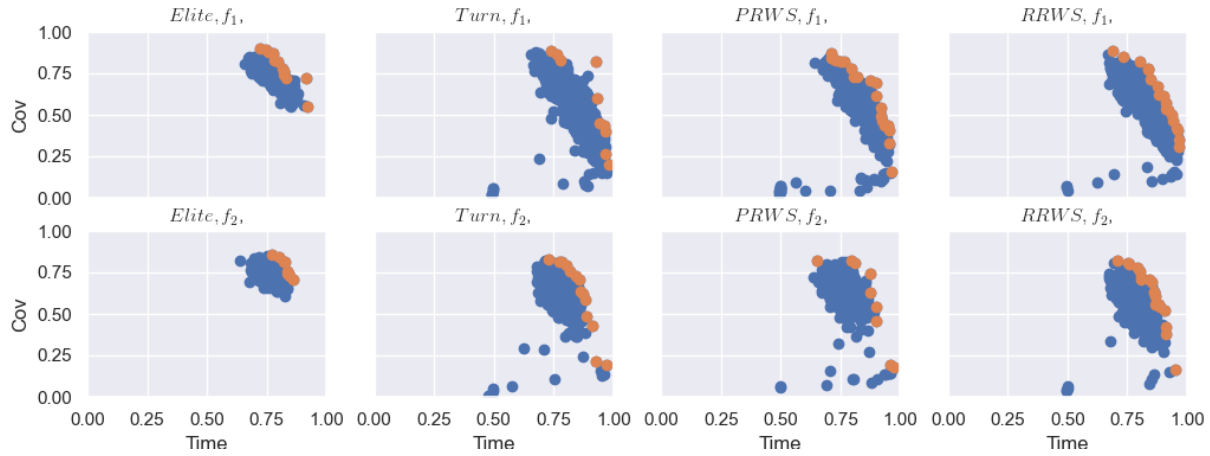


(a) Crossover

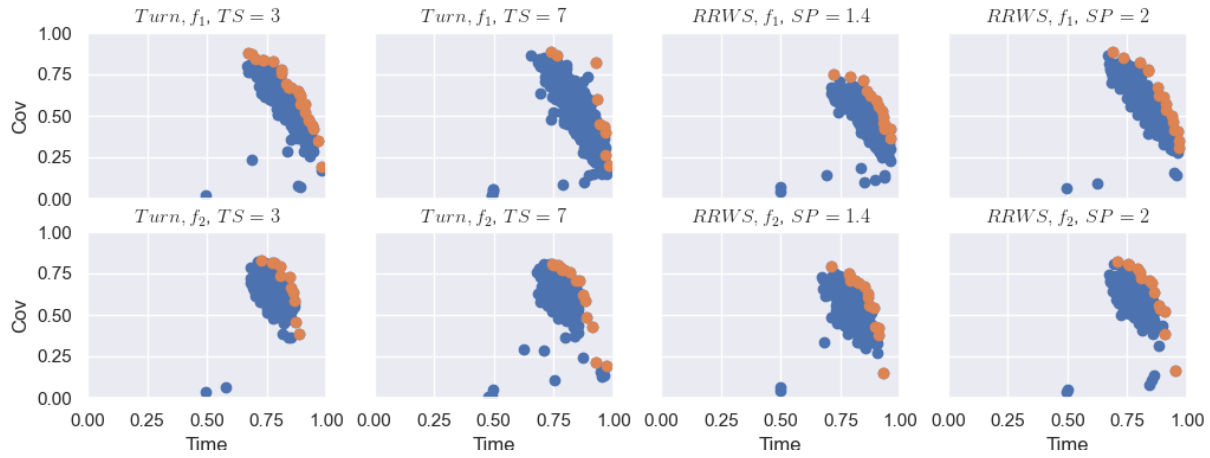


(b) Mutation

Figure A.4: Diversity for Crossover and mutation, $M_{type} = 1, R_{rot} = 0$

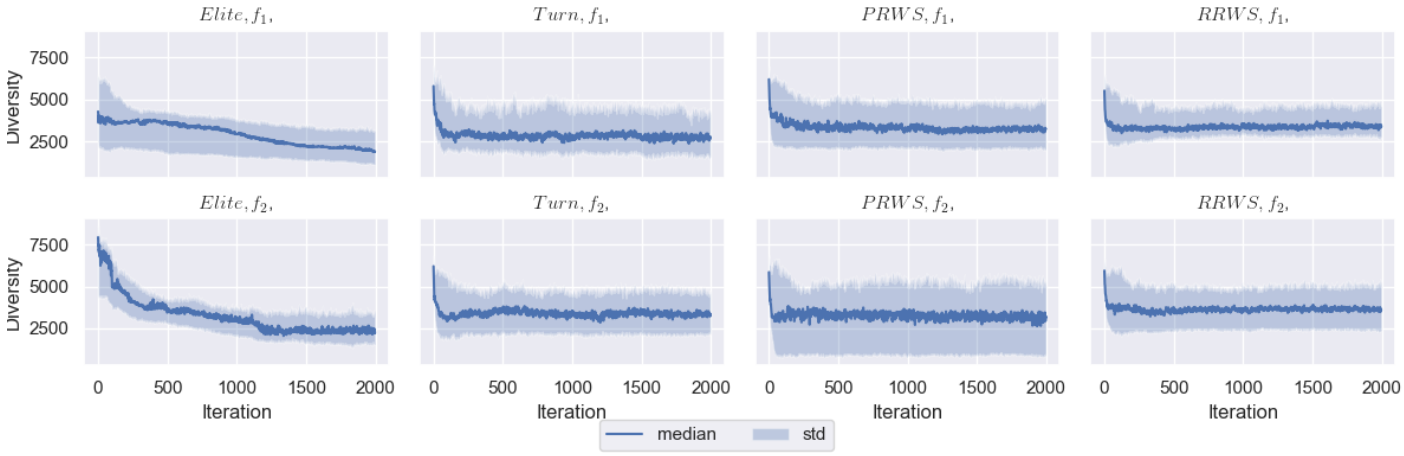


(a) Selection

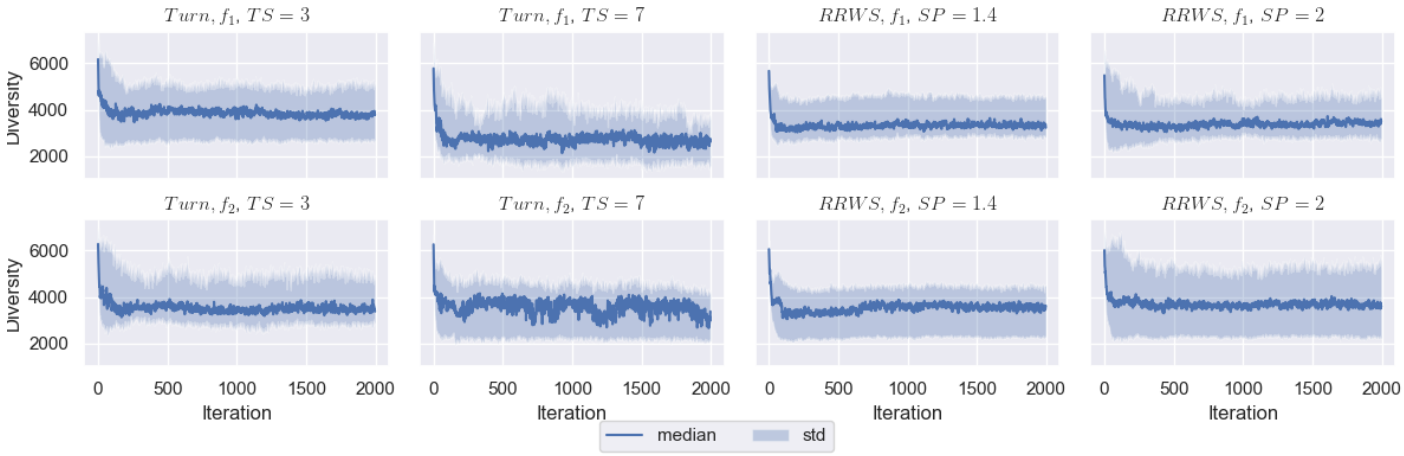


(b) Selection Pressure

Figure A.5: Population distribution for Selection, $M_{type} = 1, R_{rot} = 1$

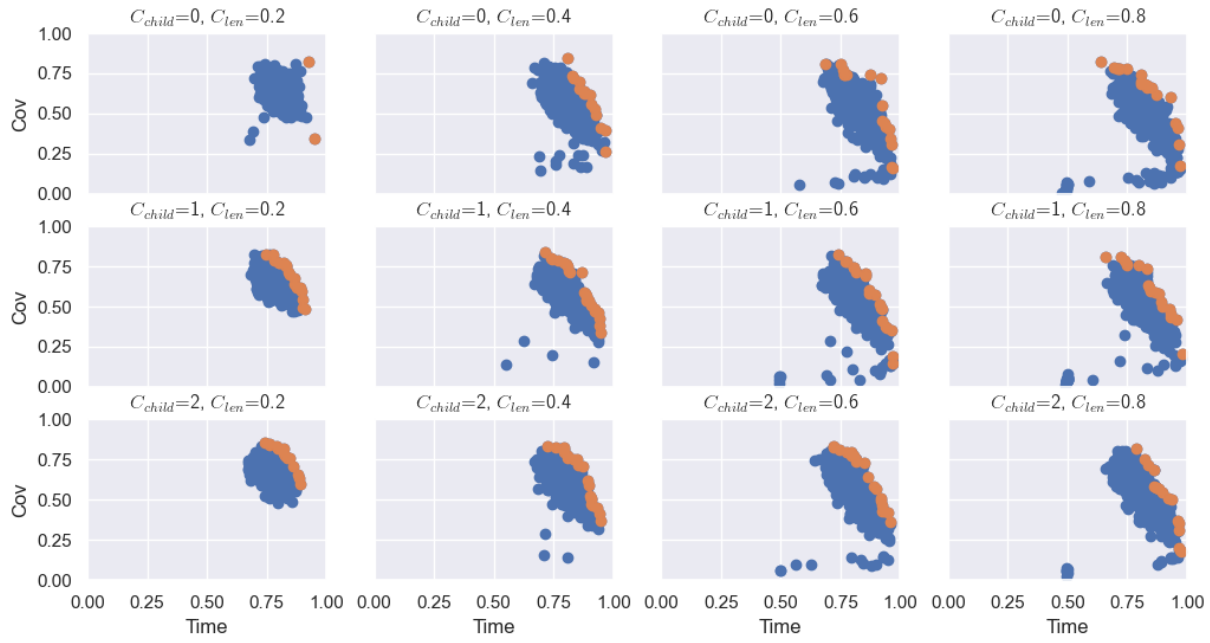


(a) Selection

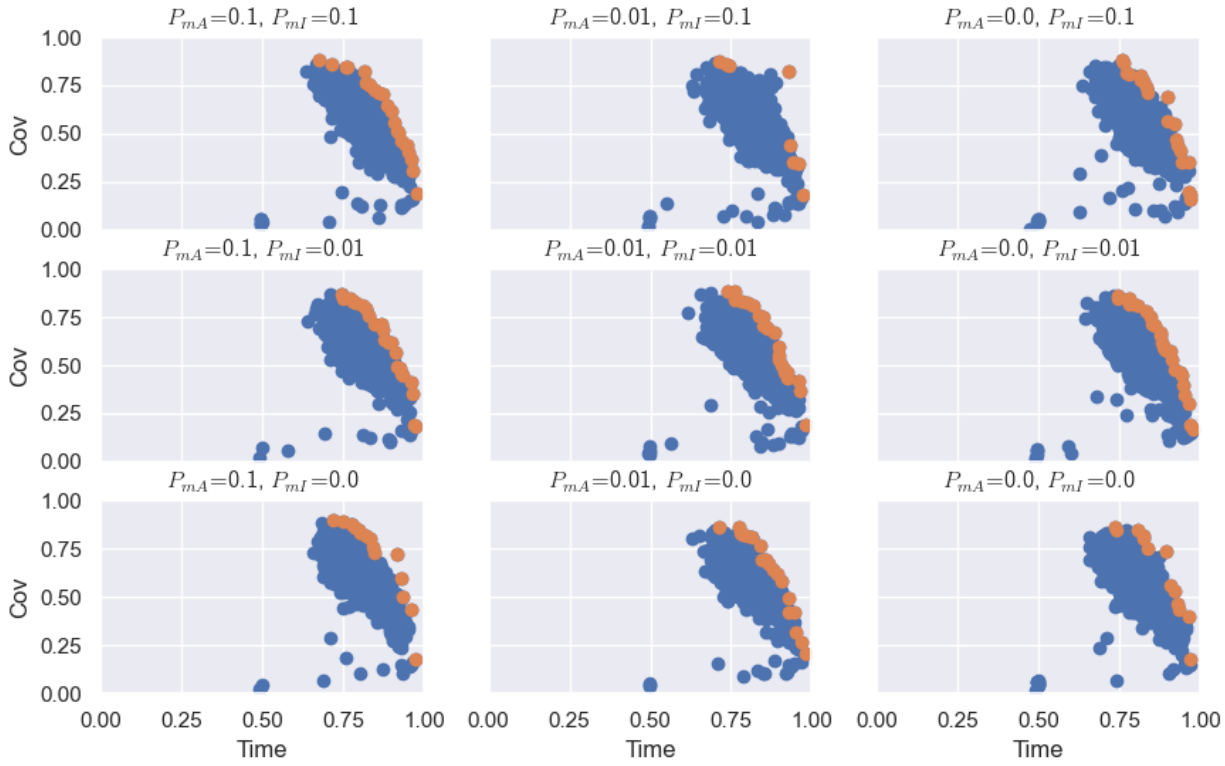


(b) Selection Pressure

Figure A.6: Diversity for Selection, $M_{type} = 1, R_{rot} = 1$

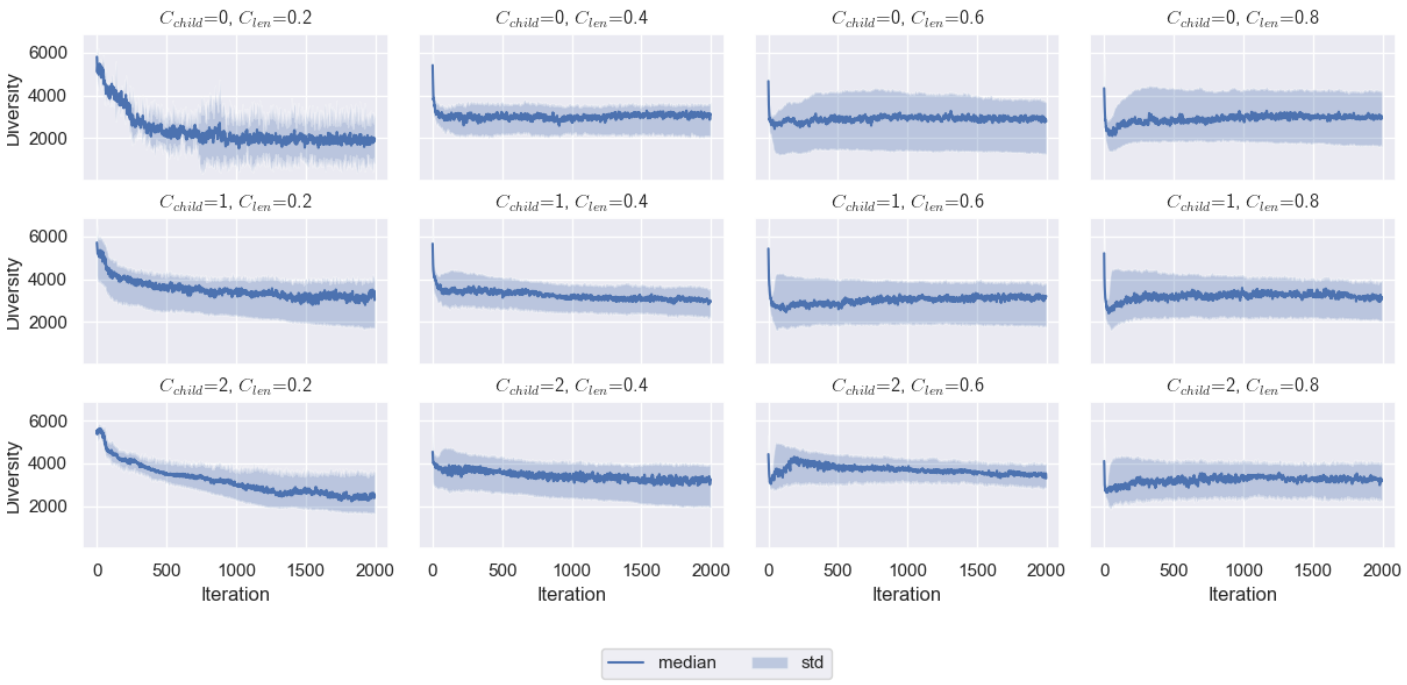


(a) Crossover

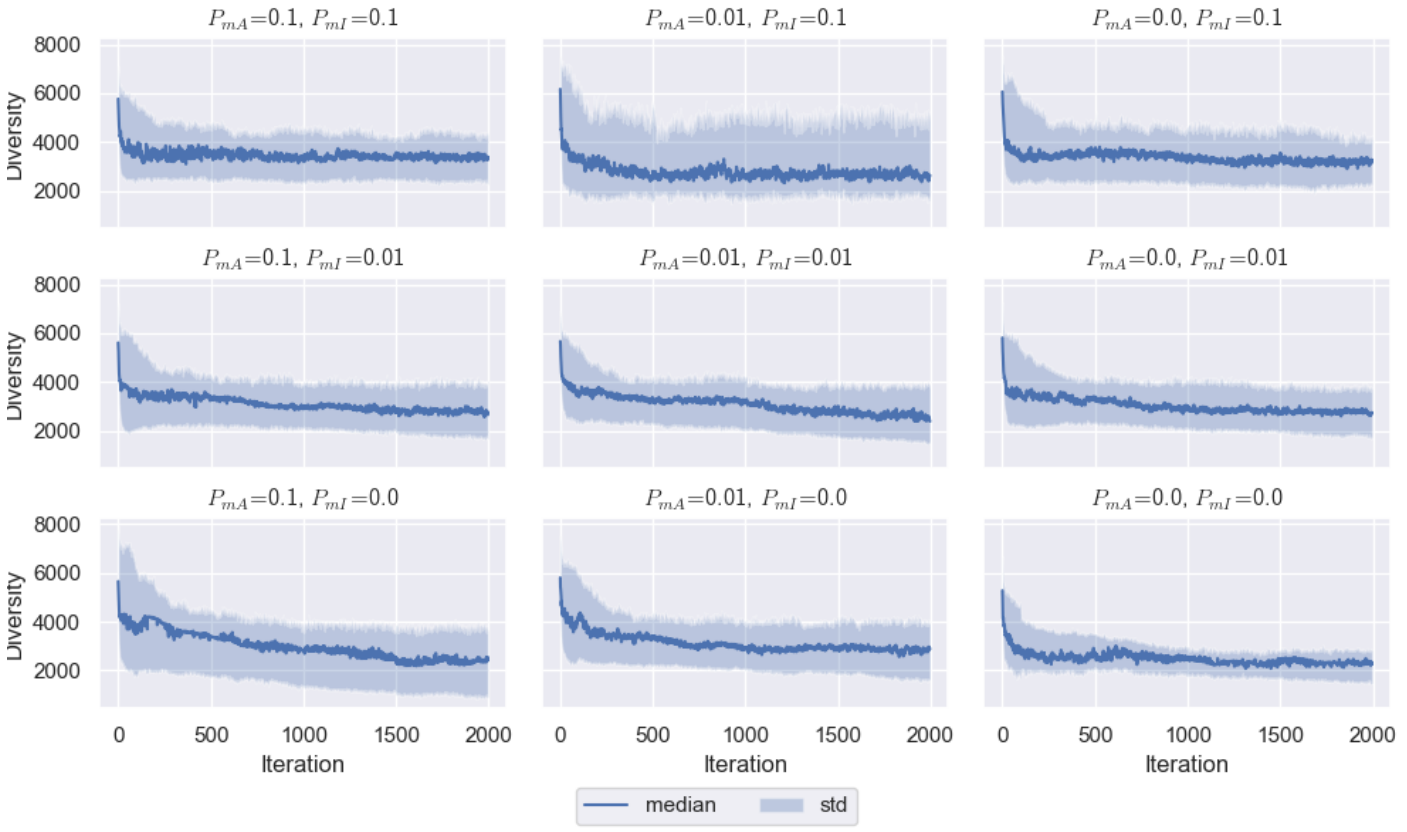


(b) Mutation

Figure A.7: Population distribution for crossover and mutation, $M_{type} = 1$, $R_{rot} = 1$

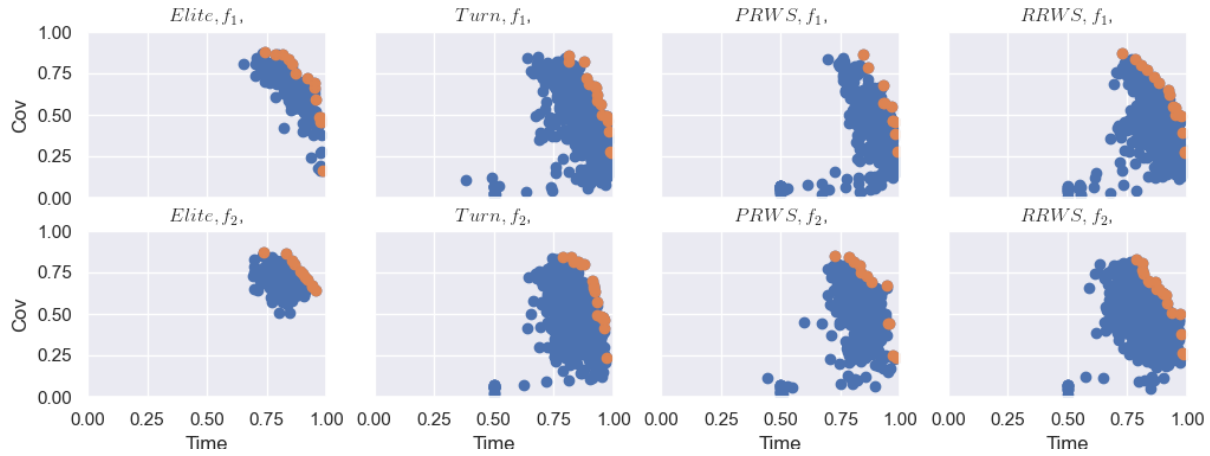


(a) Crossover

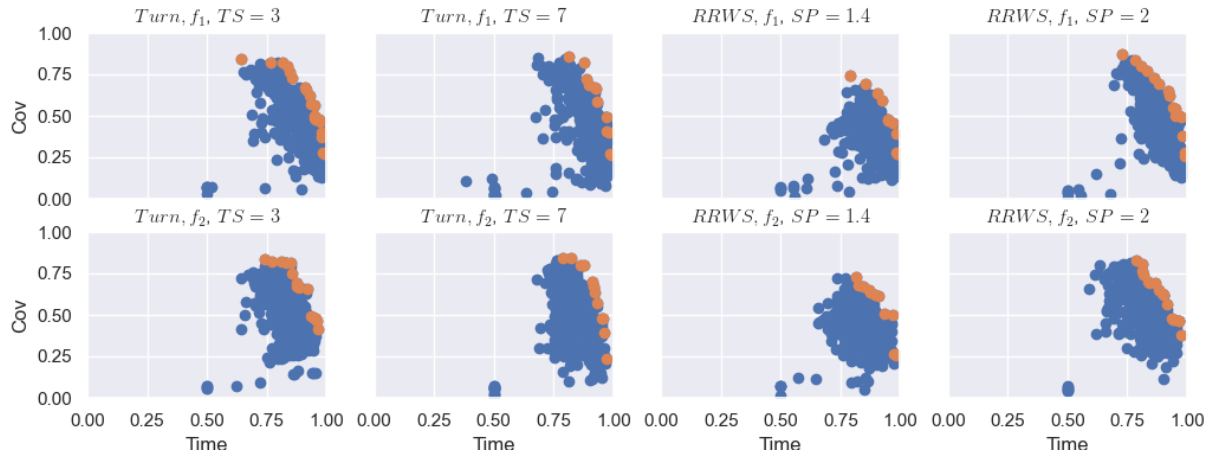


(b) Mutation

Figure A.8: Diversity for Crossover and mutation, $M_{type} = 1, R_{rot} = 1$

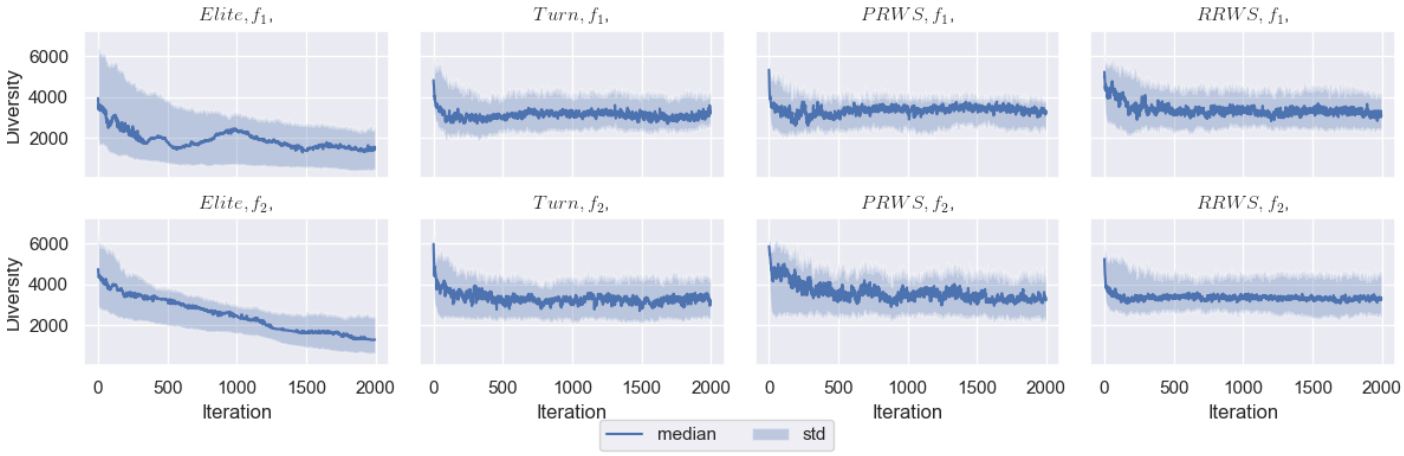


(a) Selection

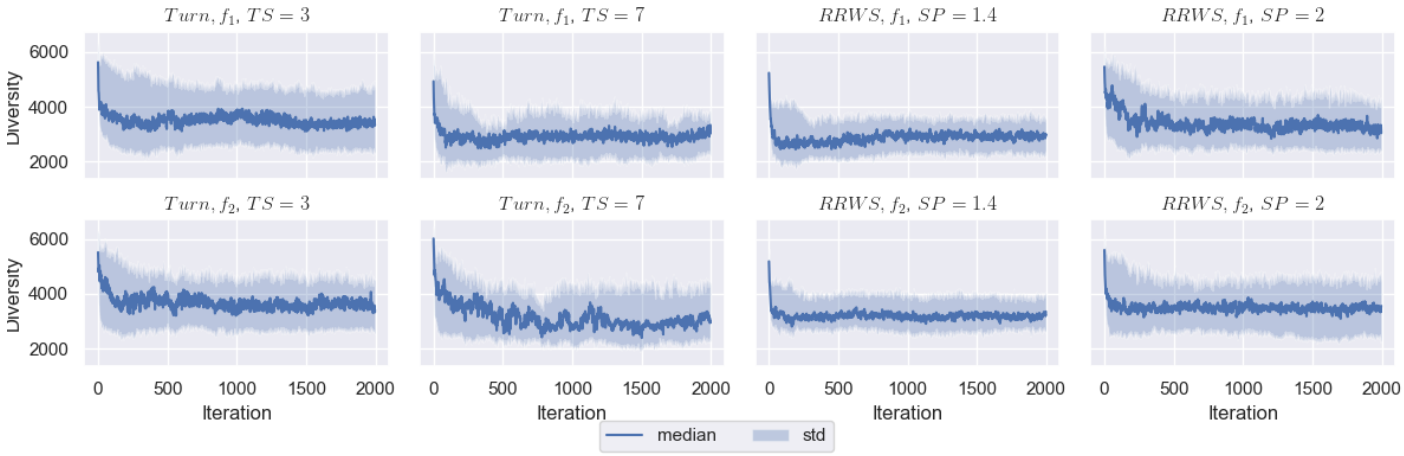


(b) Selection Pressure

Figure A.9: Population distribution for Selection, $M_{type} = 2$, $R_{rot} = 0$

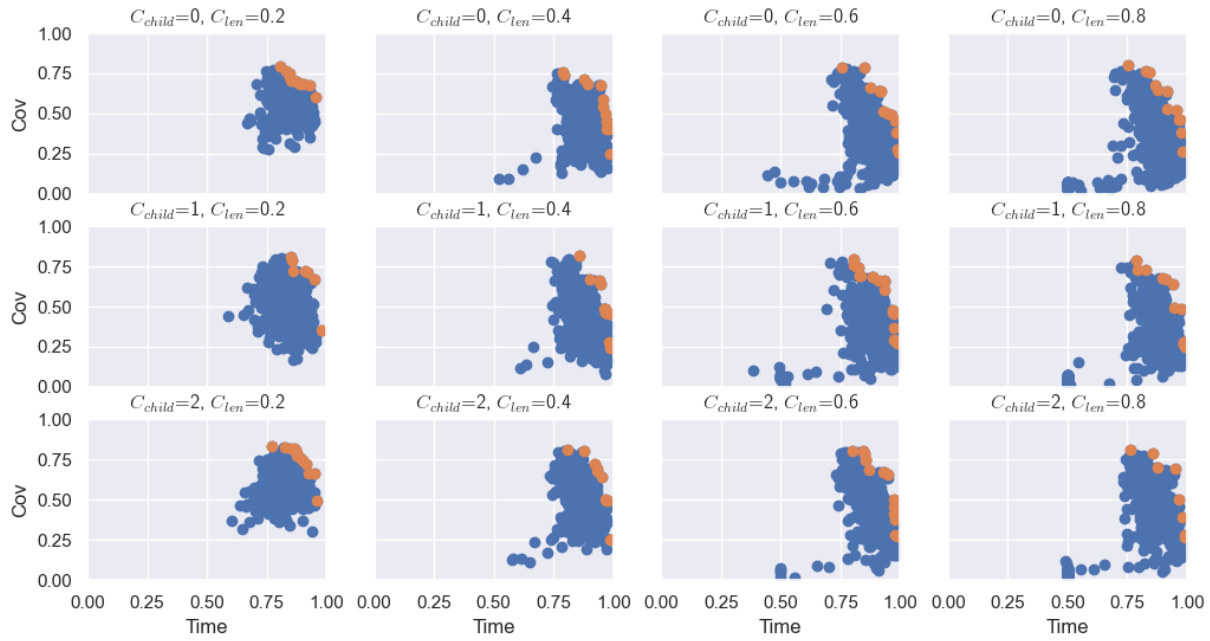


(a) Selection

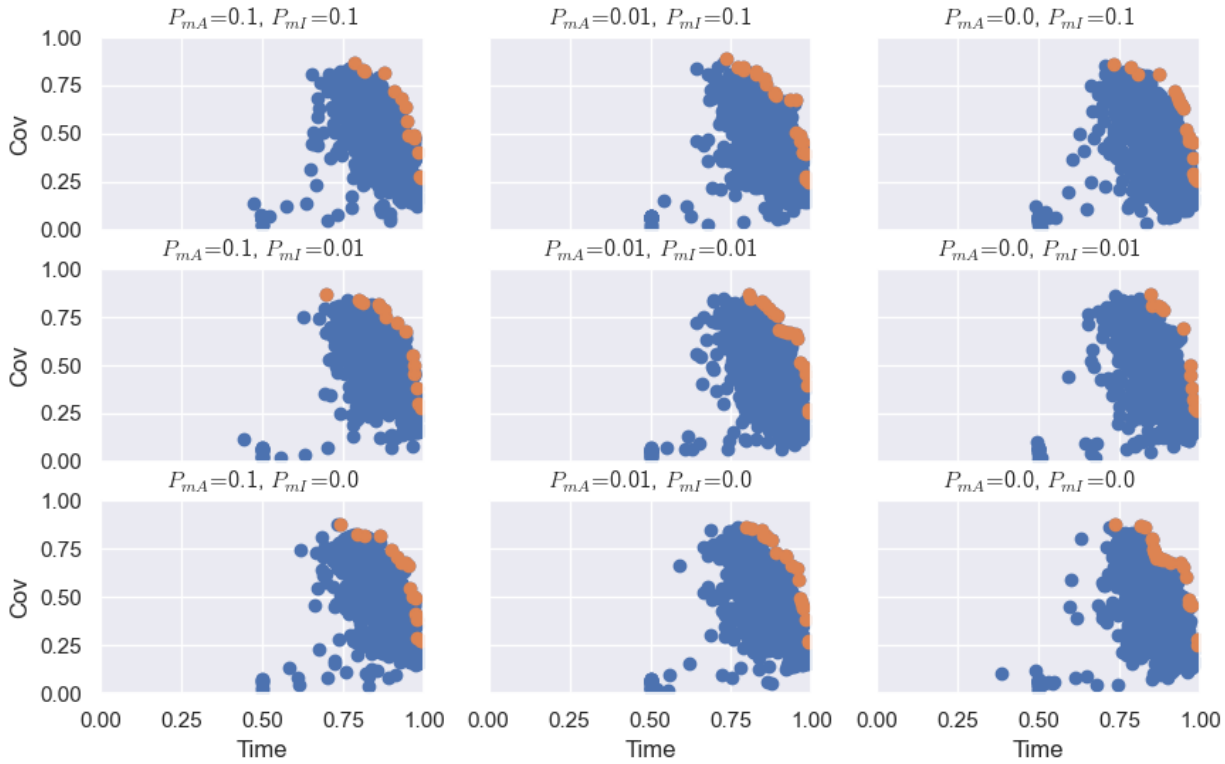


(b) Selection Pressure

Figure A.10: Diversity for Selection, $M_{type} = 2, R_{rot} = 0$

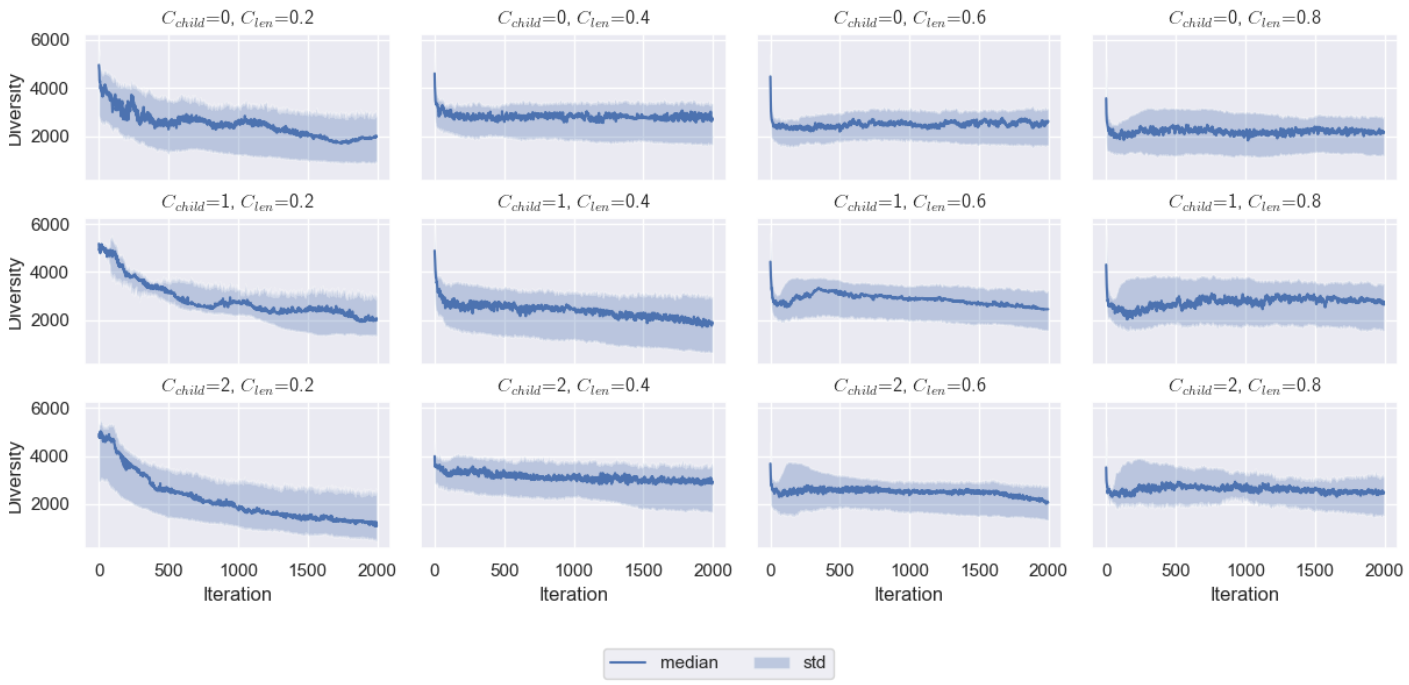


(a) Crossover

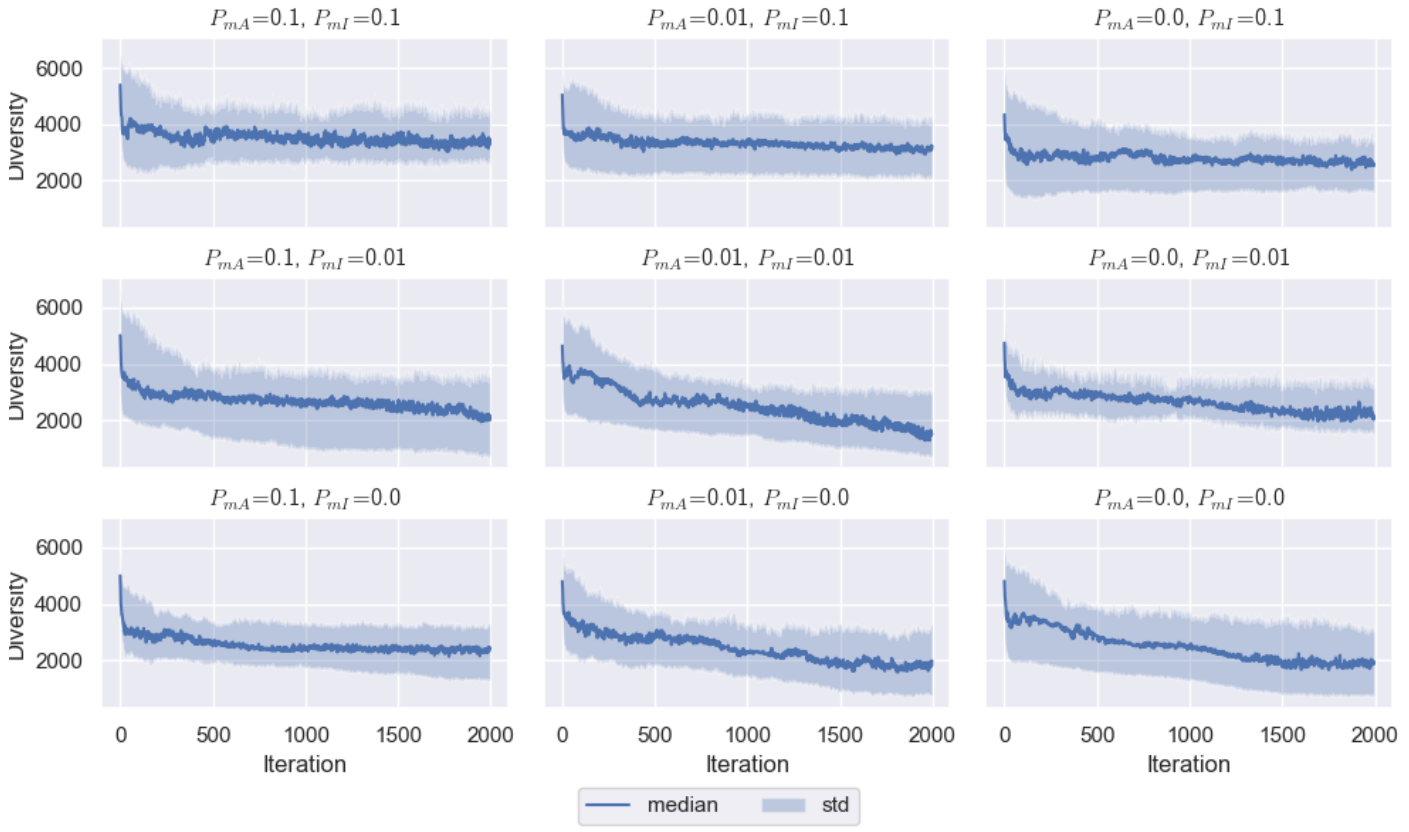


(b) Mutation

Figure A.11: Population distribution for crossover and mutation, $M_{type} = 2$, $R_{rot} = 0$

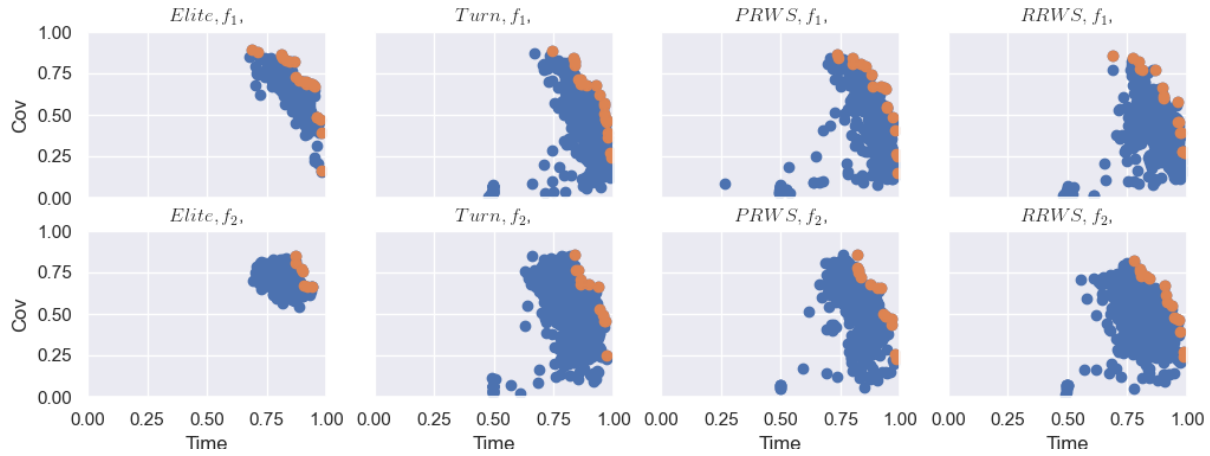


(a) Crossover

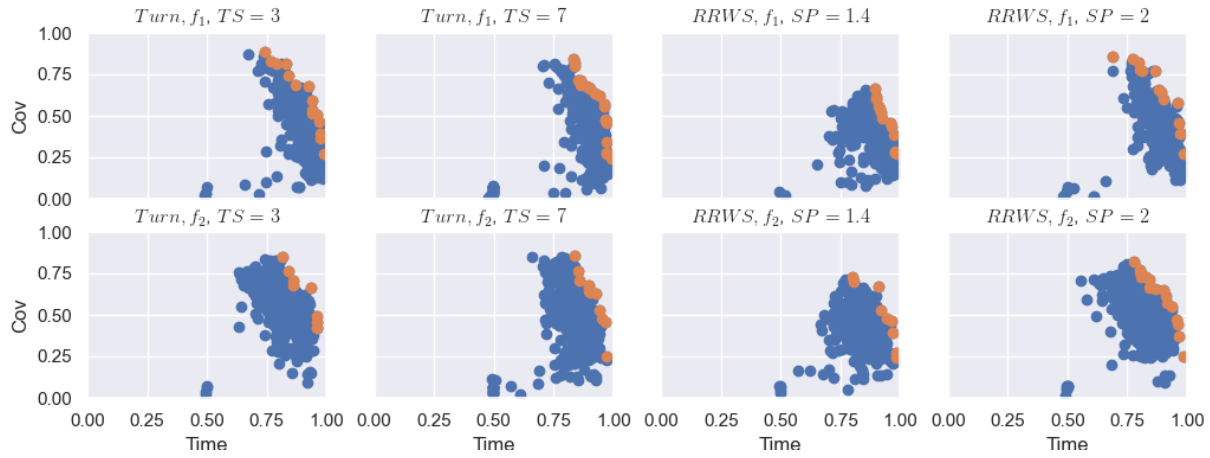


(b) Mutation

Figure A.12: Diversity for Crossover and mutation, $M_{type} = 2, R_{rot} = 0$

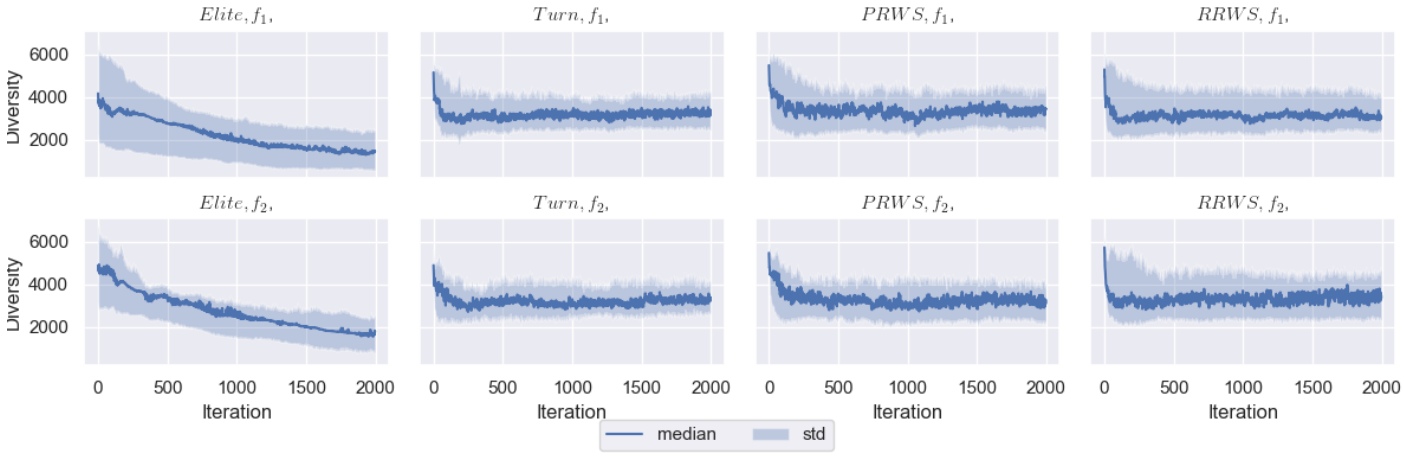


(a) Selection

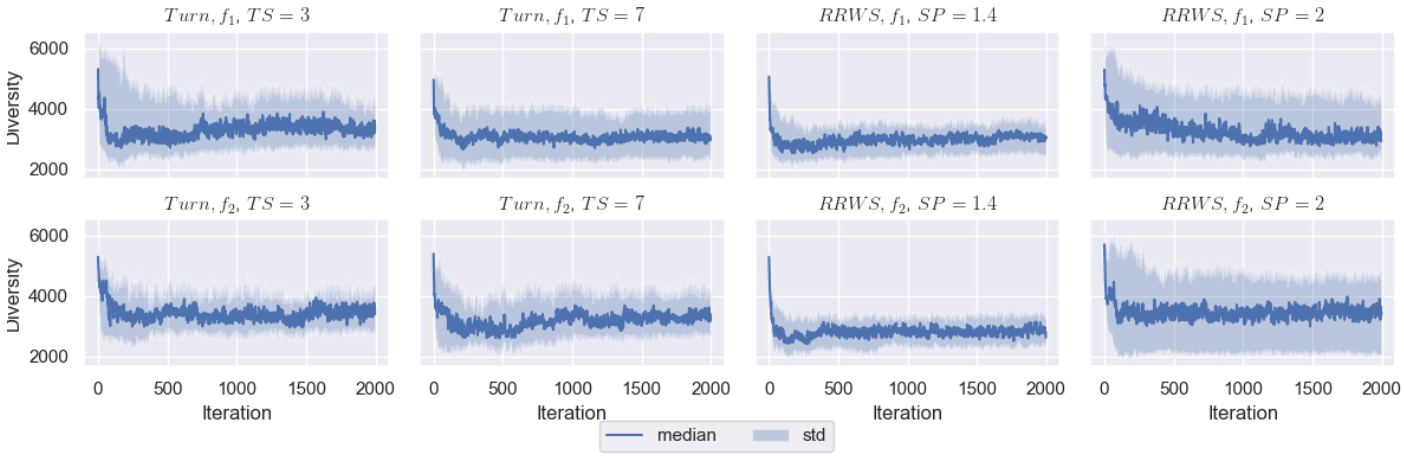


(b) Selection Pressure

Figure A.13: Population distribution for Selection, $M_{type} = 2, R_{rot} = 1$

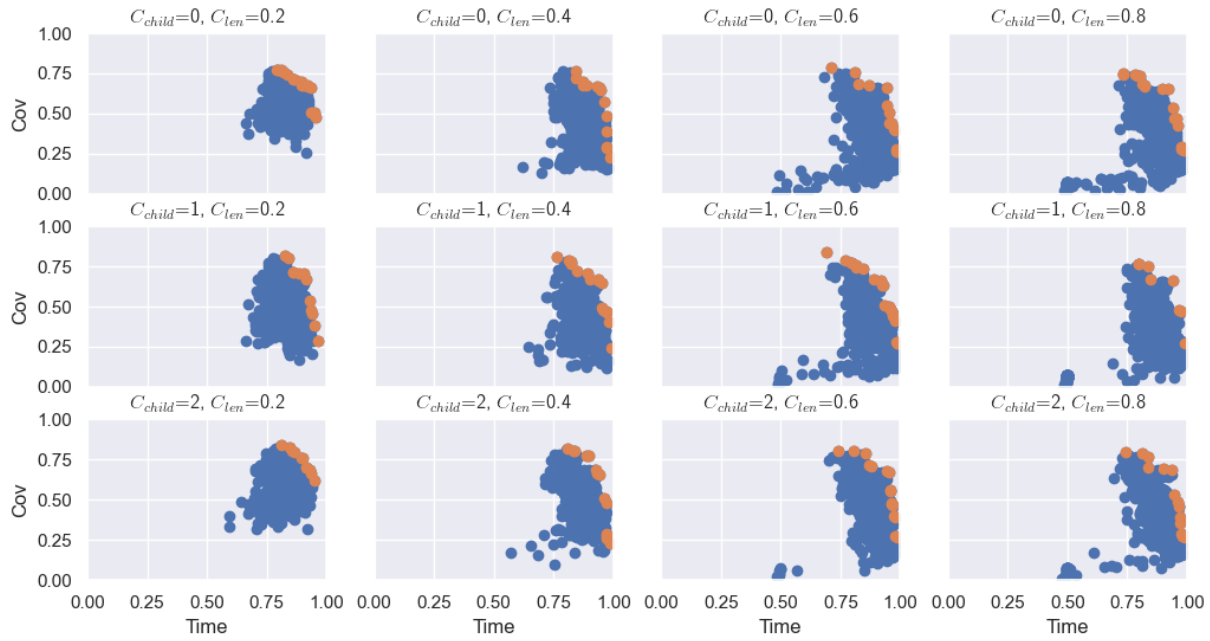


(a) Selection

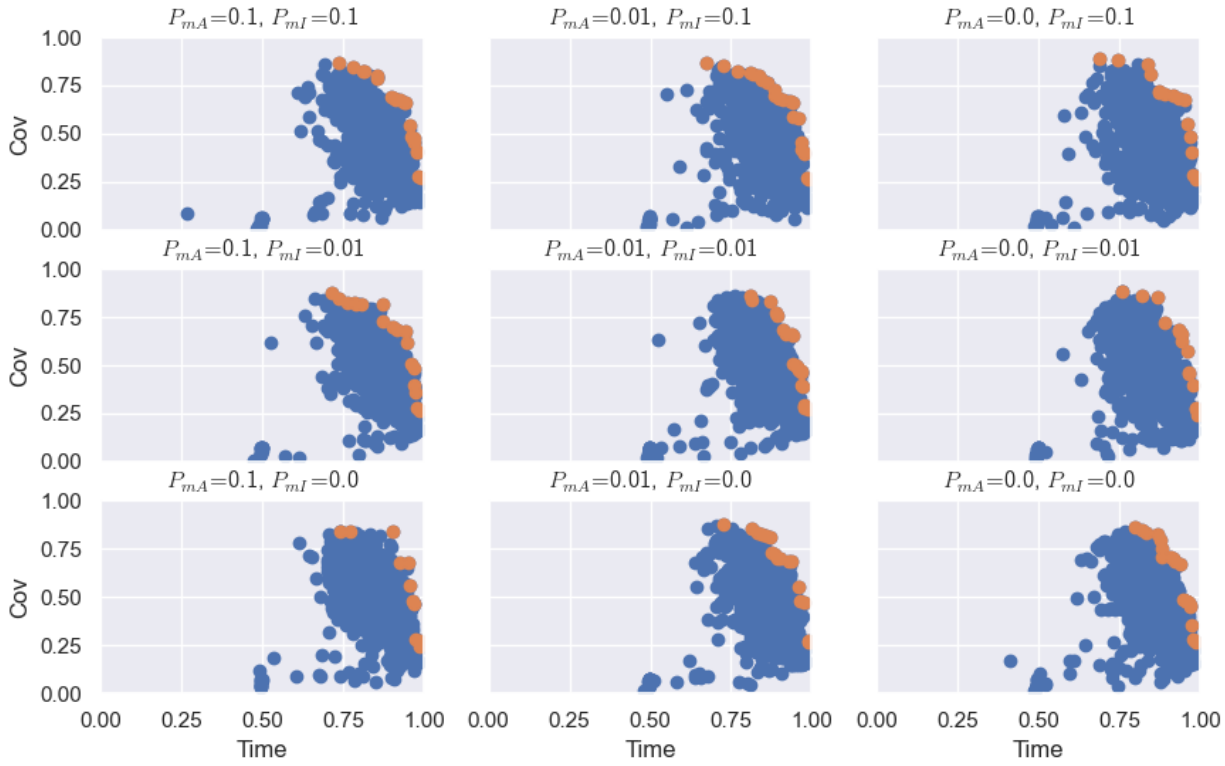


(b) Selection Pressure

Figure A.14: Diversity for Selection, $M_{type} = 2, R_{rot} = 1$

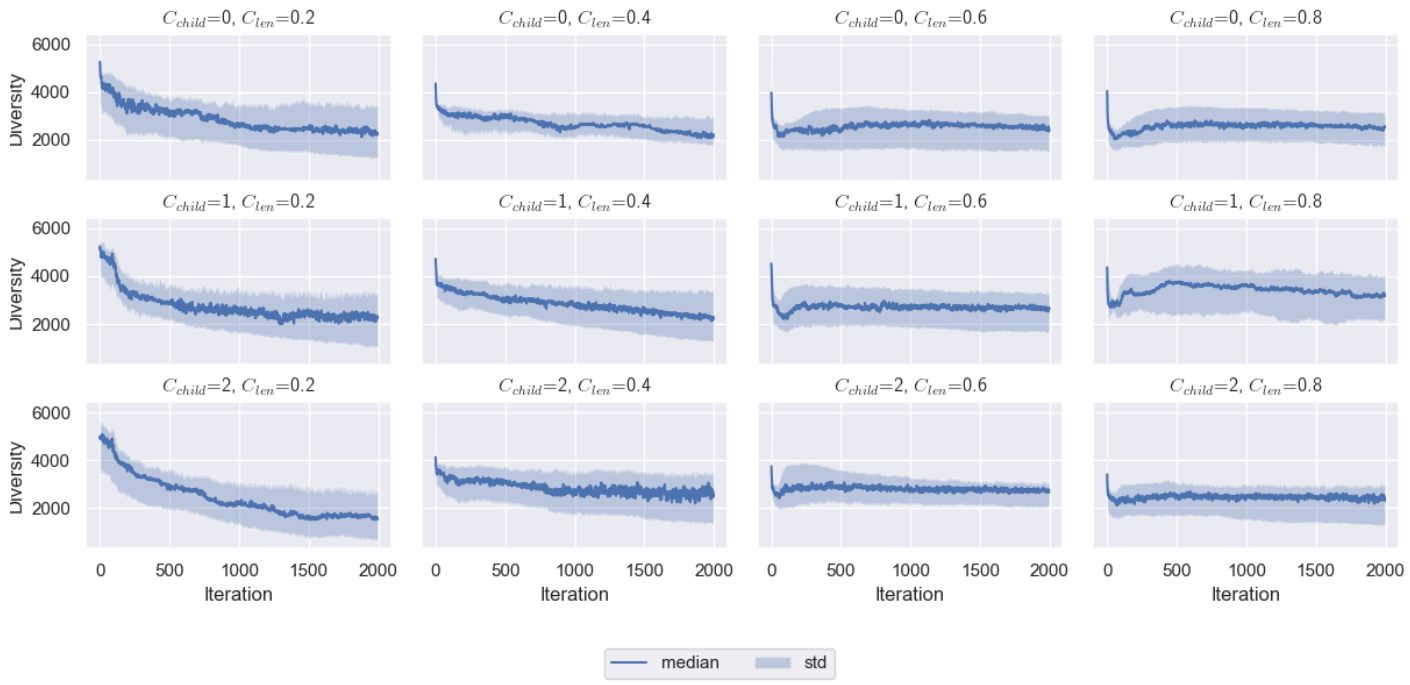


(a) Crossover

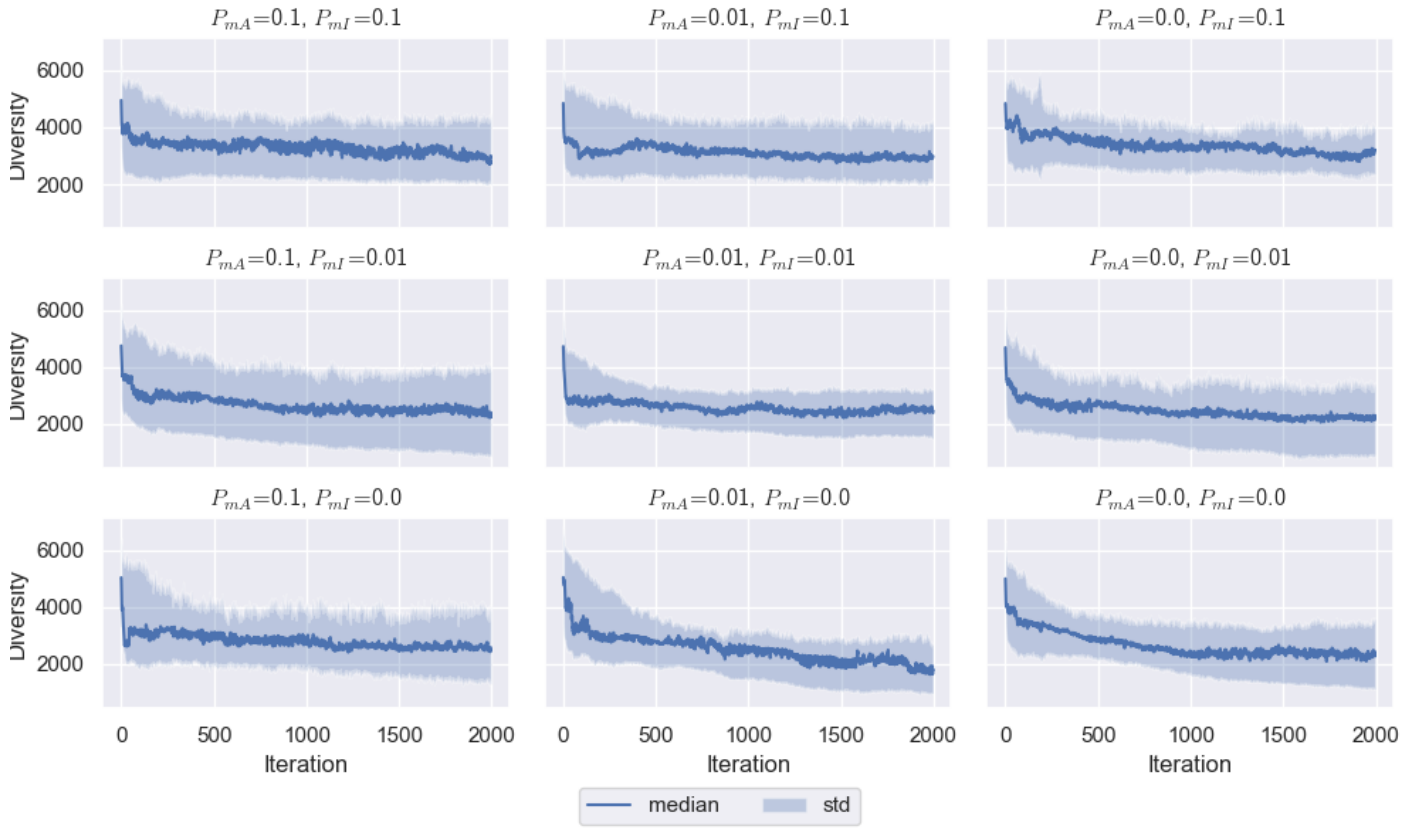


(b) Mutation

Figure A.15: Population distribution for crossover and mutation, $M_{type} = 2$, $R_{rot} = 1$



(a) Crossover



(b) Mutation

Figure A.16: Diversity for Crossover and mutation, $M_{type} = 2, R_{rot} = 1$

A.1.2 Path generation

A.2 Best Configurations selected for retraining

Table A.1: Best configurations generated for $M_{type} = 1$

FigureA.17	R_{rot}	$f_{obstacle}$	Selection	$keep$	$select$	TS	SP	C_{len}	C_{child}	P_{mA}	P_{mI}	cov_{final}	t_{final}
(a)	0	f_1	Elite	0	0	0	2	0.6	0	0.01	0.1	0.810864	0.889973
(b)	0	f_1	Turn	10	20	3	2	0.4	0	0.1	0.1	0.743704	0.963532
(c)	0	f_1	PRWS	10	20	0	0	0.1	0	0.0	0.0	0.827654	0.794313
(d)	0	f_1	RRWS	10	20	0	2	0.1	0	0.0	0.01	0.768395	0.825027
(e)	0	f_2	Elite	0	0	0	0	0.1	2	0.01	0.0	0.858765	0.805839
(f)	0	f_2	Turn	10	20	7	0	0.3	2	0.1	0.0	0.784198	0.852389
(g)	0	f_2	PRWS	10	20	0	0	0.1	0	0.0	0.01	0.814815	0.818859
(h)	0	f_2	RRWS	10	20	0	2	0.1	2	0.0	0.0	0.777778	0.795455
(i)	6.28	f_1	Elite	0	0	0	0	0.3	2	0.1	0.0	0.874074	0.776625
(j)	6.28	f_1	Turn	10	20	7	2	0.2	0	0.01	0.1	0.822222	0.926432
(k)	6.28	f_1	PRWS	10	10	0	0	0.3	0	0.0	0.1	0.692346	0.897165
(l)	6.28	f_1	RRWS	10	20	0	2	0.1	0	0.01	0.01	0.824691	0.799912
(m)	6.28	f_2	Elite	0	0	0	0	0.1	2	0.1	0.0	0.84642	0.799684
(n)	6.28	f_2	Turn	10	20	3	0	0.1	1	0.01	0.01	0.813333	0.782359
(o)	6.28	f_2	PRWS	10	20	0	0	0.1	0	0.1	0.01	0.809383	0.810188
(p)	6.28	f_2	RRWS	10	20	0	2	0.1	2	0.1	0.01	0.781235	0.791927

Table A.2: Best configurations generated for $M_{type} = 2$

FigureA.18	R_{rot}	$f_{obstacle}$	Selection	$keep$	$select$	TS	SP	C_{len}	C_{child}	P_{mA}	P_{mI}	cov_{final}	t_{final}
(a)	0	f_1	Elite	0	0	0	0	0.1	2	0.0	0.0	0.866279	0.817783
(b)	0	f_1	Turn	10	20	7	0	0.1	0	0.1	0.1	0.819186	0.876244
(c)	0	f_1	PRWS	10	20	0	0	0.1	2	0.0	0.01	0.86686	0.846678
(d)	0	f_1	RRWS	10	10	0	2	0.1	0	0.01	0.0	0.836628	0.78249
(e)	0	f_2	Elite	0	0	0	0	0.1	2	0.0	0.0	0.861628	0.830717
(f)	0	f_2	Turn	1	20	7	0	0.1	0	0.01	0.0	0.796512	0.875399
(g)	0	f_2	PRWS	10	20	0	0	0.1	2	0.0	0.1	0.843605	0.783901
(h)	0	f_2	RRWS	10	20	0	2	0.1	2	0.0	0.01	0.830233	0.787211
(i)	6.28	f_1	Elite	0	0	0	0	0.1	2	0.1	0.0	0.819767	0.864434
(j)	6.28	f_1	Turn	10	20	7	0	0.1	2	0.0	0.1	0.845349	0.830881
(k)	6.28	f_1	PRWS	10	10	0	0	0.1	0	0.1	0.1	0.794186	0.853525
(l)	6.28	f_1	RRWS	10	20	0	2	0.1	0	0.0	0.0	0.773256	0.866285
(m)	6.28	f_2	Elite	0	0	0	0	0.1	2	0.0	0.01	0.853488	0.868523
(n)	6.28	f_2	Turn	10	20	7	0	0.1	2	0.0	0.1	0.859884	0.837755
(o)	6.28	f_2	PRWS	10	10	0	0	0.1	2	0.0	0.01	0.85814	0.817946
(p)	6.28	f_2	RRWS	10	20	0	2	0.1	0	0.0	0.0	0.82093	0.779031

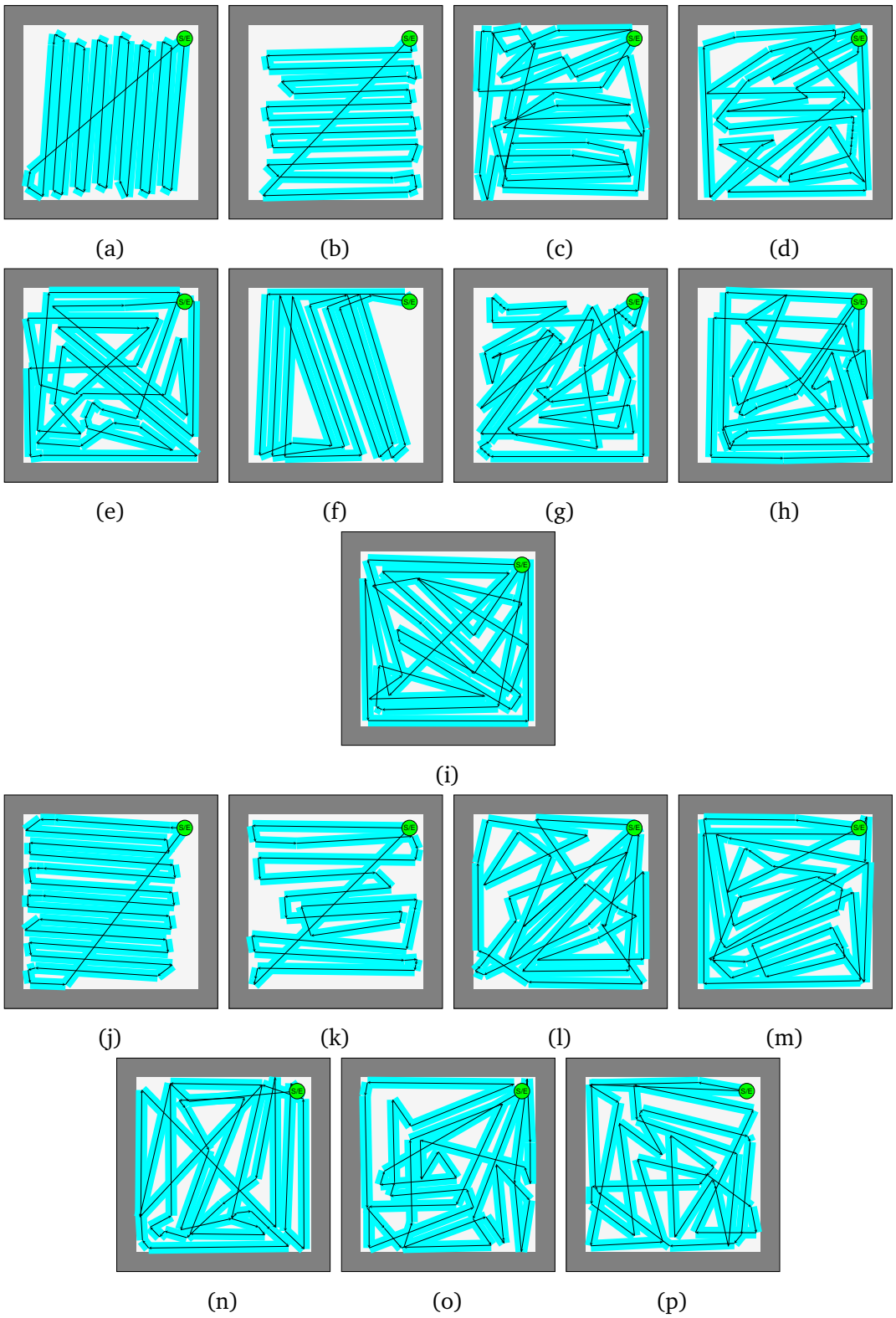


Figure A.17: Best Paths generated for $M_{type} = 1$. The corresponding parameter setting is listed in TableA.1

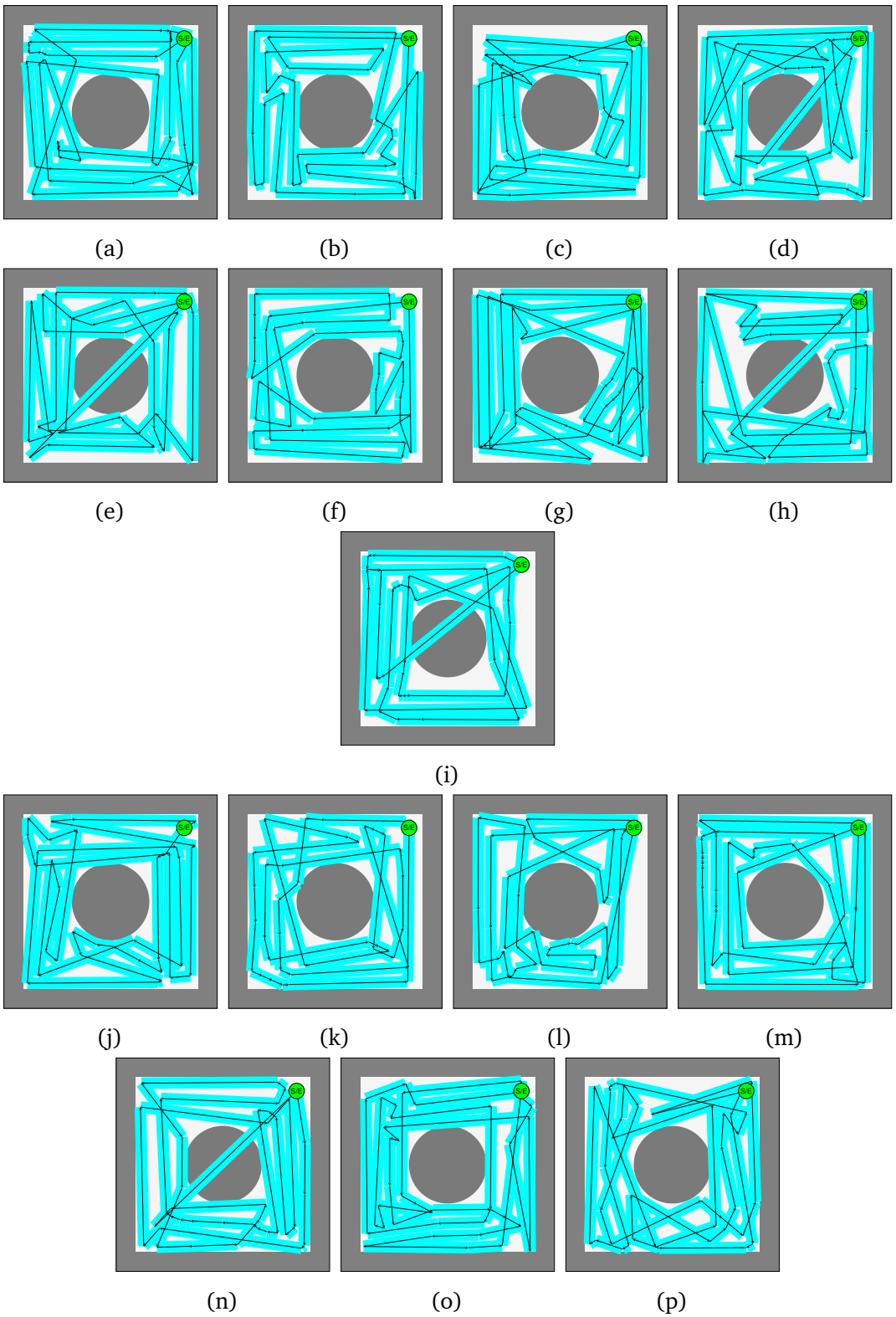


Figure A.18: Best Paths generated for $M_{type} = 2$. The corresponding parameter setting is listed in TableA.2

A.3 Retraining

Table A.3: All retrained populations for $M_{type} = 1$

FigureA.19	R_{rot}	$f_{obstacle}$	Selection	$keep$	$select$	TS	SP	C_{len}	C_{child}	P_{mA}	P_{mI}	cov_{final}	t_{final}
(a)	0	f_1	Elite	0	0	0	2	0.6	0	0.01	0.1	0.625	0.872383
(b)	0	f_1	Turn	10	20	3	2	0.4	0	0.1	0.1	0.558571	0.861233
(c)	0	f_1	PRWS	10	20	0	0	0.1	0	0.0	0.0	0.812143	0.785221
(d)	0	f_1	RRWS	10	20	0	2	0.1	0	0.0	0.01	0.852857	0.704841
(e)	0	f_2	Elite	0	0	0	0	0.1	2	0.01	0.0	0.906429	0.806226
(f)	0	f_2	Turn	10	20	7	0	0.3	2	0.1	0.0	0.797143	0.718147
(g)	0	f_2	PRWS	10	20	0	0	0.1	0	0.0	0.01	0.84	0.807138
(h)	0	f_2	RRWS	10	20	0	2	0.1	2	0.0	0.0	0.84	0.695035
(i)	6.28	f_1	Elite	0	0	0	0	0.3	2	0.1	0.0	0.843571	0.804334
(j)	6.28	f_1	Turn	10	20	7	2	0.2	0	0.01	0.1	0.812143	0.897921
(k)	6.28	f_1	PRWS	10	10	0	0	0.3	0	0.0	0.1	0.531429	0.928226
(l)	6.28	f_1	RRWS	10	20	0	2	0.1	0	0.01	0.01	0.884286	0.612448
(m)	6.28	f_2	Elite	0	0	0	0	0.1	2	0.1	0.0	0.862857	0.808757
(n)	6.28	f_2	Turn	10	20	3	0	0.1	1	0.01	0.01	0.799286	0.757339
(o)	6.28	f_2	PRWS	10	20	0	0	0.1	0	0.1	0.01	0.892143	0.753495
(p)	6.28	f_2	RRWS	10	20	0	2	0.1	2	0.1	0.01	0.83	0.732657

Table A.4: Best configurations generated for $M_{type} = 2$

FigureA.20	R_{rot}	$f_{obstacle}$	Selection	$keep$	$select$	TS	SP	C_{len}	C_{child}	P_{mA}	P_{mI}	cov_{final}	t_{final}
(a)	0	f_1	Elite	0	0	0	0	0.1	2	0.0	0.0	0.902857	0.751359
(b)	0	f_1	Turn	10	20	7	0	0.1	0	0.1	0.1	0.862041	0.77306
(c)	0	f_1	PRWS	10	20	0	0	0.1	2	0.0	0.01	0.883265	0.755059
(d)	0	f_1	RRWS	10	10	0	2	0.1	0	0.01	0.0	0.846531	0.722145
(e)	0	f_2	Elite	0	0	0	0	0.1	2	0.0	0.0	0.850612	0.8336
(f)	0	f_2	Turn	1	20	7	0	0.1	0	0.01	0.0	0.817959	0.754518
(g)	0	f_2	PRWS	10	20	0	0	0.1	2	0.0	0.1	0.862857	0.677564
(h)	0	f_2	RRWS	10	20	0	2	0.1	2	0.0	0.01	0.859592	0.691399
(i)	6.28	f_1	Elite	0	0	0	0	0.1	2	0.1	0.0	0.86449	0.819004
(j)	6.28	f_1	Turn	10	20	7	0	0.1	2	0.0	0.1	0.89551	0.696545
(k)	6.28	f_1	PRWS	10	10	0	0	0.1	0	0.1	0.1	0.81551	0.61372
(l)	6.28	f_1	RRWS	10	20	0	2	0.1	0	0.0	0.0	0.820408	0.700505
(m)	6.28	f_2	Elite	0	0	0	0	0.1	2	0.0	0.01	0.828571	0.718359
(n)	6.28	f_2	Turn	10	20	7	0	0.1	2	0.0	0.1	0.900408	0.595374
(o)	6.28	f_2	PRWS	10	10	0	0	0.1	2	0.0	0.01	0.861225	0.711546
(p)	6.28	f_2	RRWS	10	20	0	2	0.1	0	0.0	0.0	0.835102	0.74824

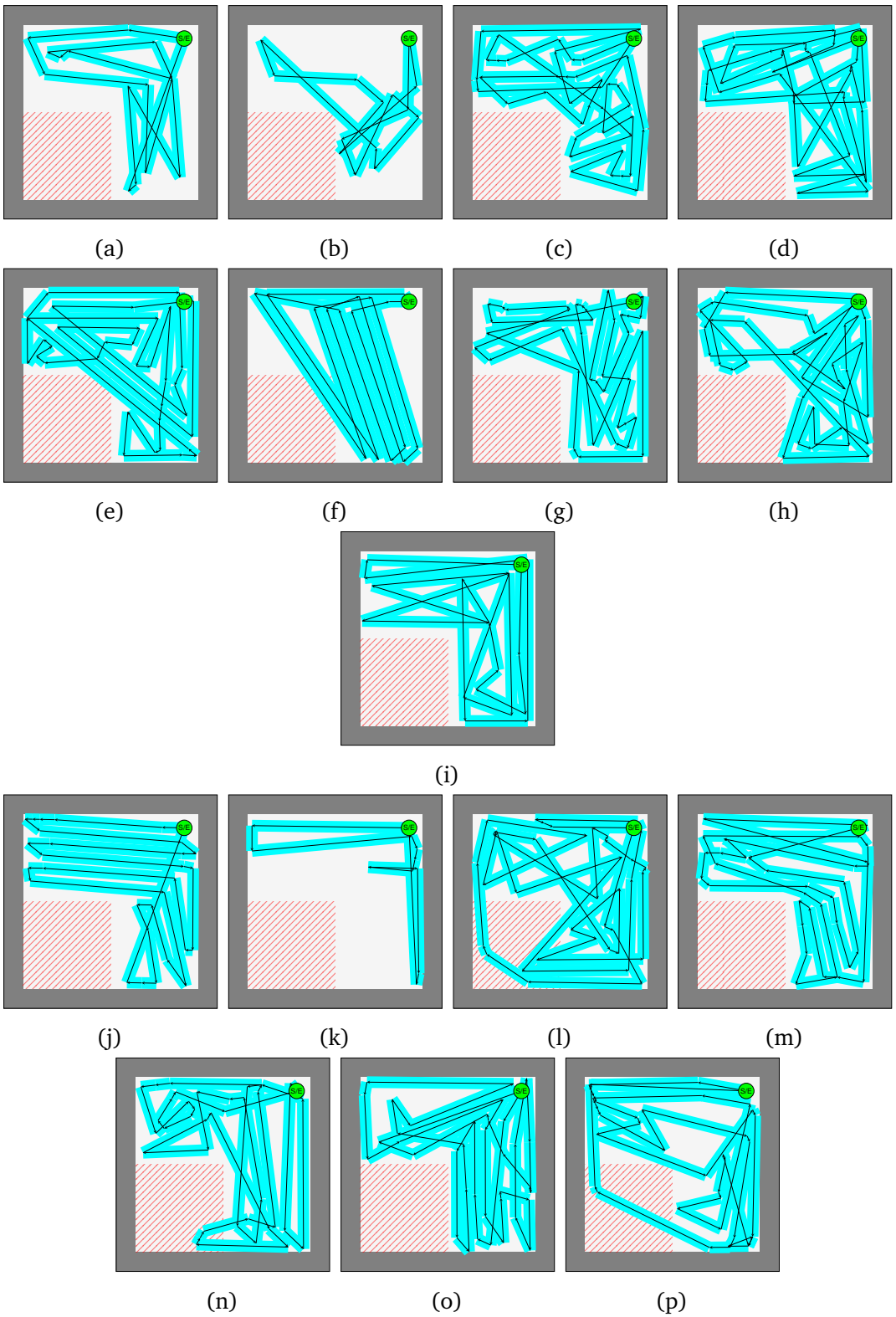


Figure A.19: Paths generated by retrain procedure for $M_{type} = 1$. The corresponding parameter setting is listed in TableA.3

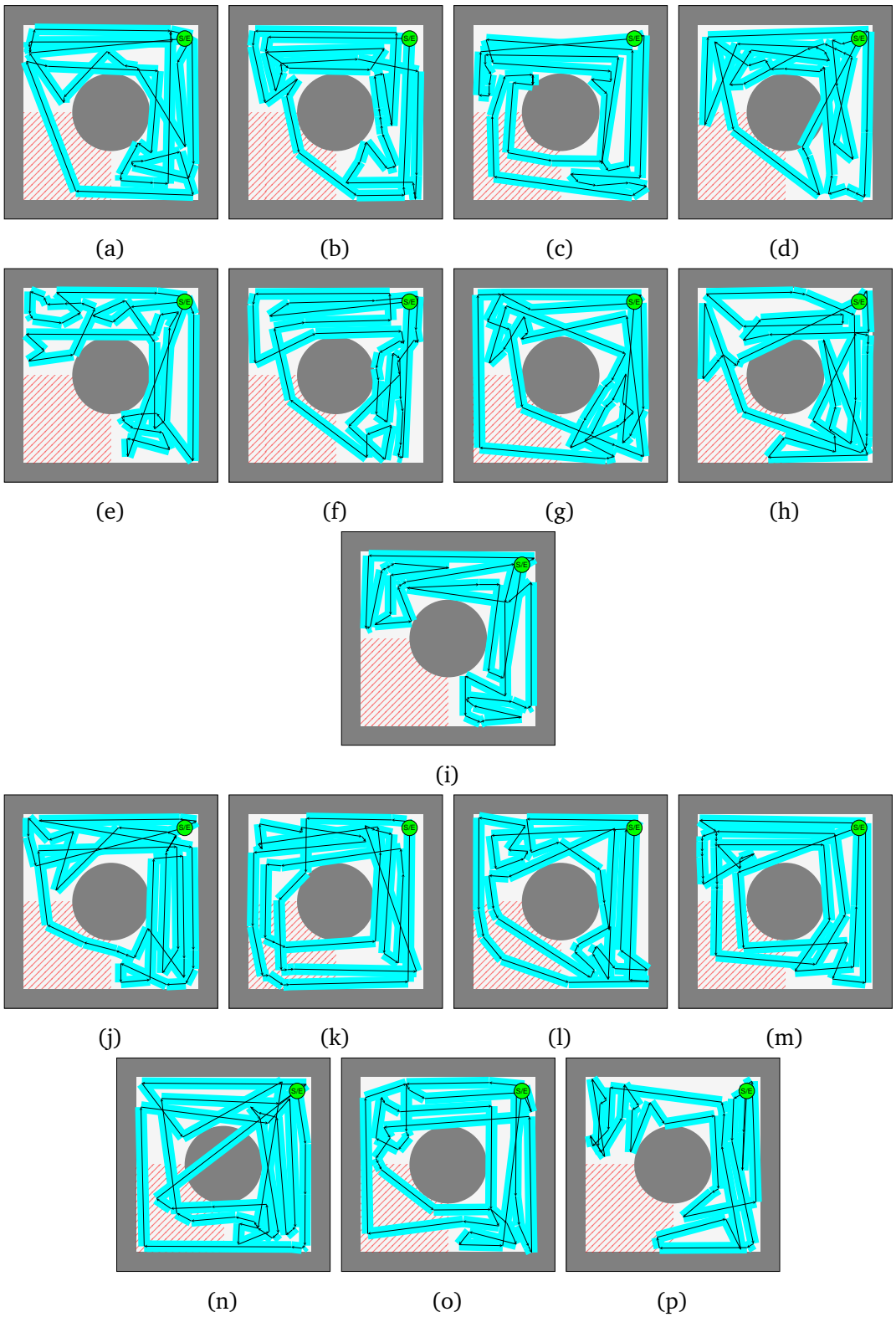


Figure A.20: Paths generated by retrain procedure for $M_{type} = 2$. The corresponding parameter setting is listed in TableA.4