Playing Halma with Swarm Intelligence

Isabel Kuehner

Baden-Wuerttemberg Cooperative State University
Mannheim, Germany
Email: isabel.kuehner@isik-media.de

Adrian Stock

Baden-Wuerttemberg Cooperative State University
Mannheim, Germany
Email: adrian.stock@isik-media.de

***Abstract*—Swarm Intelligence algorithms are inspired by animals living together in swarms. Those algorithms are applicable to solve optimization problems like the Travelling Salesman Problem. Furthermore, they can be extended for playing games, e.g., board games. This paper proposes a novel approach for playing the board game Halma by combining Swarm Intelligence algorithms. It focuses on the implementation of a Swarm Intelligence player for the Halma game by combining two state-of-the-art algorithms, namely the Ant Colony Optimization, and the Bee Colony Optimization. In addition, we propose a modular Model View Controller software architecture for implementing the game and its players. Moreover, this paper evaluates the performance of the Swarm Intelligence agent for the single player and two player cooperative version of Halma. The algorithm presented in this paper is successful in learning the dynamics of the game and provides a stable basis for further research in this area.**

***Keywords*—Swarm Intelligence; Traveling Salesman Problem; Ant Colony Optimization; Bee Colony Optimization; Halma.**

I. Introduction

Animals like bees or ants live together in huge swarms. Although the number of members in the swarm is large, they are able to coordinate and divide tasks, e.g., splitting up the food searching process. The tasks are optimized within the swarm. Therefore, it is possible to derive algorithms, called Swarm Intelligence (SI) algorithms, for optimization problems, e.g., the Traveling Salesman Problem (TSP). Our previous work, which compared and applied different SI algorithms to the TSP problem [1], was published at "The Twelfth International Conference on Information, Process, and Knowledge Management eKNOW 2020". As SI algorithms are well suited for optimization problems like the TSP, several applications arise. SI approaches can be used for robotic swarms to explore unknown environments, e.g., in space. Another application for those algorithms are video or board games. This paper extends a previously published paper [1] by implementing and testing SI for a more complex application, namely the board game Halma.

Halma is a traditional board game which can be played using two different types of boards. Either on a square board or, as in our case, on a star-shaped board. Halma on a star-shaped board is also called "Sternhalma" in Germany or "Chinese Checkers" in the rest of the world, although it is not a variant of Checkers [2]. This paper focuses on "Sternhalma", but we will use the name Halma to refer to it. The main similarity between the TSP and the board game Halma is the structure of the problem. The Halma board, shown in Figure 1, is divided into nodes and edges, which is also the basis of the TSP. For a TSP, nodes represent cities which are visited by a salesman. He uses edges to travel from city to city. The aim is to find the optimal solution, so the salesman visits all nodes exactly once and reaches his starting point in the end. For a game of Halma, a player tries to find an optimal solution to get from its starting position to the goal position by using the edges. Due to the similarities and successful tests of SI algorithms for the TSP, it is promising to use SI approaches to construct a player for the Halma game.

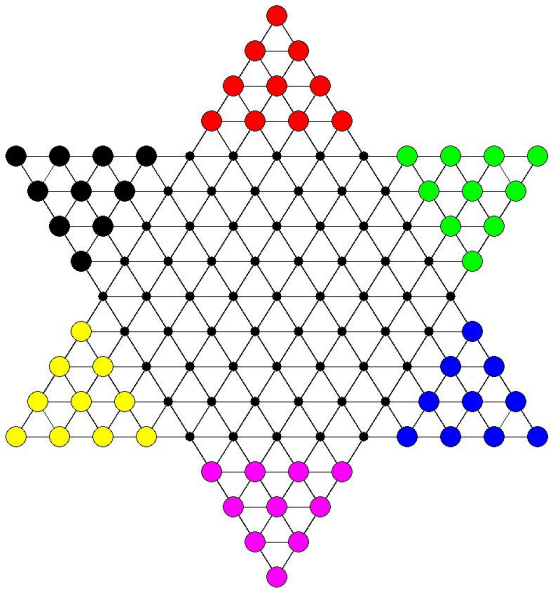


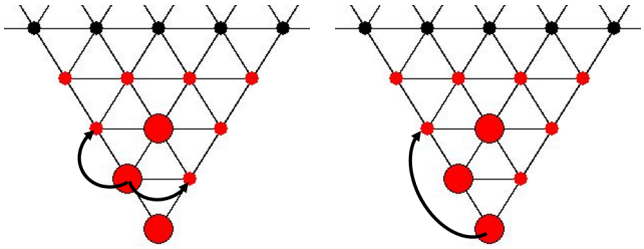
Figure 1. A Halma ("Sternhalma" or "Chinese Checkers") board for six players

The rules of the game are simple. Yet, the game is still challenging and interesting [3]. Its complexity is discussed in Section IV. Halma is played by one to six players having ten game characters each. A board with six players is shown in Figure 1. The board is star-shaped and consists of nodes and edges. Characters are allowed to stand on nodes and move from node to node using the edges. Goal of the game is to bring all game characters to the opposite side of the board.

For example, blue needs to place all its characters on the starting position of black and black needs to place all its game characters on the starting position of blue. To reach this goal, the player is allowed to move one game character at a time. Each character has two possible types of moves which are illustrated in Figure 2. It can either

- make a step move or
- make a jump move.

If it decides to make a step move (Figure 2(a)), the character can be moved in any direction to the next node, requiring the destination node to be unoccupied. If the player decides to jump, a neighboring node needs to be occupied and the node on a direct line behind this node needs to be unoccupied. It is irrelevant if the neighboring node is occupied by an own or an opponent's piece. This move is shown in Figure 2(b). If possible, the player is allowed to do multiple jumps in a row with the same character. He is able to decide how many jumps he wants to do as long as the move is valid.



(a) A character is allowed to make one step in any direction

(b) A character is allowed to jump over another character

Figure 2. The two different kinds of valid moves in Halma

The game ends if one player places all its game characters on the opposite side of its starting position. As Halma is a competitive game, it is a valid strategy to prevent other players from winning.

In general, games are interesting problems in the fields of Artificial Intelligence (AI). The complexity of games is often high and there are several possible strategies. Solving them needs a large amount of time and computational resources. Different games provide different challenges for AI. Most of all, board games are well suited to do research in the fields of AI, as they have simple rules which lead to a simple implementation. Additionally, experiments can be conducted on hardware with less computational power [4].

An AI player can play a game to either win or to gain experience out of it [4]. Halma, like most board games, is a competitive game. It ends when one of the players wins. Nevertheless, for the two player case, the shortest possible solution to win the game can only be reached, if the two players cooperate with each other. Therefore, this paper presents a strategy for a single player, forming a swarm of ten characters, as well as a strategy for two cooperating players.

Additionally, we propose a Model View Controller (MVC) architecture to realize the game. It is designed to easily exchange the AI player with different algorithms.

For accomplishing this, the paper makes use of the results gained in [1]. First, related work is presented. Second, two SI algorithms, the Ant Colony Optimization (ACO) and the Bee Colony Optimization (BCO), are presented. The complexity of the game is discussed in Section IV. The combination of the two SI algorithms to form a Halma player is presented in Section V. Section VI focuses on the experimental setup, our software architecture, and the results gained for a single player game and a two player game. The last section (Section VII) concludes the work and presents future work.

II. Related Work

Both, [5] and [6], give a good overview of the different SI approaches and their analogue in nature. This paper focuses on two out of several SI algorithms, the ACO [7] and the BCO [8]. Recent researches utilize those algorithms for a wide range of applications. Approaches based on the ACO are used, among others, for a routing protocol for Wireless Sensor Networks (WSN) [9], to load balance peer-to-peer networks [10] or a fuzzy logic controller [11]. Furthermore, the ACO has been applied in swarm robotics, e.g., for Unmanned Aerial Vehicles (UAVs) [12], or path planning on mobile robots in [13] and [14]. Variants of the BCO have been employed, e.g., for a swarm of autonomous drones [15] or path planning [16].

An extensive analysis of the game Halma can be found in [17] studying the six-piece game and in [2] focusing on the ten-piece game.

To the authors knowledge solving the Halma game with AI, especially with SI, is not a widely researched area. In [18], the authors design a Halma player based on deep reinforcement learning. Roschke and Sturtevant [19] use an Upper Confidence Bounds (UCB) applied to trees (UCT) algorithm to solve the Halma game. Both papers focus on the two-player game reducing the star-shaped Halma board to the two player square board. Additionally, both approaches are applied to a Halma game with six game characters per player instead of ten, which is the number of pieces used throughout this paper.

There are also only a few approaches using SI algorithms to learn playing games. In [20], the authors use an SI approach for a board game called "Terra Mystica". Kapi et al. [21] consider SI as a method to solve path planning in video games. In [22], the ACO has been used to for a video game called "Lemmings". Daylamani-Zad et al. [23] discuss a variant of BCO and its applicability to strategy games.

Thus, we can see that this paper tackles two ill-researched areas and presents a novel approach in playing Halma with SI based methods.

III. SI Algorithms

In the following, two state-of-the-art SI algorithms are presented. In this paper, the application for SI algorithms is the

board game Halma. The Halma game, as well as the TSP, is based on nodes connected by edges. Consequently, this chapter focuses on the definition of both SI algorithms for discrete problems. The introduction of the algorithms has been adapted from our previous publication [1]. First, the ACO is presented. Second, the BCO is summarized. Third, both algorithms are compared for a TSP application and we evaluate their use for the Halma board game.

A. Ant Colony Optimization (ACO)

When searching for food, ants leave pheromone trails on their path. Other ants can sense the pheromones and plan their path accordingly. This behavior is adapted for the ACO algorithm [24]. Ants mediate information over the environment and communicate therefore indirectly with the other members of the swarm. This form of communication is known as stigmergy communication [25]. Assume a simple example, where an ant can choose between two possible ways to get from the nest to the food source. One path is shorter than the other as visualized in Figure 3.

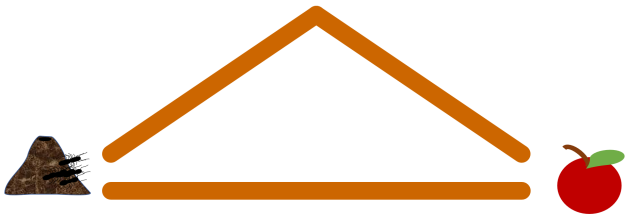


Figure 3. A swarm of ants will choose the short path in favor of the long path to get to the food source over time.

In the beginning, each individual ant chooses its way randomly, i.e., both paths have the same probability to be chosen. The members of the swarm that have chosen the short path will reach the food source earlier than the ants that chose the long path. When they arrive at the food source, they take a piece of food and return to the nest. Now, they have to make the decision once more, which way to chose. As ants leave pheromones on the path, while they move, they sense those pheromones on the short path. If the ants that took the long path have not arrived at the food source yet, the ants sense less pheromones on the long path. Ants choose the path where they can sense more pheromones. Therefore, they will choose the short way to get back to the nest.

To avoid a convergence of the swarm towards local minima, the pheromones on the paths evaporate partly [26]. Nevertheless, the pheromone value on the shorter path is higher than on the longer one. When the ants go to the food source and back to the nest multiple times, the pheromone value on the short path will grow over time. As a result, all ants decide to take the short way in the end [24].

The ACO algorithm simulates this food searching process. In the following, the algorithm is explained for a TSP application. For this application, the path is represented by a sequence of nodes, which are connected by edges. Table I summarizes the symbols used in the equations throughout this section.

The procedure of the ACO is divided into four phases:

1) path planning depending on the pheromone values on the path,
2) pheromone update on each ant's path,
3) pheromone update of the global-best path,
4) pheromone evaporation on all edges.

All phases are iterated multiple times, which is visualized in Figure 4. The next paragraphs focus on the explanation of each phase.

1) Path Planning: Path planning of each individual ant is based on the State Transition Rule

$$s = \begin{cases} \arg\max_{u \in J_{k(r)}} \{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}\}, \\ \quad \text{if } q \leq q_0 \text{ (exploitation)} \\ S, \quad \text{otherwise (biased exploration)} \end{cases}, \quad (1)$$

where r is the current node of the ant k, s is the next node, and q is calculated randomly [27]. Equation (1) defines the weighting between exploration and exploitation. If q is smaller than or equal to q_0, the ant chooses exploitation. Otherwise it chooses exploration. When choosing exploitation, path planning is based on the value of pheromones on the edge $\tau(r,u)$ and the distance $\eta(r,u)$ between the current node r and a possible next node u. The parameter β regulates the balance between distance and pheromone value. The maximum is chosen from calculating $[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}$ for all nodes that have not been visited yet (for all $u \in J_{k(r)}$) [27].

TABLE I. SYMBOLS USED IN THE FORMULAS EXPLAINED IN SECTION III-A [1]

Symbol Used	Meaning
s	next node
r	current node
u	next possible node
k	ant
$J_{k(r)}$	all nodes that have not been visited yet by ant k
$\tau(r,u)$	pheromone value of an edge between r and u
$\eta(r,u)$	inverse of distance between r and u
β	parameter to manipulate the proportion between distance and pheromone value ($\beta > 0$)
q	random number between [0...1]
q_0	proportion between exploration and exploitation ($0 \leq q_0 \leq 1$)
S	random variable connected to the random-proportional rule
$p_k(r,s)$	probability to choose node s as next node
ρ	pheromone decay parameter for local update ($0 < \rho < 1$)
τ_0	initial pheromone value
α	pheromone decay parameter for global update ($0 < \alpha < 1$)
δ	evaporation parameter ($0 < \delta < 1$)

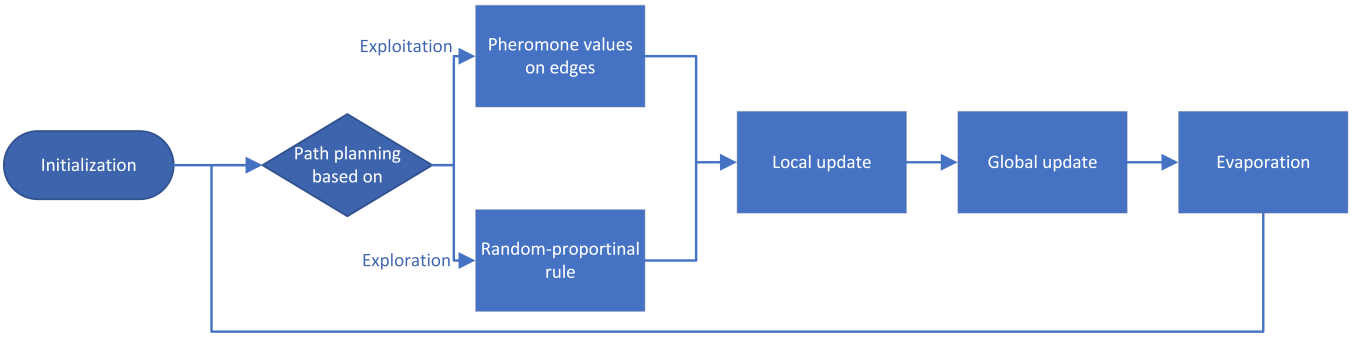


Figure 4. Steps of the ACO algorithm [1]

For biased exploration, the node selection is made with the random-proportional rule

$$S = p_k(r,s) = \begin{cases} \frac{\tau(r,s)\cdot\eta(r,s)^\beta}{\sum_{u\in J_{k(r)}} \tau(r,u)\cdot\eta(r,u)^\beta} & \text{if } s \in J_{k(r)} \\ 0, & \text{otherwise} \end{cases}, \quad (2)$$

where S represents the result of this random-proportional rule [27]. Equation (2) calculates a probability for each node. It is based on the length of an edge and its pheromone value. The shorter the edge and the higher the pheromone value, the more likely this node is chosen. As both, the pheromone value and the length of an edge are considered, $[\tau(r,u)] \cdot [\eta(r,u)]^\beta$ is calculated as well [27]. This results in a weighted value and the exploration is referred to as biased exploration [7]. The term is divided by the sum of all $[\tau(r,u)] \cdot [\eta(r,u)]^\beta$ to calculate the probability.

Each ant leaves a pheromone trail on its path. The trail is updated after each ant has finished its tour and has returned to the initial node. This pheromone update is called the local update.

2) Local Update Rule: Real ants leave pheromones on their trail. The pheromone values increase with the quality and the quantity of the food at the food source [24]. In analogy to real ants, the pheromone values on a path constructed by the artificial ants are updated with

$$\tau(r,s) = \tau(r,s) + \rho \cdot \Delta\tau(r,s)\,, \quad (3)$$

where $0 \leq \rho \leq 1$ [27]. The local update rule is influenced by $\Delta\tau(r,s)$. Depending on the implementation, you can choose different approaches to set $\Delta\tau(r,s)$ [27]. There are implementations which use, e.g., reinforcement learning to determine an appropriate $\Delta\tau(r,s)$ [27]. For sake of simplicity, we use a constant value

$$\Delta\tau(r,s) = \tau_0\,, \quad (4)$$

where τ_0 corresponds to the initial pheromone value.

3) Global Update Rule: After the local update has been performed and all ants have returned to the nest, the global update is conducted. All paths that have been found by the individual ants are compared to the global best path. For the TSP this is the path with the shortest length. For the Halma game this can, e.g., be a move that brings the game character closest to the goal positions or a move that uses many jumps. If one of the ants found a better path, the global best path is updated. No matter if it has been updated in the current iteration or not, the pheromone values on the edges are updated for the globally best path. Extra pheromones are added with

$$\tau(r,s) = \tau(r,s) + \alpha \cdot \Delta\tau(r,s)\,, \quad (5)$$

where α is a predefined pheromone decay parameter between 0 and 1 [27].

By adding pheromones to the edges in each iteration, the pheromone values on the edges increase over time. When a good solution was found in, e.g., iteration 10 and the swarm does not find a better solution during the following iterations, the pheromone value on this path is high. If an ant finds a better solution, it takes many iterations until the swarm will use this solution, as the pheromone value on the old best path is still high. Therefore, it is difficult to abandon old solutions. Consequently, we have to introduce evaporation to overcome this issue.

4) Evaporation: To avoid rapid convergence towards a non-optimal solution, pheromone values evaporate partly when they are updated. It offers the possibility to explore new areas [24]. The evaporation is regulated with a parameter δ and results in

$$\tau(r,s) = \delta \cdot \tau(r,s)\,. \quad (6)$$

We can also combine the evaporation with the local and global update rule [27]. The global update rule is now calculated with

$$\tau(r,s) = (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s)\,, \quad (7)$$

whereas the local update is computed with

$$\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s)\,. \quad (8)$$

Combining all steps, we result in the ACO as summarized in Figure 4.

The second algorithm, our SI Halma player is based on, is the BCO, which is presented in the following section.

B. Bee Colony Optimization (BCO)

Another state-of-the-art SI algorithm is the BCO. It is derived from the foraging behavior of bees [28]. In contrast to ants, bees use a different kind of communication. Ants communicate indirectly over the environment by leaving pheromone trails. On the contrary, bees communicate directly with the other swarm members by means of dancing. Bees fan out the hive to search for food, return and dance to the other bees to communicate the location of a food source. The dance is a form of advertisement to convince the other bees to choose the food source they are advertising [28].

The BCO was inspired by this behavior to solve optimization problems, e.g., the TSP. The algorithm mainly consists of two steps,

1) the forward pass,
2) and the backward pass.

The algorithm is visualized more detailed in Figure 5. Like the ACO, the BCO uses multiple iterations to converge towards a solution. In contrast to the ACO, which constructs a path at once, the BCO is divided into several stages. During each stage, a bee conducts the forward, as well as the backward pass and builds a partial solution, i.e., a part of the path. The number of stages for the TSP application depends on the number of nodes m the path is appended by during each stage. The two steps, conducted during each stage, are explained in the following.

1) Forward Pass: The forward pass is the step of building a partial solution. Each bee fans out the hive and appends its path by its own partial solution. This represents exploration, as the partial solutions are calculated randomly. For the TSP the bees append their current path by m nodes they have not visited yet [8]. After appending the path and returning to the hive, the bees perform the backward pass.

2) Backward Pass: The backward pass is the phase, where the bees perform the waggle dance. Each bee has two options to either

- abandon its partial solution (exploitation) or
- dance and advertise its solution to the other bees (exploration) [28].

By abandoning its solution, the bee will exploit another bee's solution (one of the bees that dances) or the global best solution (the best solution that has been found so far). For the TSP application, the shorter the path length of another bee's partial solution, the more likely the bee will choose it. After choosing a partial solution the bee will add this partial solution to its path that has been constructed during the previous stages. It basically exchanges its partial solution constructed during the forward pass with another partial solution [8]. For the TSP, cities are visited only once. Consequently, it is crucial to check whether the chosen partial solution does contain cities that have already been added to the path. If that is the case, for our implementation, the honeybee returns to its own partial solution constructed during the forward pass.

A solution for the problem has been found if the forward and the backward pass have been finished for all stages. Subsequently, the global best solution is updated [28] and one iteration has been finished.

The following section focuses on the comparison of the two algorithms and their application for the TSP as well as for the Halma game.

C. Comparison of the Algorithms

In order to decide which of the algorithms to use for the Halma board game, we first tested them for the TSP. The TSP was simulated by placing ten nodes, representing the cities, randomly on a grid. The edges, connecting all nodes, have different length. The algorithms are supposed to find a path which connects all nodes while traveling a minimum distance. Each algorithm performed 200 iterations per test and 1000 tests have been conducted. The results shown in Figure 6 give an idea of the performance of the algorithms. As the TSP is only an exemplary application to compare the algorithms, the implementations are not optimized with respect to time efficiency and performance. Additionally, this paper focuses on a board game application and the goal is to play against a human player, so time efficiency can be neglected. For the evaluation of performance of each algorithm, the interested reader is referred to [29]. Figure 6 visualizes the average path length for each iteration. The ACO is visualized in blue, whereas the BCO is shown in red. The parameter configurations used for the experiments visualized in Figure 6 are summarized in Table II.

TABLE II. PARAMETERS USED FOR EXPERIMENTS [1]

ACO		BCO	
parameter	**value**	**parameter**	**value**
iterations	200	iterations	200
population	100	population	100
β	0.7	m	3
q_0	0.8		
ρ	0.7		
τ_0	10.0		
α	0.9		

The ACO converges towards the optimal solution, which is shown on the bottom in black, whereas the BCO converges quickly but towards a non-optimal path. Consequently, the ACO has a better balance between exploration and exploitation than the BCO. For the BCO, exploitation predominates over exploration. From Table II we can see that the BCO needs less parameters than the ACO that need to be tuned. Nevertheless, the balance between exploration and exploitation is much better for the ACO.

The ACO is well suited for the TSP problem, as the swarm contributes from all solutions of all other ants and does not only consider the global best solution found at some point in the past. Distributing pheromones on the edges makes an easy planning possible. If we decide to us the ACO on its own for a Halma player, the algorithm outputs us 10 different paths

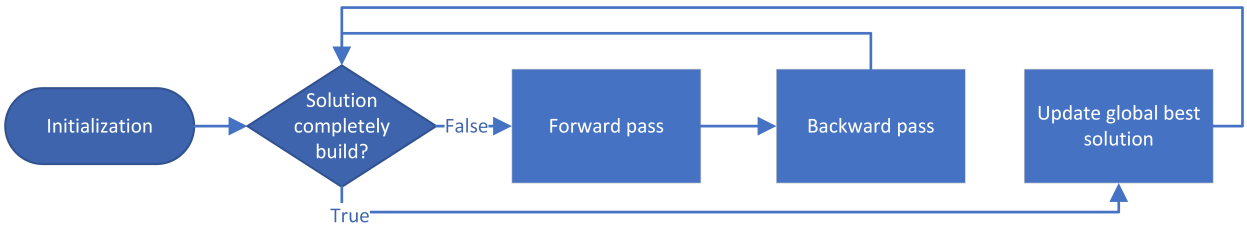


Figure 5. Steps of the BCO algorithm [1]

for 10 different characters. Then, we have to decide which game character executes its path, as the rules only allow one move of one game character at a time. With its abandoning and advertising scheme, the BCO is well suited to find the best solution within a swarm and to decide for one game character to move. Therefore, the ACO and the BCO are combined for the board game Halma. All game characters first plan a path with the ACO algorithm and then the BCO is used to decide which character will make the move. The implementation and combination of the two algorithms is further explained in Section V.

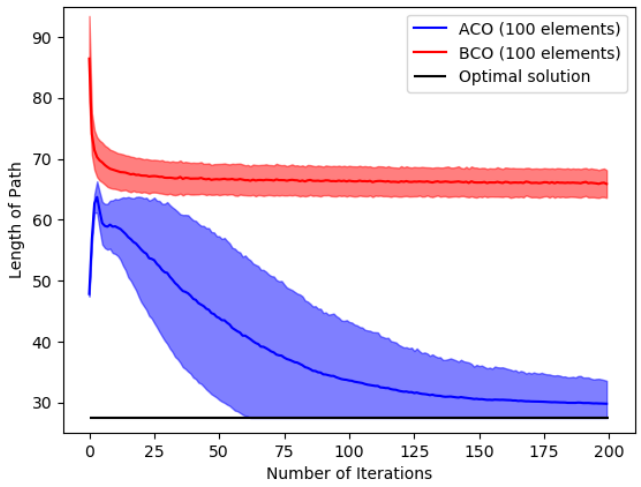


Figure 6. Average length of path by solving the TSP with the ACO (blue), the BCO (red) and the optimal solution (black) [1]

IV. Complexity of Halma

Even though its rules are simple, the board game Halma offers a high complexity. In [17], Sturtevant mentions that there are $1.73 \cdot 10^{24}$ states referring to a game with six game characters per player. The number of states is even higher for the ten-piece version of the game.

Due to the different possible moves, i.e., steps and jumps, and a varying number of players per game, and therefore also a varying number of game characters per game, there are usually a lot of different options per turn a player can choose from. Furthermore, players are allowed to perform multiple jumps in one turn with the same figure, if possible. Successive jumps do not necessarily have to follow the same direction. This contributes to even more different turns one can perform. As players are allowed to jump over the characters of other players, more players in a game allow a higher number of possible moves. In general, it is advantageous to jump as often as possible as this allows extra moves per turn. To reach this, the player needs to build ladders which transport characters quickly from one side to the other [2].

When playing Halma with only one player, the shortest solutions are often palindromic. This means, that the second half of moves is symmetric to the first half. According to [2] the shortest solution possible for one player has 27 moves and no shorter solution is possible.

For a game of two players the shortest game consists of 30 moves, that is 15 moves per player [2]. However, this only works if both players cooperate. In this scenario, two ladders are built. The first one is built by both players whereas the second one is only built by the player that is going to lose.

The following chapter considers this information and presents an implementation for a SI player for Halma using the algorithms introduced in Section III.

V. Combination of SI Algorithms for Halma

To use Swarm Intelligence for playing a game of Halma, the algorithms mentioned in Section III are combined. Figure 7 gives an overview of the implementation of an SI player for Halma. Table III includes the symbols used in equations throughout this chapter that have not been already introduced in Table I.

The ACO algorithm is well suited for the local path planning of each character. A single character is able to plan a path based on the pheromone values on the edges between the nodes. It decides whether to choose exploration or exploitation. The path is planned accordingly. In Figure 7 this process is marked in orange. The implementation of the BCO is illustrated in green. Originally, the forward pass of the BCO is used to make a local decision for each member of the swarm. In our algorithm, this decision is based on the ACO, so the forward pass is neglected, whereas the backward pass plays a major role. It is used to make the decision which character is going to make a move. The characters can either abandon their choice or advertise their solution to the others. This decision making process is further discussed in Subsection V-C.

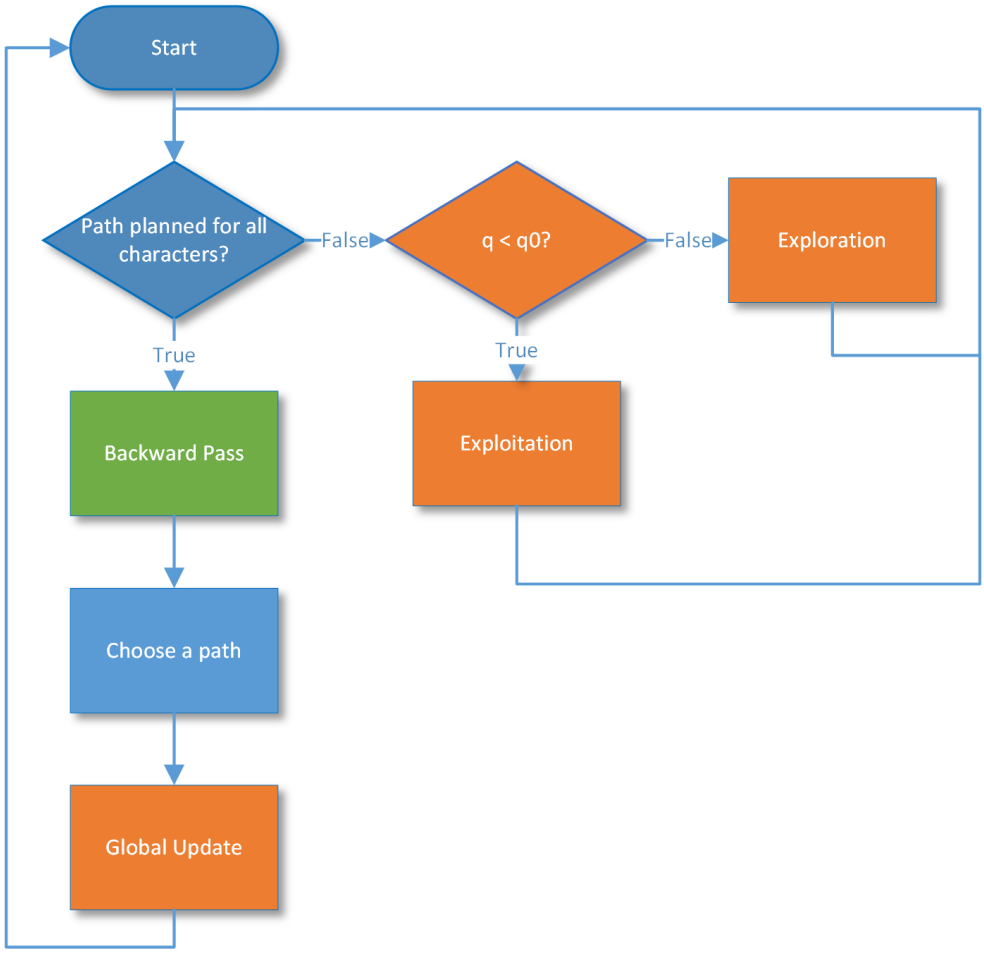


Figure 7. Process of choosing a character and path to make a move

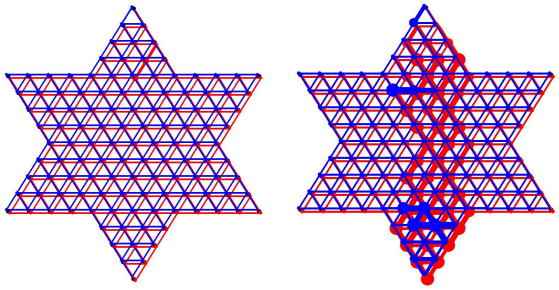
TABLE III. SYMBOLS USED IN THE FORMULAS EXPLAINED IN SECTION V THAT ARE NOT CONTAINED IN TABLE I.

Symbol Used	Meaning
$\tau 0_{goals}$	initial pheromone value for the two edges connected to the first row
j	jumping factor to reward moves with many jumps
g	in goal factor to increase the pheromones on edges that lead to the goal
e	evaporation rate
w_d	weight of distance to the first row
w_l	weight of the length of the path
w_p	weight of the pheromone values of the path
f_g	fitness value for characters that already reached the goal area
b	influence of best game played so far on the initial pheromone distribution
c	pheromone update value added during cooperative games
t_o	threshold to update the pheromone value during cooperative games

In the beginning, the pheromone values are distributed on all edges. This initialization is described in Subsection V-A. Subsection V-B discusses how the ACO can be applied to Halma. The global backward pass and the update rule, including evaporation, are further explained in Subsection V-C and V-D, respectively. Subsection V-E introduces some limitations for the pieces. The last subsection of this section focuses on the two-player cooperation.

A. Initial Distribution of Pheromones

The Halma board is divided into nodes and edges. The game characters need to move from node to node by using the edges. All edges are bidirectional and it is possible to use the same edge to move from node A to node B and from B to A. Therefore, it is necessary to distinguish which direction of the edge the agent moves along in order to judge if it is approaching the goal. As a consequence, it is necessary to double all edges for the implementation of the ACO. As a result, there are two edges connecting two nodes A and B, one to move from A to B and one to move from B to A. Each of the doubled edges has its own pheromone value. This value differs if the edge between A and B or the edge between B and A is examined. As each edge is doubled to have both directions, in Figure 8, one direction is marked in blue and the other in red.



(a) Pheromone distribution if all edges have the same starting pheromone value

(b) Pheromone distribution if the best solution found in a previous game is considered

Figure 8. Different pheromone distribution modes

Tests have shown that distributing the pheromones equally on all edges with

$$\tau(r,s) = \tau_0\,, \tag{9}$$

as visualized in Figure 8(a), the swarm needs a lot of time to reach the goal. Therefore, we introduced a second pheromone initialization mode. The pheromone values are initialized depending on their distance to the goal with

$$\tau(r,s) = \frac{1}{d} \cdot \tau_{0_{goals}} + \tau_0\,, \tag{10}$$

where d is the distance to the last goal position (the first row in Figure 9(a)), and $\tau_{0_{goals}}$ is a parameter for the initial pheromone value for the two edges connected to the first row. Edges connecting goal nodes have a higher initial pheromone value than edges connecting the start nodes. As a result, the swarm knows the direction where to go from the beginning. Additionally, we gave the swarm the ability to remember pheromone values of previous games. The pheromone distribution of the best game with the lowest number of moves played so far is stored. Its influence on the initial distribution of the pheromones can be weighted with the parameter b. Figure 8(b) illustrates the pheromone distribution for considering the best solution found so far. It can be exploited by the characters in the beginning of the new game.

After distributing the pheromones on all edges, the characters start planning their path by using the ACO algorithm.

B. Implementation of the ACO for Halma

Each game character needs to choose between exploration or exploitation. The parameter q_0 defines the probability to choose either of them. In the following exploration and exploitation are examined separately.

1) Exploitation: The character chooses exploitation if the randomly generated variable $q < q_0$. Exploitation means that the path is generated by using the trail with the highest pheromone value. Therefore, a character calculates all its valid moves. For each neighboring node of the character's position, all valid step and jump moves are calculated. To choose one of those paths, each path is evaluated with

$$p = l \cdot \sum_{e}^{k} \tau(r,s), \tag{11}$$

where l is the length of the path, $\tau(r,s)$ the pheromone value of an edge, and k the number of edges in the path. In contrast to the TSP application, for Halma we prefer longer paths over short moves. Long paths mean that we found several characters to jump over. Making several jump moves at once is crucial to find a efficient way to approach the goal. We therefore choose the path with the highest p. The chosen path is then considered for the global decision, which character is going to move.

2) Exploration: If the character chooses exploration, the general procedure is similar to exploitation. In contrast to the standard ACO the random-proportional rule was replaced, so the edges are chosen completely random without any bias. The edges of the character's path are chosen randomly from all neighboring edges that enable a valid move. The exploration procedure also implements both, jump moves and step moves.

In contrast to exploitation, exploration only produces one path and no decision is needed to choose between potential paths.

3) Balance Between Exploration and Exploitation: The balance between exploration and exploitation is important to generate new solutions as well as to exploit good solutions. For exploration, choosing edges randomly leads to new paths to visit nodes and edges which have not been part of a path so far. With exploration it is possible to find paths to the goal which lead to fewer draws than the already found solutions. Those paths found by exploration can be used by other game characters throughout exploitation. They follow the paths that have already been chosen by other characters. The more characters follow a path the higher is the pheromone value on these paths. This results in the exploitation of good explored solutions.

With the ACO each character plans the path it would take if it is going to move. As only one character is allowed to move at a time, the global backward pass of the BCO is used to make this decision.

C. Global Backward Pass

For performing the backward pass, each character calculates a fitness for its path. This fitness is calculated with

$$f = w_d \cdot \frac{1}{d} + w_l \cdot l + w_p \cdot p\,. \tag{12}$$

To calculate the fitness, three components are taken into account:

- the distance d between the end node of the path and the first row (farthest goal position)
- the length of the path (l)
- the sum of all pheromone values on all edges of the path (p)

Each component has its own weighting factor w_d, w_l, w_p respectively. Those weighting factors are given as parameters. If a character is already at one of the goal positions the equation for calculating the fitness is changed to

$$f = w_d \cdot \frac{1}{f_g} + w_l \cdot l_p + w_p \cdot p\,. \tag{13}$$

This avoids draws inside the goal, although there are still characters at the starting position or in the middle of the field. Here, the distance d was replaced by a factor f_g that is associated with a parameter.

After calculating the fitness of each character, the sum of all fitness values is calculated. Then, the decision is made if a piece abandons its path or if it presents its solution to the others. The decision is based on a probability $\tau(r, s)$ for a character i which is calculated with the roulette wheel rule according to

$$\tau(r, s) = \frac{f_i}{\sum\limits_{c \in C} f_c}. \tag{14}$$

The fitness f_i of the character i is divided by the sum of all fitness values of all characters C.

The probability $\tau(r, s)$ defines if the character abandons or advertises its solution. Is $\tau(r, s) > \frac{1}{|C|}$, the character advertises its path. Otherwise, it abandons it.

Each character that decides to abandon its solution, needs to choose one of the solutions advertised by another character. Therefore, a similar equation to Equation (14) is used, but only the fitness values of characters are summed up that advertise their path.

One character is chosen from all characters that advertise their solution by an Equation similar to (14) and makes a move.

D. Update Rule

After a character has been chosen to make a move, the edges of its path are updated. In contrast to the original ACO algorithm, the update rule is only used on a global level and the local update rule has been neglected. We use only the best result of the paths proposed by our ten characters. The paths of the other nine characters may be worse and would negatively influence the pheromone distribution. Therefore, when testing the local update rule, their results have been worse and we decided to neglect it. The pheromone update is illustrated with

$$\tau(r, s) = \tau(r, s) + \alpha \cdot l \cdot \tau_0 \cdot \begin{cases} 2 \cdot j, & \text{jump moves and } l > 2 \\ 1 \cdot j, & \text{jump moves} \\ 1, & \text{else} \end{cases}. \tag{15}$$

A pheromone value, depending on the properties of the path, is added to the pheromone value of an edge $\tau(r, s)$. If the character makes a move without jumping, a pheromone value of $\alpha \cdot l \cdot \tau_0$ is added, where l is the length of the path. If the character's move includes one or more jumps, a jumping factor j, given as a parameter, is taken into account. If the character jumps more than once, this factor is multiplied by two. This has the effect that a long path raises the pheromone value of the edges significantly and other characters might consider this path as well in the next iteration.

If a character reaches one of the goal positions, the pheromone values on all edges the character has been visiting throughout the game are updated. This increases the pheromone values on edges which lead to the goal. Other characters are able to plan their path accordingly. Consequently, after one character reaches a goal position, the probability for the other characters increases to get to the goal faster. The edges are updated with

$$\tau(r, s) = \tau(r, s) + \rho \cdot \tau_0 \cdot g\,. \tag{16}$$

The initial pheromone value τ_0 is multiplied by an in goal factor g and by ρ which are both given as a parameter. The result is then added to the edge's pheromone value.

To avoid the algorithm to get stuck into a local minimum the pheromone values on all edges evaporate partly. The pheromones only evaporate after a predefined number of moves (e), to avoid an evaporation of the pheromones on long paths too quickly. The updated pheromone value of an edge is calculated with

$$\tau(r, s) = (1 - \alpha) \cdot \tau(r, s)\,. \tag{17}$$

E. Limitations for the Characters

The characters plan their path autonomously. After doing first tests a few problems were detected that needed to be restricted. First of all, the characters are supposed to stay in the goal area once they arrive there. Figure 9(a) illustrates the goal positions of one player in red. If a character reaches one of the red points, it is in the goal region.

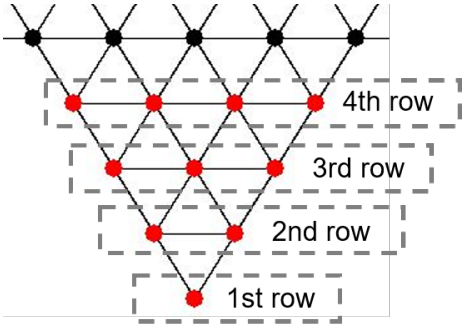
To avoid that the characters do small moves in front and inside the goal, they need to fill the goal from the end. This is shown in Figure 9(b). If a character reaches the first row, it is not allowed to move anymore. The second row needs then to be filled next. The character on the left of the two nodes is not able to move. For the characters that are not allowed to move, no local path is calculated and they are not part of the decision making process. If the character in the Figure 9(b) moves to the node where the arrow is pointing to, it is not allowed to move in future moves. The list of nodes that will be filled next is updated so the third row is filled during the next draws.

F. Cooperative Game

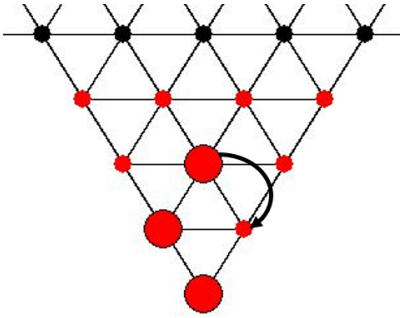
As mentioned in [2], an optimal solution for the two-player case can only be reached if the two players cooperate. Consequently, we implemented an option to start a cooperative two-player game. After each move, the two players exchange their pheromone distribution. As both players start at the opposite of each other, the pheromone distributions are then inverted and compared to the own distribution. The pheromone values are then updated according to

$$\tau(r,s) = \tau(r,s) + \begin{cases} c, & \text{if} \tau(s,r)_o > t_o \\ 0, & \text{otherwise} \end{cases}, \qquad (18)$$

where $\tau(s,r)_o$ is the pheromone value of the other agent matching $\tau(r,s)$ and c and t_o are given as a parameter.



(a) Visualization of the different rows in the goal positions



(b) The goal is filled from the end

Figure 9. Goal positions of one player

The implementation proposed throughout this section has been tested for playing Halma games with SI. The experimental results are presented in the following chapter.

VI. Experiments

This section focuses on the experiments conducted for the SI player. First, the experimental setup is defined and the architecture of our implementation is proposed. Second, the experimental results for single as well as two-player games are presented.

A. Experimental Design

In order to optimize the algorithm designed for Halma we implemented a framework for Halma. As depicted in Fig. 10 the application is based on a Model View Controller (MVC).

Due to the modularization of the different components, we can connect both, an agent based on the presented SI algorithm but also an agent that uses any other algorithm. This has the following advantages:

- It allows future research with algorithms of different archetypes. We could, e.g., include agents based on reinforcement learning by solely exchanging the agent.
- A learning algorithm can play against a human player and learn from their strategy.
- It allows the initialization of the presented SI algorithm with different hyperparameters. This can lead to one algorithm playing ”defensive”, i.e., building up a ladder and the other algorithm playing ”offensive”, i.e., making use of the ladder to reach the other side more quickly. This could greatly benefit the optimization of our solutions.

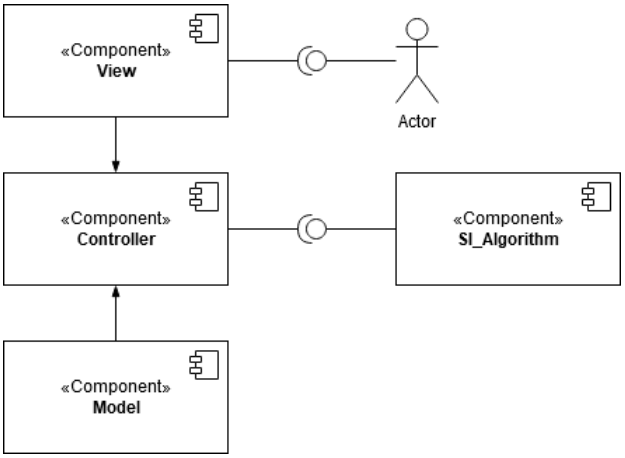


Figure 10. Component Diagram of the Halma framework

To test the SI player for a Halma game, we conducted experiments for the single player and two-player game. For both setups, the parameter configurations are the same and summarized in Table IV.

TABLE IV. Parameters used for playing Halma with SI

Parameter	Value
τ_0	5
$\tau_{0_{goals}}$	10
q_0	0.8
β	2.7
α	0.1
ρ	0.5
j	5000
g	10
e	11
w_d	2
w_l	50
w_p	5
f_g	50
b	2.5
c	5
t_o	20

We want to force the players to favor jump moves over step moves because they contribute to less draws. Therefore, the parameter value for j and w_l are high. Additionally, the large value for f_g reduces the number of draws within the goal region. If a game has not been finished after 500 moves, the game has been stopped and restarted.

Goal of the experiment with one agent is the comparison of number of moves a single SI player needs to win the game, in comparison to the best possible solution as discussed in Section IV. Additionally, we want to evaluate the effect of the initial distribution of pheromones on the number of moves needed to finish a game. Moreover, we conducted two-player experiments to evaluate the cooperation between two agents.

B. Experimental Results

In the following, we present the results gained for single, as well as two-player experiments. All configurations and the evaluation of number of moves are summarized in Table V and Figure 11.

1) Single player experiments: Figure 12 shows the number of moves of three test scenarios for 1000 games each. The more bold a point, the more often this number of draws has been achieved. The blue points on the bottom represent the number of moves without using the best solution found so far in a previous game (Id 1). Nevertheless, it uses an initial distribution of the pheromones introducing the direction where the swarm has to go. The orange points in the model represent the number of moves when using the best solution found so far with 68 moves (Id 2). The game with 68 moves has been the game with the smallest number of draws found when doing experiments. It has been achieved in a previous game. The pheromone distribution at the end of this game has been used to initialize the pheromones. The green points on the top represent the total number of moves when starting without the best solution found so far and updating it throughout the game, i.e., the first game of the 1000 played games was the best solution in the beginning and was updated during the experiment (Id 3). Figure 12 shows that using the best solution that was found by the swarm in the past has an effect on the number of draws. Using the best solution found so far reduces the median of the number of moves. Without using best result of the past, the median is 133. In comparison, when starting without a best solution and updating it throughout the experiment, the median is 120. When initializing the pheromone values considering the best solution with 68 moves, the median of total moves is 113. Furthermore, from Table V, the mean, as well as the standard deviation is lowest when using the best result found so far with 68 moves.

As a consequence, the initial distribution of pheromones has a large effect on the number of moves needed by the swarm. Without a hint in which direction to move, the swarm needs more moves to win the game.

Furthermore, we tested the SI player for different balance values between exploration and exploitation shown in Figure 13. In the aforementioned experiments, the parameter q_0 balancing between exploration and exploitation is 0.8. For the following test, we choose an initial pheromone distribution exploiting the best solution of 68 moves. The experiment has been conducted for 1000 games each by varying the q_0 parameter. Three values for q_0 have been tested, namely 0.6 (blue on the bottom), 0.8 (orange in the middle), and 1.0 (green on the top).

As 68 moves was the best result that has been reached so far with the algorithm, exploiting this solution more, results in a lower total number of moves than when increasing exploration. The SI player can exploit the pheromone values of the 68-moves solution, because they lead to a good result in the past. As seen in Figure 8(b), when using the best solution for pheromone initialization, the pheromones are mostly distributed on the area directly connecting the start and the goal. More exploring will therefore lead to worse results, as no new faster ways besides the direct paths to the goal can be found. Consequently, choosing $q_0 = 1.0$ leads to better results than $q_0 = 0.6$. As stated in [2], the 68 draws solution is still far from optimal. In order to improve this, we need exploration and therefore, we chose for the following experiments $q_0 = 0.8$.

2) Two-player Experiments: As mentioned in [2], the perfect game with two players consists of less moves per player than single player games. In order to validate if that is also the case for our SI Halma player, we present the results of several experiments in the following.

First, we directly compare a single player game with a two-player game using the same parameters. For this experiment, the players do not cooperate while playing. The results are presented in Figure 14. 1000 tests have been conducted for the single player scenario on the bottom (blue, Id 2), whereas the two-player scenario on the top (orange, Id 5) has been repeated 500 times. Comparing the median of the number of moves for both experiments, the two-player scenario outperforms the single player setup with 104 compared to 113 moves.

The following experiment focuses on the two-player setup. Four scenarios with 500 games each have been compared and are shown in Figure 15. If the best solution found so far has been used, the 68 moves solution already used for previous experiments has been chosen. The results for using the best solution and having a cooperation between the two players (Id 4) is shown in blue on the bottom. Using the best solution, but not having a cooperation is shown in orange (second from bottom, Id 5). For the other two results, no best solution has been used. For the green results (second from top, Id 6), the players have cooperated, wheres for the tests resulting in the red dots (top, Id 7), no cooperation has been introduced.

Both using the best result and the cooperation affect the median of the number of moves as visualized in Figure 11. The median of the number of moves when using both is 102, whereas it is 121.5 when not using both. Introducing a cooperation when not using the best solution improves the median from 121.5 to 114. When using the best solution found so far, the cooperation slightly improves the median of moves from 104 to 102. As visible in Figure 15, the standard deviation when not using the best solution is much higher than when relying on it. This is also proven by the standard deviations shown in Table V. Despite the fact, when using the best solution, the median of moves for cooperation is lower, the standard deviation is higher. Although cooperation can

TABLE V. RESULTS FOR THE NUMBER OF MOVES FOR DIFFERENT EXPERIMENTAL SCENARIOS.

ID	Players	Tests	Best Solution	Cooperation	Median	Mean	Standard Deviation
1	1	1000	x	-	133	153.46	63.27
2	1	1000	✓	-	113	116.82	33.42
3	1	1000	construct	-	120	132.81	39.2
4	2	500	✓	✓	102	118.35	58.45
5	2	500	✓	x	104	121.75	54.96
6	2	500	x	✓	114	142	73.32
7	2	500	x	x	121.5	152.91	81.4

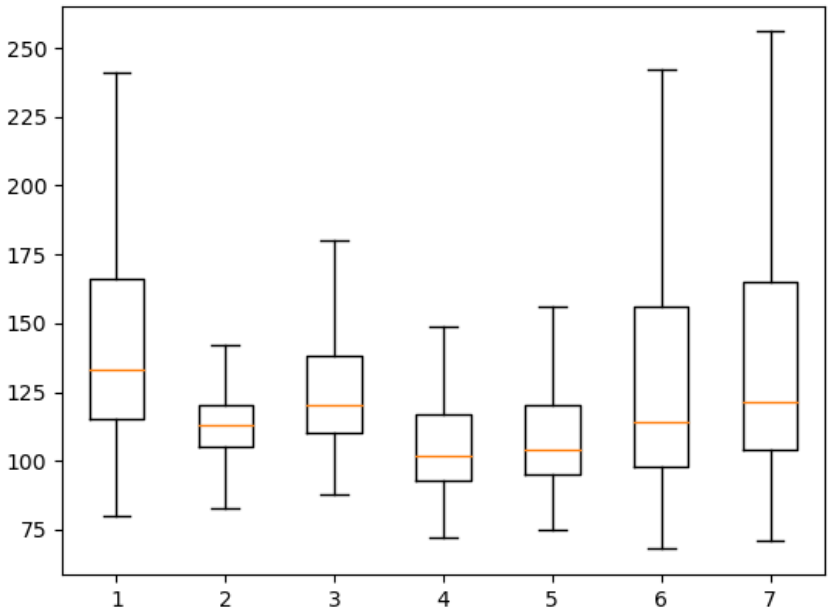


Figure 11. Distribution of draws until a player wins the game for the scenarios in Table V identified by the ID.

improve the number of moves, the solutions are far from the optimal solution of 15 moves per player specified in [2].

In general we can see from Table V and Figure 11 that we were able to improve the performance of the algorithm by remembering the best solution found so far. In the single, as well as the two-player scenarios, the number of total draws is less when using the best solution found so far than when the initial pheromone distribution is only based on the distance to the goal. Additionally with this approach, it was possible to reduce the standard deviation significantly. We can see that our algorithm is able to solve the game and our modifications including a ”memory” increased its performance. Although, the results for the single as well as the two-player case are far from optimal, further modifications and optimization can help to reach the optimal solutions.

VII. CONCLUSION AND FUTURE WORK

This paper presented an application for using a combination of SI algorithms. The ACO algorithm and the BCO algorithm have been combined to realize a nonhuman player for the board game Halma. Experiments have shown that the initial distribution of pheromones has a big influence on the performance of the SI player.

Nevertheless, the number of moves resulting from the experiments is still high in comparison to the optimal solution. Future work will therefore focus on decreasing the number of moves needed to win a game with one and multiple players. Furthermore, it is possible to compare the SI player to a human player. Most humans will not find the optimal solution when playing Halma. Therefore, it is interesting to do experiments by comparing human players and SI players.

In future work, we want to make use of this architecture to find out about the performance of other algorithms searching

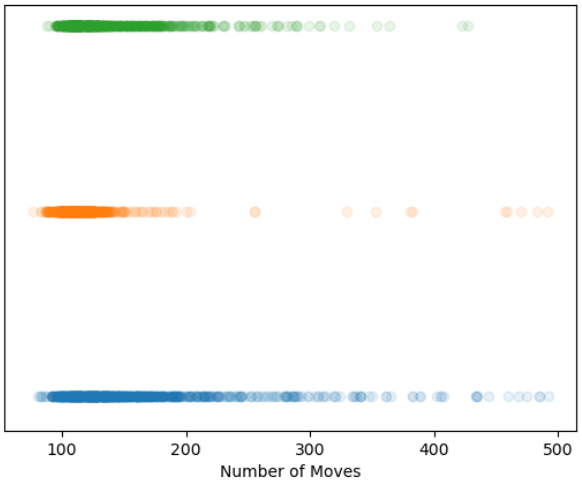


Figure 12. Number of moves for 1000 games. Blue (bottom) without using the best solution found in other games (Id 1). Orange (middle) using the best solution found so far with 68 moves (Id 2). Green (top) starting without a best solution and updating it whenever a better result has been found (Id 3).

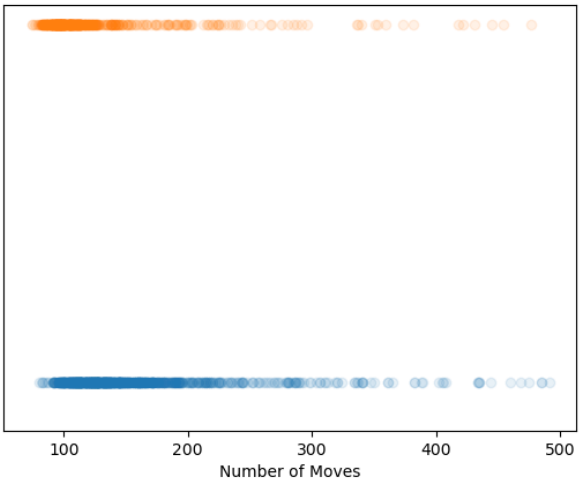


Figure 14. Number of moves for one player using the best solution found so far with 68 moves. 1000 games for a single player game (blue bottom, Id 2). 500 tests for two players without cooperation (orange top, Id 5).

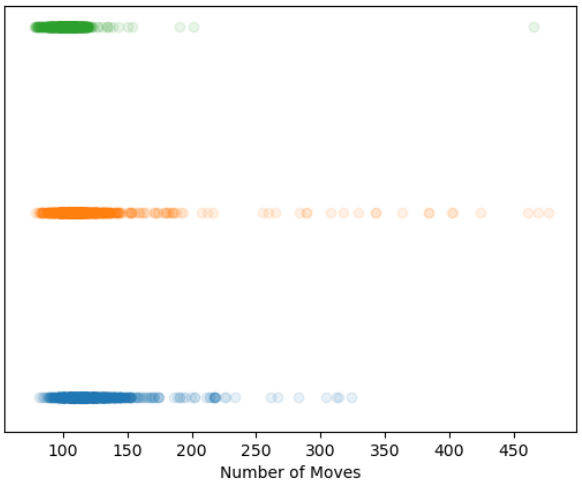


Figure 13. Number of moves for 1000 games. Blue (bottom) with $q_0 = 0.6$. Orange (middle) with $q_0 = 0.8$. Green (top) with $q_0 = 1.0$.

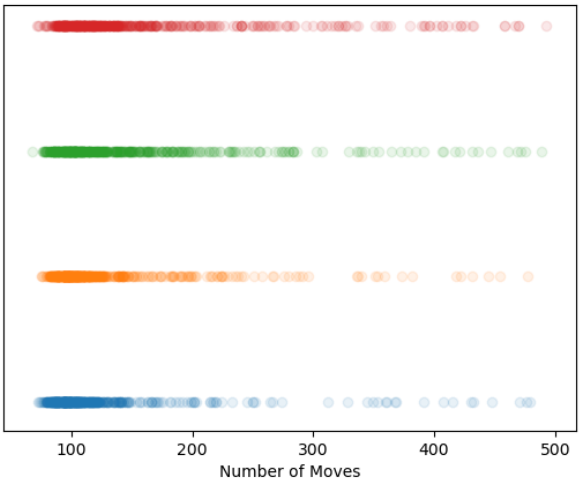


Figure 15. 500 Tests per experiment. Blue (bottom, Id 4) using the best solution (68 moves) and cooperation of the agents. Orange (second from bottom, Id 5) using the best solution (68 moves) and no cooperation. Green (second from top, Id 6) not using the best solution, but having a cooperation of the agents. Red (top, Id 7) no using the best solution and not cooperating.

for the shortest game possible. Furthermore, the architecture allows the initialization of the presented SI algorithm with different hyperparameters. This can lead to one algorithm building up a ladder and another exploiting the ladder to reach the other side more quickly. This could greatly benefit the optimization of our solutions.

This paper focuses only on one board game. SI can also be used for other board games with multiple characters. Another possible application would be Chess. In contrast to Halma, where all characters are equal, in Chess the members of the swarm have different roles. The decisions of the swarm need to consider the inequality of its members. SI can not only be used for board games, but for every game were multiple characters play together and need to make decisions. In video games, the human player often needs to play against other players and characters. If the game involves armies of opposing players, they can also act like a swarm. They need to find solutions themselves by making decisions which consider the solution of every single member of the swarm. Video games are more complex than board games and the effort for implementing an SI algorithm for a video game is higher than for a board game, but it enables a swarm-like behavior of the opposing player.

Not only simulated swarms in games can be optimized,

but the approaches can also be expanded for path planning in multi-agent systems and robotic swarms.

Acknowledgement

This paper has been written during a cooperative study program at the Baden-Wuerttemberg Cooperative State University Mannheim and the German Aerospace Center (DLR) Oberpfaffenhofen at the Institute of Communications and Navigation.

References

[1] I. Kuehner, "Swarm intelligence for solving a traveling salesman problem," in *eKNOW2020, The Twelfth International Conference on Information, Process, and Knowledge Management*. IARIA, 2020, pp. 20–27.

[2] G. I. Bell, "The shortest game of chinese checkers and related problems," *Integers*, vol. 9, no. 1, pp. 17–39, 2009.

[3] A. Elithorn and A. Telford, "Game and problem structure in relation to the study of human and artificial intelligence," *Nature*, vol. 227, no. 5264, p. 1205, 1970.

[4] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*. Springer, 2018, vol. 2.

[5] C. Blum and D. Merkle, *Swarm intelligence: introduction and applications*. Springer Science & Business Media, 2008.

[6] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*, ser. The Morgan Kaufmann series in evolutionary computation. San Francisco [u.a.] Morgan Kaufmann, 2009.

[7] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial life*, vol. 5, no. 2, pp. 137–172, 1999.

[8] D. Teodorović, "Bee colony optimization (bco)," in *Innovations in Swarm Intelligence*, C. P. Lim, L. C. Jain, and S. Dehuri, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 39–60. [Online]. Available: https://doi.org/10.1007/978-3-642-04225-6_3

[9] P. Maheshwari, A. K. Sharma, and K. Verma, "Energy efficient cluster based routing protocol for wsn using butterfly optimization algorithm and ant colony optimization," *Ad Hoc Networks*, vol. 110, p. 102317, 2021.

[10] S. Asghari and N. J. Navimipour, "Resource discovery in the peer to peer networks using an inverted ant colony optimization algorithm," *Peer-to-Peer Networking and Applications*, vol. 12, no. 1, pp. 129–142, 2019.

[11] O. Castillo, E. Lizárraga, J. Soria, P. Melin, and F. Valdez, "New approach using ant colony optimization with ant set partition for fuzzy control design applied to the ball and beam system," *Information Sciences*, vol. 294, pp. 203–215, 2015.

[12] Z. Ziyang, Z. Ping, X. Yixuan, and J. Yuxuan, "Distributed intelligent self-organized mission planning of multi-uav for dynamic targets cooperative search-attack," *Chinese Journal of Aeronautics*, vol. 32, no. 12, pp. 2706–2716, 2019.

[13] Y. Liu, J. Ma, S. Zang, and Y. Min, "Dynamic path planning of mobile robot based on improved ant colony optimization algorithm," in *Proceedings of the 2019 8th International Conference on Networks, Communication and Computing*, 2019, pp. 248–252.

[14] R. Uriol and A. Moran, "Mobile robot path planning in complex environments using ant colony optimization algorithm," in *2017 3rd international conference on control, automation and robotics (ICCAR)*. IEEE, 2017, pp. 15–21.

[15] A. Viseras, T. Wiedemann, C. Manß, V. Karolj, D. Shutin, and J. M. Gomez, "Beehive inspired information gathering with a swarm of autonomous drones," *Sensors*, October 2019. [Online]. Available: https://elib.dlr.de/129660/

[16] Z. Li, Z. Zhang, H. Liu, and L. Yang, "A new path planning method based on concave polygon convex decomposition and artificial bee colony algorithm," *International Journal of Advanced Robotic Systems*, 2020. [Online]. Available: https://doi.org/10.1177/1729881419894787

[17] N. R. Sturtevant, "On strongly solving chinese checkers," in *Advances in Computer Games*. Springer, 2019, pp. 155–166.

[18] Z. Liu, M. Zhou, W. Cao, Q. Qu, H. W. F. Yeung, and V. Y. Y. Chung, "Towards understanding chinese checkers with heuristics, monte carlo tree search, and deep reinforcement learning," *arXiv preprint arXiv:1903.01747*, 2019.

[19] M. Roschke and N. R. Sturtevant, "Uct enhancements in chinese checkers using an endgame database," in *Workshop on Computer Games*. Springer, 2013, pp. 57–70.

[20] L. J. P. de Araújo, A. Grichshenko, R. L. Pinheiro, R. D. Saraiva, and S. Gimaeva, "Map generation and balance in the terra mystica board game using particle swarm and local search," in *International Conference on Swarm Intelligence*. Springer, 2020, pp. 163–175.

[21] A. Y. Kapi, M. S. Sunar, and M. N. Zamri, "A review on informed search algorithms for video games pathfinding," *International Journal*, vol. 9, no. 3, 2020.

[22] A. Gonzalez-Pardo, F. Palero, and D. Camacho, "An empirical study on collective intelligence algorithms for video games problem-solving," *Computing and Informatics*, vol. 34, no. 1, pp. 233–253, 2015.

[23] D. Daylamani-Zad, L. B. Graham, and I. T. Paraskevopoulos, "Swarm intelligence for autonomous cooperative agents in battles for real-time strategy games," in *2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*. IEEE, 2017, pp. 39–46.

[24] C. Blum and X. Li, "Swarm intelligence in optimization," in *Swarm Intelligence: Introduction and Applications*, C. Blum and D. Merkle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 43–85. [Online]. Available: https://doi.org/10.1007/978-3-540-74089-6_2

[25] V. Trianni, *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots*. Springer, 2008, vol. 108.

[26] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge : MIT Press, 2004.

[27] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on evolutionary computation*, vol. 1, no. 1, pp. 53–66, 1997.

[28] K. Diwold, M. Beekman, and M. Middendorf, "Honeybee optimisation – an overview and a new bee inspired optimisation scheme," in *Handbook of Swarm Intelligence: Concepts, Principles and Applications*, B. K. Panigrahi, Y. Shi, and M.-H. Lim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 295–327. [Online]. Available: https://doi.org/10.1007/978-3-642-17390-5_13

[29] J. Odili, M. N. M. Kahar, A. Noraziah, and S. F. Kamarulzaman, "A comparative evaluation of swarm intelligence techniques for solving combinatorial optimization problems," *International Journal of Advanced Robotic Systems*, vol. 14, no. 3, p. 1729881417705969, 2017.