Extending the Scalable Multi-Agent Pathfinding Problem to Modifiable Environments

Carlo Nübel[1],Malte Speidel[1], and Sanaz Mostaghim[1,2]
[1]Faculty of Computer Science, Otto von Guericke University Magdeburg, Germany
[2]Fraunhofer IVI, Dresden, Germany
{carlo.nuebel, malte.speidel, sanaz.mostaghim}@ovgu.de

***Abstract*—In this paper, we extend the scalable multi-agent pathfinding problem to modifiable environments. Instead of only finding collision-free paths, agents collaboratively alter the environment to transform it to a predefined target state. Applications of this problem include smart warehousing, terrain leveling, autonomous mining, and automated construction. We introduce a scalable simulation environment for such tasks and propose an evolutionary optimization approach. Two experiments are conducted: (1) analyzing the effect of the number of agents on task performance and (2) comparing the performance of different crossover operators. We propose a new two-level crossover operator acting on different layers of the gene encoding and compare it with two one-level operators. Results indicate that more agents reduce the task completion time, but fewer agents yield more consistent solutions with better objective values due to the collision constraints. Moreover, the introduced two-level crossover operator achieves significantly better objective values than both one-level operators in most problem instances.**

***Index Terms*—Crossover Operators, Evolutionary Algorithms, Modifiable Environments, Multi-Agent Pathfinding,**

I. Introduction

The problem of multi-agent pathfinding (MAPF) finds application in many domains, such as robotics, autonomous navigation and transportation systems, and games. In these domains, multiple agents, for example, robots or drones, navigate a shared environment and fulfill their designated tasks within, while avoiding collisions with the other agents. The tasks of the agents in these environments can be as simple as reaching a specific location (navigation), picking up and dropping off specific items within the environment (warehouse logistics), or altering the environment (terrain leveling). Many solutions to the MAPF problem involve the agents communicating with each other to avoid collisions and divide the tasks. This communication can be challenging and comes with many problems. If individual robots fail to communicate correctly, this can lead to collisions of the agents or, in general, unwanted and potentially dangerous behavior. In the first part of this paper, we introduce the problem of multi-agent navigation in modifiable environments, where the agents have to alter their surroundings in a specific way while avoiding collisions, which is comparable to robots moving shelves in a warehouse. We propose to approach this adaptation of the MAPF problem offline by globally optimizing the paths and actions of the agents before the agents start to navigate through the environment, with the use of evolutionary algorithms (EAs). This avoids the need for communication between the agents and the risks and challenges that come with it. Secondly, we evaluate new, problem-specific crossover operators on this optimization task. In EAs, the crossover operator creates new individuals by recombining the genes of two or more parent individuals. This can lead to a more diverse population by exploring the search space more thoroughly. Next to the mutation operator, the crossover operator is a key to the success of an EA in finding optimal solutions. Therefore, it is important to develop problem-specific crossover operators when trying to solve new problems with the evolutionary approach. The main contributions of this paper are threefold: first, we introduce a simulation environment for the scalable multi-agent navigation problem in modifiable environments. Second, a solution approach based on evolutionary single-objective optimization is formulated and applied to the problem. And third, we compare a novel two-level crossover operator with two different one-level crossover operators on the introduced problem, including an ablation study. With these contributions, we aim to answer two questions. Firstly, how does the number of agents used simultaneously influence the problem-solving capabilities of the evolutionary approach on the modification task? And secondly, what is the influence of the proposed crossover operators on the optimization process?

II. Related Work

Pathfinding has been extensively studied, with the shortest path problem being one of the most prominent. In it, agents are tasked with finding the shortest possible route between two or more points in an environment. This problem has been extended to optimize multiple objectives simultaneously and use multiple agents to do so [1]–[3]. Many solution approaches to these problem variations of the shortest path problem have been introduced in the past, as summarized in [4]. The multi-agent pathfinding problem involves multiple agents navigating a shared environment while avoiding collisions and finding the shortest path between waypoints [5]. In addition to the multi-agent pathfinding and the shortest path problem, other closely related problems have been formulated as well, such as the pick-up and delivery problem (PD). It finds application, e.g., in warehouse logistics, where fleets of robots navigate the warehouse to collect and deliver the necessary items, boxes, or shelves [6], [7]. In [8], the authors extended the multi-agent PD problem to allow robots to alter the layout of the environment so that the PD task can be fulfilled more efficiently. Another

study extends MAPF to configurable environments, where agents must find the shortest collision-free path across different environments [9].

In contrast to the aforementioned tasks of the agents and their respective environments' characteristics, we propose to extend MAPF to modifiable environments. Here, agents themselves must actively alter the environment in specific ways without colliding to fulfill the task. To the best of our knowledge, this has not been studied before, and no comparable benchmark problems in the domain of MAPF exist. Practical applications of this new problem include smart warehousing, where shelves or crates have to be rearranged in limited space, or terrain leveling and renaturation, where soil has to be removed from and moved to specific locations. In all of these examples, the goal is to change the environment in a specific way. In addition to introducing a simulation environment for multi-agent navigation in modifiable environments, we present a solution approach based on evolutionary optimization and compare the performance of different crossover operators. EAs have proven to be capable of solving a wide range of pathfinding problems [10], [11]. The crossover operator in an EA generates new solutions by recombining genes from two or more parent solutions. It is argued to be one of the most important aspects of an EA [12]. Many crossover operators have been introduced in the past, such as a one-point crossover, uniform crossover, or partially mapped crossover, to name a few. For a detailed explanation of different crossover operators, see [13]. With our proposed encoding of an individual in the population of an EA, we have two different levels in one gene sequence: the agents themselves and the paths and action sequences of the different agents within one individual. The crossover operator can be applied to these different levels.

III. Multi-Agent Navigation in Modifiable Environments

In the multi-agent navigation problem in modifiable environments, agents navigate through a shared environment. We model the environment as a scalable grid world. This environment is filled with randomly placed, movable obstacles. The goal is to change the environment from a random obstacle distribution to a certain target distribution of obstacles.

We use the terms *environment* and *map* as synonyms throughout this paper. Each random start map has the same number of obstacles as the target map to ensure that the task is always solvable. The task is considered complete once the environment has been transformed into the desired shape, without regard to the agents' final positions. The agents use the Von-Neumann neighborhood to traverse the environment. They have the following three options to interact with the obstacles. (1) They can pick up an obstacle from their current location, (2) drop off an obstacle at their current position, or (3) navigate through the environment while carrying or not carrying an obstacle. This enables the agents to redistribute the obstacles, modifying their environment, as shown in Figure 1.

There are a number of limitations as to how the agents can interact with the obstacles. If an agent is already carrying an obstacle, or if there is no obstacle at the current position of the agent, it cannot pick up an obstacle. Moreover, an agent can only place an obstacle on a free cell in the environment, and only if that agent is currently carrying an obstacle. Instead of each agent having complete knowledge of the target environment, it only has knowledge about whether there should be an obstacle at its current position or not. Therefore, the agent only picks up an obstacle from a cell if that cell has to be obstacle-free and only drops off an obstacle to a cell if there has to be an obstacle at that cell. Scenarios in which multiple agents try to pick up the same obstacle or drop off obstacles to the same cell at the same time are handled by the collision constraint, making these solutions infeasible. We consider three different target environment types: the river, the checkerboard, and the circular environment, as shown in Figure 2. These environments are encoded as scalable, quadratic grid worlds. The target environments are designed to be of different difficulties to determine how each problem is solved by our solution approach. Each cell in the environment can hold a value of 1, representing an obstacle on that cell (black cells), or a 0, representing an empty cell (white cells). There cannot be more than one obstacle in each cell. These maps are deliberately simplified and are not intended to reflect real-world conditions, but rather to enable a controlled evaluation of the proposed approach. The starting positions of the agents are determined based on the total number of agents. If there is only a single agent, it is positioned at the center of the map's height on the far-left side. If there is more than one agent, we only consider even numbers of agents. Half of the agents are evenly spaced along the left side of the map, and the other half are symmetrically placed on the right side. Our framework for this problem is scalable in the number of agents and in the size of the environment.

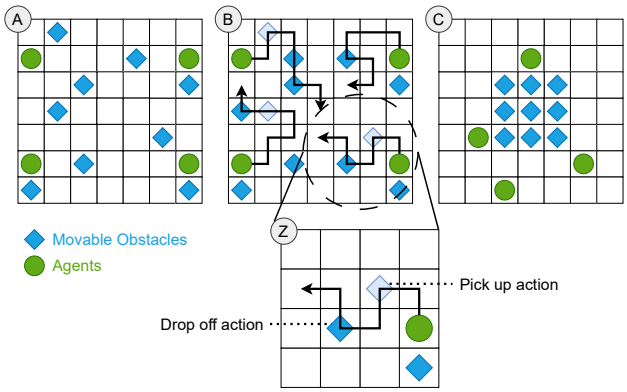

Fig. 1. Problem Description. The faded blue obstacles indicate previous positions of obstacles. A: Start with randomly distributed obstacles; B: Agents interact with the environment and move around obstacles; C: Target state. Z: Zoom in on one agent picking up and dropping off an obstacle.

IV. Optimization Approach

Evolutionary algorithms optimize a population of solutions to a problem iteratively over generations using selection mechanisms, genetic operators, and objective functions [14]. In this work, we used `pymoo` [15] to apply this optimization approach

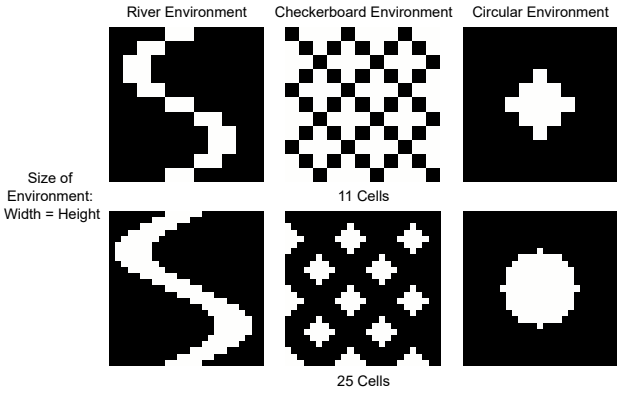

Fig. 2. Target environments with 3 different patterns in two different sizes.

to our problem. Therefore, an encoding for the solutions to our problem is developed, and the objectives and constraints are defined. We also describe the sampling and mutation operators and focus on describing the new two-level crossover operator.

Encoding: An individual X is a list of N agents:

$$X = [agent_1, agent_2, ..., agent_N] \tag{1}$$

each agent follows a path of length M. The paths' positions as well as the interactions with obstacles are encoded at discrete time steps. An agent is represented as a sequence of genes g_m for $m \in [0, ..., M]$ where each gene encodes the agent's current position $[x_m, y_m]$ and the action a_m performed at that position. The possible actions are encoded as: $a_m = 1$: pick up an obstacle, $a_m = -1$: drop off an obstacle, and $a_m = 0$: no interaction with obstacles. Therefore, a single gene is encoded as a list.

$$g_m = [[x_m, y_m], a_m] \tag{2}$$

Each agent's path must fulfill two conditions: first, the sequence of genes defines a fully connected path, meaning that consecutive positions $[x_m, y_m]$ and $[x_{m+1}, y_{m+1}]$ must either be adjacent cells in the environment (moving up, down, left, or right) or the same cell, as we allow agents to stay in the same position over multiple time steps. Secondly, each agent follows a path with a fixed length of M steps. As an alternative to this encoding, it is also possible to save the sequence of movement directions, which would be more memory efficient. However, this representation would increase the computational cost of collision detection, as the agents' positions would first need to be reconstructed from the direction sequences.

Map Difference Objective: During the optimization process, we minimize the objective *map difference*. It describes the difference between a resulting map R, which is the start map after the agents of an individual traversed and redistributed the obstacles on it, and the target map T with $T, R \in \{0, 1\}^{H \times W}$. $T_{x,y}$ and $R_{x,y}$ are 1 if there is an obstacle on (x, y), otherwise 0. We calculate the map difference as the absolute difference between the two maps normalized with the map dimensions H and W as shown in Equation 3.

$$f(R) = \frac{1}{HW} \sum_{x=1}^{H} \sum_{y=1}^{W} |R_{x,y} - T_{x,y}| \tag{3}$$

Collision Constraint: A collision can occur in two different ways. First, if two different agents i and j of one individual are at the same position on the map at the same time as defined in the collision condition in Equation 4.

$$[x_{i,m}, y_{i,m}] = [x_{j,m}, y_{j,m}], \text{ for } i \neq j \tag{4}$$

Therefore, collision-free individuals must fulfill the following condition as defined in 5.

$$\begin{aligned} &\forall i, j \in \{1, ..., N\}, \forall m \in \{1, ..., M\}, \\ &i \neq j \Rightarrow [x_{i,m}, y_{i,m}] \neq [x_{j,m}, y_{j,m}] \end{aligned} \tag{5}$$

Second, a collision also occurs if two neighboring agents swap positions in the next time step. The swap collision condition for two agents' paths p_i and p_j is defined in Equation 6.

$$p_i(m) = p_j(m+1) \quad \wedge \quad p_j(m) = p_i(m+1) \tag{6}$$

An individual is only feasible if it has a constraint violation of zero, meaning there are no collisions.

Sampling: We use polygonal chains to sample the initial paths for each agent in each individual. From the fixed starting position, we sample a random cell anywhere in the environment and connect the start point to the sampled cell using the shortest path between the two points. Once this connection is made, we sample another random cell and extend the path by connecting it to the previous endpoint, again using the shortest path. This is done until the desired path length is reached. If the total path length exceeds the allowed path length, we truncate the path at the end. Each cell in the path also requires an action to be carried out at that cell, either 0, 1, or -1. For each action in this action sequence, we define a fixed probability p of the action being an interacting action (pick up or drop off), and with the probability $1 - p$, the action will be an idle action, which does not interact with the environment. To avoid creating invalid actions, we decide on the next interacting action based on the previous interacting action, meaning if the last interacting action was a pickup, the next will be a drop-off action, and vice versa.

Mutation: First, we randomly determine how many of the agents within one individual are being mutated. At least one agent and, at maximum, all agents of one individual will be mutated. The agents to be mutated are then selected based on how many invalid actions they try to carry out. This way it is more likely that worse agents are mutated. For each of the selected agents, we choose with equal probabilities whether only the path, only the action sequence, or both are being mutated. If both are to be mutated, we first carry out the path mutation followed by the action sequence mutation.

Path Mutation: The path mutation utilized in this optimization process is based on the perimeter mutation proposed in [16] and is adapted to fit our problem description. The path of the agent is split at two distinct cutoff points. A random point is then sampled anywhere in the environment. Both cutoff points are then reconnected to the sampled point using the shortest path. Since we use a fixed-length encoding for our individuals, this approach can result in paths that are either too short or too long. If a path exceeds the desired length, it

is truncated at the end. If it is too short, a randomly generated path segment is appended to the end of the original path to maintain the desired path length. The shortest paths connecting the cutoff points to the randomly sampled point, as well as the random path segment added to the end of too-short paths, also require action sequences. Therefore, for each gene, an action is appended to each position in the path in the same way that it is done during the initial sampling.

Action Mutation: To alter the action sequence of an agent, we first relocate non-idle actions. First, a non-idle action is chosen at random. The number of idle actions before and after this position is then counted, and the direction with the higher count is selected as the target region. Within this region, one idle action is randomly chosen and swapped with the initially selected action. However, with this approach, the overall interacting action count is maintained. This can be a problem when there are either too many or not enough interacting actions in the sequence. To avoid this, additionally, random indices are sampled, and the actions in the action sequence at these indices are exchanged at random. Idle actions are exchanged with interacting actions, and vice versa. This way, mutating an action sequence can also change the total number of idle and interacting actions and not just switch actions around that are already in the sequence, leading to a better exploration of the search space.

Crossover Operator: The micro crossover exchanges segments of paths and action sequences between corresponding agents of the parents, whereas the macro crossover swaps entire agents between individuals. The mixed crossover combines both micro and macro crossover operations.

Micro Crossover: To exchange the paths of the agents within the parent individuals, we use a one-point crossover, as shown in Figure 3. The two parent parts are cut at a random cutoff point. These cutoff points are then connected using the shortest path between the two points (dotted line). The first part of each child's path stems from the first parent, and the second part from the second parent, as shown on the right in Figure 3. This crossover is done for one agent of the parents. This agent is picked at random. The indices of the agents whose paths are exchanged must be the same because the agents must have the same starting positions to avoid invalid paths. This means that if, for example, the second agent from the first parent is selected, then also the second agent from the second parent individual is selected.

Macro Crossover: For the macro crossover operator, we also use a one-point crossover. This time, however, the crossover is not operating on the paths of the agents but on the agent sequence of two parent individuals. A random cutoff point is selected, and the agents to the right of that cutoff point are exchanged between the individuals.

Mixed Crossover: To mix the described crossover operators, we simply carry out one after the other. The order in which this is done can be ignored, as both would result in the same children because they are operating on two different parts of the encoding and do not influence each other. Therefore, in the mixed crossover operator, we first perform the micro crossover followed by the macro crossover.

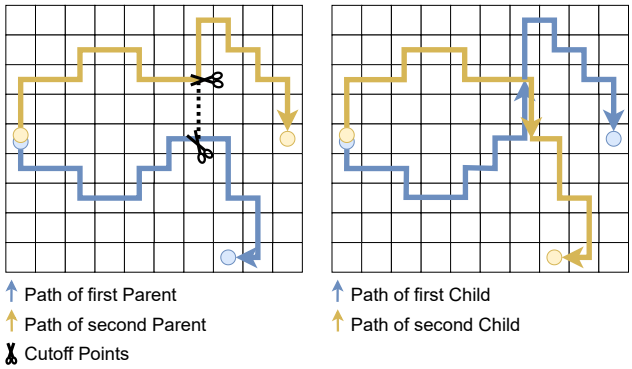

Fig. 3. Micro crossover operator.

V. Experiments and Results

In the first experiment, we test 1, 2, 4, 6, 8, and 10 agents on the six problem instances (as previously shown in Figure 2), using the micro crossover operator. In the second experiment, the algorithm using the three different crossover operators is run on the six problem instances with 10 agents. For the ablation study, the algorithm is also run without any crossover operator. This serves as a benchmark to compare the proposed operators against. For all experiments, we used the following parameter settings: each population has a size of 200 individuals. We use a fixed crossover probability of 80% and a mutation probability of 50%. Each run was terminated after 100,000 function evaluations. The crossover experiment is run with 31 different random seeds, whereas the multi-agent experiment is run with 10 different seeds. Lastly, each individual is given a step budget. This determines how many steps in total can be taken by all agents within one individual. We set this step budget to 5,000. Each agent in an individual with, for example, two agents would get 2,500 steps. This way, the agents of each individual use the same number of steps, making the results comparable. We use the genetic algorithm class from `pymoo` to run our experiments. This algorithm uses tournament parent selection and an environmental selection based on the constraint violation and the fitness value.

Multi-Agent Experiment: Figure 4 shows the average map differences over the ten seeds, the interquartile range (IQR), and outliers per number of agents across all three map types. On the left, this is shown for the smaller 11×11 environments, and on the right side for the 25×25 environments. On the 11×11 maps, the problem is always solved when using one, two, and four agents, which can be seen by the average map difference of zero. When using more agents than that, the task is not always solved; however, it is still solved in most cases. With larger environment sizes, this changes, and the task is never completely solved. We can conclude that lower numbers of agents seem to perform better here than higher numbers of agents for both map sizes. This is confirmed by performing a Tukey HSD test on our results, accounting for multiple comparisons by adjusting the p-values, to test for statistically significant differences. We found that

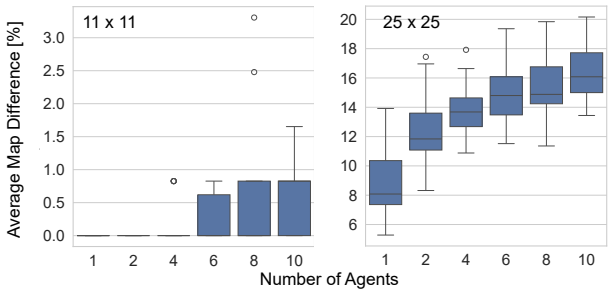

Fig. 4. Average map difference of the different numbers of agents.

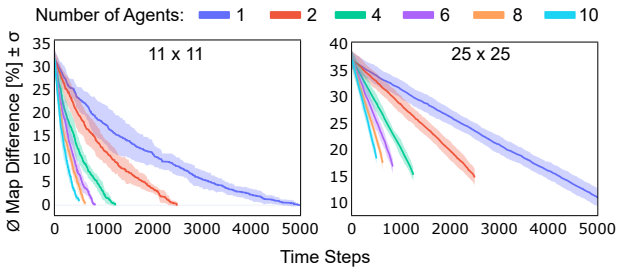

Fig. 5. Average map difference per timestep $\pm$ standard deviation σ on river environment for map sizes 11×11 and 25×25.

using 1, 2, and 4 agents on the 11×11 maps performs significantly better in terms of the objective than using 8 and 10 agents. Using 6 agents leads to a significantly better map difference than using 10 agents. Another Tukey HSD test was performed for the results of the experiments in the larger environments. The results show significantly better map differences for only using one agent compared to using more than one. Using two agents is significantly better than using six, eight, or ten agents, and using four agents is significantly better than using eight or ten agents. One reason for the worse objective values when using more agents is most likely due to the collision constraint. Individuals are only feasible if there are no collisions between the agents within the individual. Therefore, more feasible solutions are generated when fewer agents are on the map simultaneously and the search space is more thoroughly explored, leading to better solutions. Using multiple agents at the same time, however, also has advantages. When we look at the map difference per time step, we see that the map difference is decreasing more in fewer time steps when more agents are used than when fewer agents are used. In each time step, each agent can go to the next cell and interact with the obstacle there. Individuals with, for example, 10 agents can change 10 cells in one time step, whereas individuals with one agent can only change one cell. This average map difference per time step can be seen in Figure 5. On both small and large maps, the environment is modified faster if more agents are simultaneously modifying it. Finally, in Figure 6, we show the average map difference development per generation using the different numbers of agents in the small and large river environments. Again, it is visible that the maps are modified better in fewer generations and closer to the target environment if fewer agents per individual are used. Furthermore, the modification task is fulfilled more often

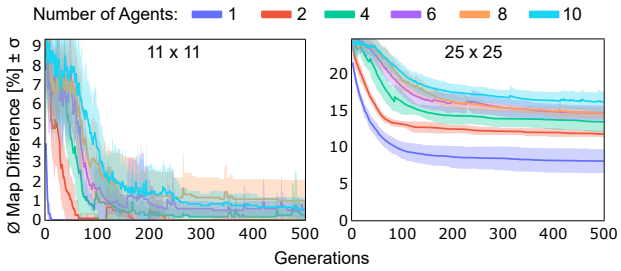

Fig. 6. Average map difference per generation $\pm$ standard deviation σ on the checkerboard environment using different numbers of agents.

on smaller maps and is not solved on the larger maps. This behavior can be seen with the other two target environments as well and is due to the larger search space on the larger maps. One way to solve the task on larger maps could be to increase the individual's budget of total steps for their agents. Currently, individuals receive the same step budget for both map sizes. Therefore, it is expected that smaller target maps are solved more effectively than larger ones.

Crossover Experiment: Figure 7 illustrates the average map difference across the 31 seeds of the three crossover operators and without crossover operator on the three environment types and two map sizes. As expected, using a crossover operator is essential to the optimization process. In all the scenarios, the algorithm without a crossover operator performs significantly worse than the other three algorithms using the different crossover operators. The mixed crossover operator is outperforming the other two operators in small environments during the first 100 generations. At around 100 generations on the small map, the micro crossover has the highest average map difference. During the end of the optimization, it appears that the objective value converges to about the same value for all crossover operators, with slightly better values for the mixed crossover operator. In the larger environments, we see that the objective value development is dependent on the map type. On the river map, the micro crossover operator seems to converge better than the others during the last 300 generations. On the checkerboard map, however, the mixed crossover operator converges better. During the first 100 generations, the macro crossover finds the best solutions in the river and the circular environment. To verify these findings, we performed two additional Tukey HSD tests to assess whether the three crossover operators show significant differences in objective value convergence during the final five generations. With the last five generations, we look at the last 1% of generations, which presumably hold the best solutions with the lowest objective values. In seven of the 12 instances where the mixed crossover is compared to the other two, the mixed crossover performs significantly better, and it never performs significantly worse. The micro crossover operator performs significantly better than the macro operator on the 11×11 river map and the 25×25 checkerboard map, while the macro crossover performs significantly better than the micro crossover on the 25×25 river and circular maps. Neither the

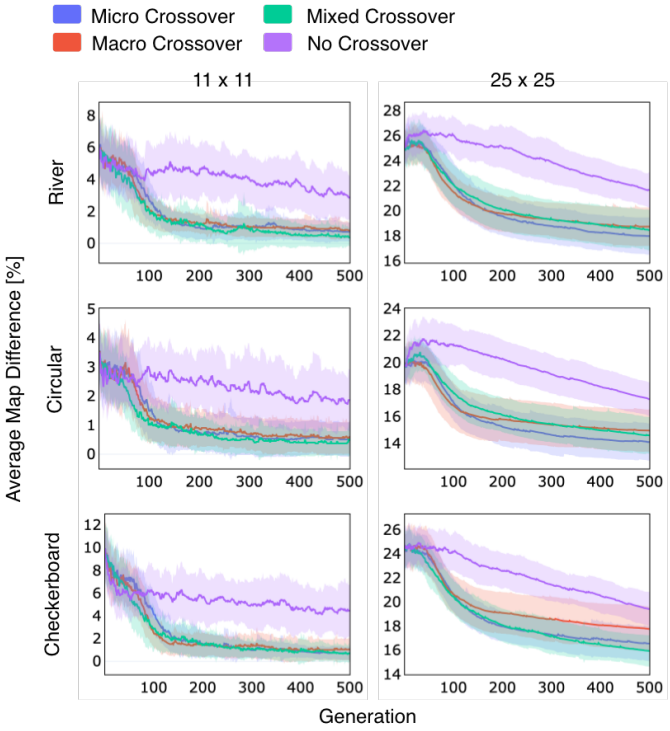

Fig. 7. Average map difference and standard deviation per generation during the optimization for micro, macro, mixed, and no crossover.

macro nor the micro crossover performs significantly better than the mixed crossover operator in any of the comparisons. One possible explanation for the significantly better results from the mixed crossover operator compared to the macro and micro crossovers is its more exploratory approach. By combining the finer adjustments from the micro crossover with the more substantial changes caused by the macro crossover, the mixed crossover exchanges more parts of the parents' genes than the other two operators. With this, the search space is more thoroughly explored, and better solutions were found.

VI. Conclusion

In this paper, we introduce a multi-agent navigation problem in modifiable environments, where agents operate in a shared grid world and can pick up, carry, and place obstacles, thereby altering the environment. The goal is to transform initial environments with randomly distributed obstacles into predefined target configurations represented by three scalable map types. While this work represents an initial step toward modeling such scenarios, potential real-world applications include smart warehousing and terrain leveling. We formulated the task as a scalable multi-agent navigation problem and applied evolutionary single-objective optimization to solve six problem instances, followed by an analysis of the impact of the number of agents on solution quality. The results indicate that increasing the number of agents can reduce solution quality, as optimization produces a higher proportion of infeasible solutions due to path collisions. Fewer agents lead to more feasible individuals and improved exploration, whereas a larger number of agents enables faster environment modification per time step through parallel actions. Our encoding of the individuals allowed us to use the crossover operator on different levels of the genes of the individuals: the agents and the paths of the agents. We propose different crossovers that operate on these different levels and find that operating on multiple levels of the encoding is significantly better in terms of convergence of the objective value in the majority of our experiments than only operating on either one of the levels of the encoding of the individuals. We conclude that there are both advantages and disadvantages to solving this problem with multiple agents simultaneously.

References

[1] R. Stern, "Multi-Agent Path Finding – An Overview," in *Artificial Intelligence*, G. S. Osipov, A. I. Panov, and K. S. Yakovlev, Eds. Cham: Springer Int. Publishing, 2019, vol. 11866, pp. 96–115.

[2] C. Nübel, M. Speidel, and S. Mostaghim, "Navigating path-influenced environments using evolutionary multi-objective optimization," in *Proc. of the Genetic and Evolutionary Computation Conf.*, ser. GECCO '25. New York, NY, USA: ACM, 2025, p. 1462–1470.

[3] J. Weise, S. Mai, H. Zille, and S. Mostaghim, "On the Scalable Multi-Objective Multi-Agent Pathfinding Problem," in *2020 IEEE CEC)*. Glasgow, UK: IEEE, Jul. 2020, pp. 1–8.

[4] A. Madkour, W. G. Aref, F. U. Rehman, M. A. Rahman, and S. Basalamah, "A Survey of Shortest-Path Algorithms," 2017.

[5] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Kumar, R. Barták, and E. Boyarski, "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks," *Proc. of the Int. Symposium on Combinatorial Search*, vol. 10, no. 1, pp. 151–158, Sep. 2021.

[6] H. Makino and S. Ito, "Online Multi-Agent Pickup and Delivery with Task Deadlines," 2024.

[7] H. A. Aryadi, R. Bezerra, K. Ohno, K. Gunji, S. Kojima, M. Kuwahara, Y. Okada, M. Konyo, and S. Tadokoro, "Multi-Agent Pickup and Delivery in Transformable Production," in *2023 IEEE 19th Int. Conf. on Automation Science and Engineering (CASE)*. Auckland, NZ: IEEE, Aug. 2023, pp. 1–8.

[8] D. Vainshtein, Y. Sherma, K. Solovey, and O. Salzman, "Terraforming – Environment Manipulation during Disruptions for Multi-Agent Pickup and Delivery," *Proc. of the Int. Symposium on Combinatorial Search*, vol. 16, no. 1, pp. 92–100, Jul. 2023.

[9] M. Bellusci, N. Basilico, and F. Amigoni, "Multi-agent path finding in configurable environments," in *Proc. of the 19th Int. Conf. on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20. Richland, SC: Int. Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 159–167.

[10] Y. Xue and J.-Q. Sun, "Solving the path planning problem in mobile robotics with the multi-objective evolutionary algorithm," *Applied Sciences*, vol. 8, no. 9, 2018.

[11] N. Urquhart, C. Scott, and E. Hart, "Using an evolutionary algorithm to discover low co2 tours within a travelling salesman problem," in *Applications of Evolutionary Computation*. Springer Berlin Heidelberg, 2010, pp. 421–430.

[12] G. Pavai and T. V. Geetha, "A survey on crossover operators," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016.

[13] P. Kora and P. Yadlapalli, "Crossover operators in genetic algorithms: A review," *Int. J. of Computer Applications*, vol. 162, no. 10, 2017.

[14] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[15] J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[16] J. Weise, "Evolutionary many-objective optimisation for pathfinding problems," 2022.