

A Study on non-Overlapping Multi-Agent Pathfinding

Mohammadreza Daneshvaramoli¹, Mohammad Sina Kiarostami², Saleh Khalaj Monfared¹, Helia karisani¹, Aku Visuri², Simo Hosio², Dara Rahmati³, and Saeid Gorgin⁴

¹ School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

{daneshvaramoli, monfared, h.karisani}@ipm.ir,

² Center for Ubiquitous Computing, Faculty of ITEE, University of Oulu, Oulu, Finland

{mohammad.kiarostami, aku.visuri, simo.hosio}@oulu.fi,

³ Computer Science and Engineering Department, Shahid Beheshti University, Tehran, Iran

d_rahmati@sbu.ac.ir,

⁴ Iranian Research Organization for Science and Technology (IROST), Tehran, Iran
gorgin@irost.ir

Abstract. In this work, first, we model the non-overlapping Multi-Agent Pathfinding (MAPF) to an *NP-complete* traditional puzzle called *Numberlink* puzzle owing to its features. Interestingly, this puzzle is reasonably shown to be analogous to the *Flow Free* game. Hence an approach that solves the puzzle can be considered as the AI for solving Flow Free game. Then, we investigate various promising approaches such as SAT, Heuristics, and Monte-Carlo Tree Search (MCTS) based methods to find a fast and accurate solution and provide a fair comparison. We implement and evaluate two SAT and MCTS-based approaches. Finally, we propose an enhanced MCTS with three optimizations to solve the problem faster with lower memory consumption, particularly in significant test sizes with many agents. All the methods are compared and analyzed on the same test cases in different grid sizes and various agents. The optimized MCTS-based method solves the most extensive test case with a size of 40×40 with 100 agents in 988.5 seconds, respectively, indicating 22.8% and 63.6% improvements in time and memory consumption compared to the state-of-the-art MCTS-based method. It also shows 72% and 39.2% improvement in performance with lower memory consumption than the best results of investigated SAT and heuristic-based methods, sequentially.

Keywords: NumberLink, Multi-Agent, Pathfinding, Monte-Carlo Tree Search (MCTS), SAT.

1 Introduction

Many AI advances and developments have emerged with attempts to solve classic computer science challenges. Finding solutions for puzzles in this context has always drowned many researchers' attention, propelling them to approach the problems with new and novel methodologies, including AI capabilities. Pathfinding in AI has known to be an essential general problem in the community, challenging many researchers over time [1]. MAPF is a critical classic problem of finding multiple agents' paths toward their goals with no (or minimum) collisions [2]. Solutions for various extensions and different generalizations with specific constraints to this problem have found real-world applications in many areas such as robotics [3], computer games [4, 5] and old classic puzzles [6]. These definitions, variables, and constraints - including the variations of agent conflicts - are discussed in-depth [7] to establish appropriate baselines and avoid confusion.

Along with well-known heuristic algorithms for MAPF, randomized algorithms such as Monte-Carlo Tree Search have shown to be helpful for the same class of problems [8–10]. Also, MCTS has been studied well to solve logical puzzles, which indicates this method's importance in puzzle solving [11, 12]. We show that by specifying the parameters accurately, the *Numberlink* puzzle could also be resolved by the MCTS approach.

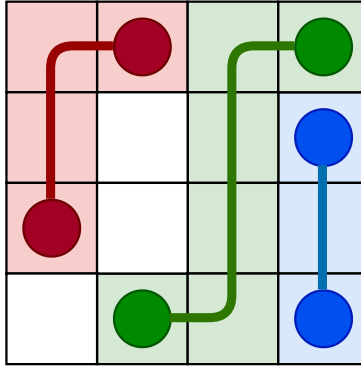


Fig. 1: Illustration of the *Numberlink* problem with three pairs of agents and specific goals with their connections.

The main objective of many well-known games could be deemed as a MAPF problem [13]. In this paper, the main goal is to investigate the possible solutions for the *Numberlink*. We tackle the same problem as a modification of the non-overlapping MAPF problem. *Numberlink*, also known as *Nanbarinku*, *Arukone*, and *Flow*, is an NP-complete [14] logic puzzle based on pairing up all matching numbers with separated lines that do not cross each other and do not cross a numbered square on an $m \times n$ grid. To be analogous to the display of *Flow*

Free game of *Numberlink*, we demonstrate the problem by distinct colours and a connected line with the same colour from the first cell of each agent to its destination. As shown in Figure 1, there are three pairs of agents and their goals with non-overlapping paths among them. Each agent and its goal have the same colour for demonstration, and evidently, the grid cells are coloured only by one colour or nothing.

This description could easily be formulated as a Multi-Agent problem with each agent representing a number. Moreover, the additional constraint where cells can not be reoccupied (a single agent only traverses each empty cell) gives the puzzle a vital characteristic that could be categorized as an unconventional modification of non-overlapping MAPF with new attributes. In most versions of the puzzle, every square is filled with a number (representing source/destination) or is crossed by a flow-line yielding a filled grid with no empty square at the end of the game. The earliest form of the *Numberlink*-style puzzle appeared in a column by Sam Loyd in Brooklyn Daily Eagle [14]. In Classic *Numberlink*, an additional constraint is added on the game, where each line should cross the fewest possible squares [15]. Also, other versions of the puzzle are similar to [16], which does not require filling all squares. This article intends to concentrate on the version of the *Numberlink* puzzle in which not necessarily all the squares are visited by the lines, and the chosen path does not necessarily use the shortest path.

In this article, the *Numberlink* as a pathfinding puzzle is defined by a mathematical description, and the puzzle is construed as a modified MAPF problem with specific conditions. Finding an optimal solution is almost impossible for the problem since it is NP-complete, as mentioned. So, we investigate randomized, approximate and heuristic methods to compare various solutions for the defined problem. As will be explored throughout the paper, the previously studied methods to approach such a problem fail to present an efficient solution when the size of the puzzle is scaled. Moreover, we highlight that many MAPF problems like the one on hand should be tackled with problem-specific methods to perform adequately. We propose an optimized MCTS-based method as an efficient and practical approach. Highlighting the presented method, we reiterate that it is imperative to utilize environment-aware modified algorithms to enhance the performance and efficiency of many MAPF problems and possibly other similar problems.

The rest of this paper is organized as follows: Section II gives the background knowledge required to address the solutions, separated into two parts. In the first part, we explain the structure of MCTS, and in the second part, the description of the SAT, applications, and solvers are studied. In section III, we discuss the mathematical formulation and the description of the *Numberlink* as a non-overlapping MAPF problem. Section IV studies Heuristic and SAT approaches to solving the *Numberlink* puzzle. The suggested approach to solve the described problem and employed optimizations are explained in section V. The experimental results of the proposed method and its comparison to other

promising methods can be found in section VI. Finally, the paper is concluded in Section VII.

2 Background

Many researchers have investigated diverse variations of MAPF by different assumptions and objectives. For instance, the assumption of whatever agents can pass through interconnection cells simultaneously or the objective of maximizing the captured goals in minimum time. These definitions, variables, and constraints - including the variations of agent conflicts - are discussed in-depth in [7] to establish appropriate baselines and avoid confusion.

Several algorithms for MAPF solutions have been proposed in various contexts. Fast MAPF solvers, Optimal algorithms, complete and incomplete solutions are the different classes of MAPF solvers with their specific characteristics. For example, Conflict-based search (CBS) [17] approach might give an optimal solution, where a Hierarchical Cooperative A* algorithm (WHCA*) [18] could solve the problem very fast, but results may not necessarily be complete and optimal. These different approaches for the MAPF could be categorized in the specification of the problem, objective, and resource criteria [19]. As another study, an algorithm selection for optimal MAPF is studied in [20].

2.1 Monte-Carlo Tree Search

MCTS algorithm is a simulation-based best-first search that employs Monte-Carlo methods to sample steps in a problematical approach [21]. As shown in Fig 2, MCTS included four main steps which are called *Selection*, *Expansion*, *Simulation*, and *Backpropagation* which have occupied in order in each iteration to find the optimal or near-optimal solution.

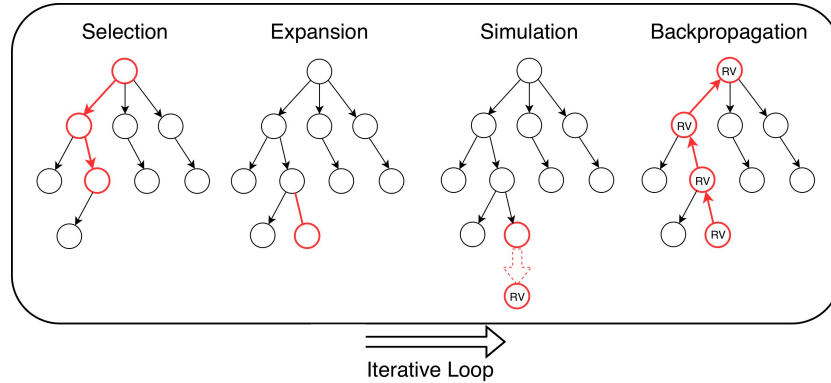


Fig. 2: Illustration of the steps of MCTS algorithm in each iteration.

- **Selection:** In this step, the algorithm starts to steer into the tree from the root and chooses the best *child* based on the *Tree Policy* among children until it reaches a leaf. *Tree Policy* can evaluate and determine the *Value* of each child.
- **Expansion:** In this step, if the selected node has a child, then the tree will be expanded. This process will be done for each iteration of the algorithm. It constructs the tree from the selected node. Moreover, It guarantees that all of the non-terminal children of the selected node being visited.
- **Simulation:** In the simulation, the algorithm randomly moves forward through the tree until it accomplishes a result. These random movements could be combined with some policies, which result in more accurate moves, and the algorithm could attain the results with better value faster. Value is a feature of terminal nodes measured by a function called *Value Function*, and it can specify how optimal the node is.
- **Backpropagation:** The algorithm tries to back up the value of the result obtained in the simulation section since the statistics of every node in the constructed tree needs to be updated by the new value in the new simulation.

2.2 Boolean Satisfiability Problem (SAT)

With the rise of the Boolean Satisfiability (SAT) solvers and their efficient realization in software, problems in different areas as diverse as software verification [22], planning [23], and even scheduling [24] are increasingly approached by SAT solvers as a general-purpose tool. SAT competitions have resulted in development of numerous of efficient implementations of these solvers [25], [26]. By employing propositional logic as the building block, SAT solvers provide general reasoning systems based on combinatorial and search platforms. SAT solvers' power appears when they are considered in applications that are not viewed as simple logical tasks. For instance, in a PSPACE-complete AI planning problem, an SAT reasoner could efficiently solve the problem. [27].

A *Boolean* or propositional formula is a logic expression defined over boolean variables with the values of a single bit *False*{0}, *True*{1}. A satisfying assignment for a boolean formula G is an assignment of σ to evaluate G to 1 under the mapping of $\sigma : G \rightarrow \{0, 1\}$. The Boolean Satisfiability Problem seeks to find a satisfying assignment for a given conjunctive normal form (CNF) formula. CNF is a conjunction formula (AND, \wedge) of clauses, where each clause is formed as a disjunction (OR, \vee) of literals. Each literal is either a boolean variable or its negation (NOT, \neg). Moreover, CNF is the generally accepted format for SAT solvers because of its simplicity and usefulness. Many problems are expressed as a conjunction of relatively simple conditions. As proved by Cook [28], this is known to be an NP-complete problem. All practical satisfiability algorithms (SAT solvers) strive to find such an assignment. In this work, we reduce the problem to a group of CNF expressions to be approached by SAT solver. We employ Z3 [25] theorem solver to represent and solve our problem.

3 Problem Definition

In this section, we explain the mathematical base of the *Numberlink* puzzle first. Then, we show how this puzzle is mathematically related to a general fundamental problem in computer networks, namely the Multi-Commodity Flow Problem.

3.1 Mathematical Foundations of *Numberlink*

Numberlink has proven to be NP-Complete [14, 29]. The exact definition of our problem which could be referred as a *MAPF with no-overlapping* is as follow: Consider a $K = \{k_1, k_2, k_3, \dots, k_n\}$ as a set, each specified by unique color and defined by $k_i = (s_i, d_i)$ where s_i and d_i are the source and destination nodes of the i^{th} path in a $m \times m$ grid $V = \{v_1, v_2, v_3, \dots, v_{m^2}\}$. The objective is to find a coloring function $\chi : V \rightarrow \mathbb{N}$ which satisfies the following:

$$\begin{aligned}
 \forall 1 \leq i \leq n : \exists \pi_i &= (v'_1, v'_2, \dots, v'_p), \\
 v'_1 = s_i, v'_p &= d_i, \\
 \forall v \in \pi_i : \chi(v) &= i, \\
 \forall 1 \leq j \leq p-1 : MD(v'_j, v'_{j+1}) &= 1
 \end{aligned} \tag{1}$$

Note that MD represents the Manhattan Distance, which is equal to 1 in our case. It indicates that the path of a specific agent should be connected.

3.2 Multi-Commodity Flow Problem

This problem could also be defined in the context of Multi-Commodity Flow Problem (MCFP). Given a flow network $G(V, E)$ where any edge $(u, v) \in E$ has the capacity of $c(u, v) = 1$. For n commodities (agents) $k_1, k_2, k_3, \dots, k_n$ defined by the triple (s_i, t_i, d_i) , where s_i and t_i are source and destination and $d_i = 1$ is the demand. Finding assignment of variable flow function $f_j(u, v)$ for any edge (u, v) , where $f_j(u, v) \in \{0, 1\}$ leads to a solution if the flow conservation law on transit lines (each node), source and destinations are satisfied and the capacity of the link are not overflowed (overlapped). These conditions are illustrated below:

$$\begin{aligned}
 \forall (u, v) \in E : \sum_{i=1}^n f_i(u, v) \cdot d_i &\leq c(u, v) \\
 \forall i, u \neq s_i, t_i : \sum_{w \in V} f_i(u, w) - \sum_{w \in V} f_i(w, u) &= 0 \\
 \forall i : \sum_{w \in V} f_i(s_i, w) - \sum_{w \in V} f_i(w, s_i) &= 1 \\
 \forall i : \sum_{w \in V} f_i(t_i, w) - \sum_{w \in V} f_i(w, t_i) &= -1
 \end{aligned} \tag{2}$$

Moreover, to apply the non-overlapping restriction, the capacity constraint should be extended to the vertex as follow:

$$\sum_{i=1}^n \sum_{w \in V} f_i(v, w) \leq c(v), \sum_{i=1}^n \sum_{w \in V} f_i(w, v) \leq c(v) \quad (3)$$

In other words, the amount of flow passing through a vertex cannot exceed its capacity ($c(v) = 1$). This restriction is also known as *Maximum flow with vertex capacities*. Heed that, in *Numberlink*; we do not mean the version with the additional condition which requires covering all of the nodes. This condition makes the problem utterly irrelevant to the MAPF. There have been some efforts to solve the problem [30], for instance, by proposing different algorithms based on 2-dimensional search [31]. However, no efficient algorithm is proposed when the size of the problem tends to large numbers.

4 Evaluated methods

In this section, we explain the methods that are promising to solve the defined problem. Since the number of constraints is high in this problem and some paths potentially can destroy other paths, we can not employ approaches that solve this problem locally. So, we implement two SAT-based and two MCTS-based procedures. We explain the advantages of these approaches more in detail in their sections. Moreover, heuristic methods are always nominated to solve a puzzle with specific constraints and relations since we can modify the heuristic function. So, we first investigate the two most promising heuristic approaches among most of them in this section.

4.1 Heuristic Approaches

This section investigates the solution proposed in [32] since its relatively comprehensive approach provides many algorithmic optimizations.

Note that the solution presented by [32] strives to solve any given *Numberlink* puzzle efficiently by filling up all cells in the grid, which is different from the version we intend to solve. However, the heuristics used in these methods could be similarly employed in the other version as well. In this approach, one could make use of a pruned backtracking search. A systematic mechanism is to arm a solver with different heuristics. As thoroughly discussed in [32], by using *Partial links* that are the cells that are not yet connected to other cells.

Among multiple ways to employ backtracking in *Numberlink*, the most straightforward method is to start at a source, choose a random path to the correspondent destination, and then perform the recursion process. Similarly, the solver can be initialized simultaneously on all sources, or as a naive approach, fill all cells from a particular starting cell in the grid, ignoring the source cells [32]. For instance, in the latter approach, the solver is initialized to fill the grid, starting in the upper left corner along the SW-diagonals. The order in which cells are

traversed is depicted in the figure 3 for a 4x4 grid. Instead of the diagonal search method, a row by row walk could also be considered. Even using conventional methods like a BFS (Breadth-First Search), the search will consequently lead to the same results.

A vital challenge in this approach is handling partial links. The inadvertent connection of a link to itself or different labelled sources should be prevented. This problem can be solved efficiently by the disjoint-set data structure [33]. Nevertheless, storing an array that holds the location status determines if the destination cell's current position would solve it. By applying the description discussed above and heuristics on corner cells, it is observed that solutions to the *Numberlink* can be represented as a group of number pairs, connecting source to terminal nodes.

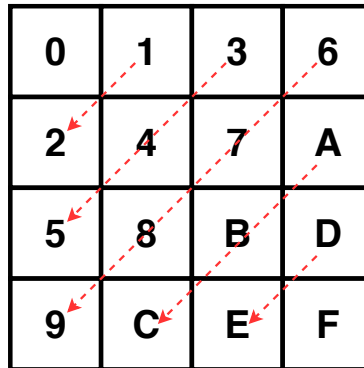


Fig. 3: Illustration of an example where the cells are visited systematically, ignoring source and destination nodes.

4.2 SAT-based Methods

As already mentioned, one convenient way to solve the puzzle is to represent the Boolean Satisfiability or SAT. A set of SAT variables and constraints carefully define a given *Numberlink* test case. This puzzle is correctly solved if an assignment of boolean variables leads to *True* evaluation of constraints. In the following, we implement and describe two SAT-based approaches for the *Numberlink* problem.

Simple SAT We describe a method employed by [34]; however, there are different approaches to solve *Numberlink* problem by SMT solvers. As discussed, we describe the puzzle by certain variables. Each cell in the $N \times N$ map is considered as a node associated with the i^{th} color (agent) where $i \in G = \{SetofColors\}$. For example, in a 3-colour map, node i is associated with only one of the three

variables of $\{blue, red, green\}$. Indeed, only one of these assignments could be true. This version assumes that all the cells are included in exactly one path from a source to a destination. Employing this requirement fills out the entire grid. As shown in Figure 4, edges on the nodes represent connections between cells in the puzzle. Nodes of each path are assigned to the same and precisely one colour. It is required to assign each cell to prevent *UNSAT* situation explicitly. So, for node $n \in N \times N$, an expression like below is defined:

$$\begin{aligned}
 & ((n = blue) \wedge (n \neq green) \wedge (n \neq red)) \vee \\
 & ((n = green) \wedge (n \neq blue) \wedge (n \neq red)) \vee \\
 & ((n = red) \wedge (n \neq green) \wedge (n \neq blue))
 \end{aligned} \tag{4}$$

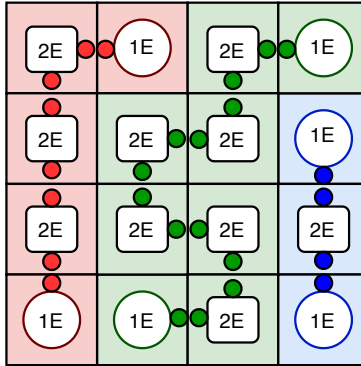


Fig. 4: Illustration of an example of the 4x4 puzzle with SAT constraints.

Terminal and connecting middle nodes of a path have only one and two connected neighbours of the same colour, respectively. One can exploit 2-D array structures in SMT theories to define these limitations. By representing the problem in such situations, we can pass it to an SAT solver and find the solution if it exists. Though, it may not be unique in some cases.

Improved SAT The previously described method tries to fill out the entire grid so that it might be inefficient. Moreover, it fails to perform accurate pathfinding in sparse grids, mainly if some holes exist in the grid. For demonstration, we describe another version of SAT representation indicated in Fig 5, which overcomes the mentioned drawback by utilizing minor tweaks to define constraints and variables.

4.3 MCTS-based Methods

MCTS first tries to find the shortest path, the lowest cost. However, finding the optimal solution is not contemplated as a rule in this problem. Therefore,

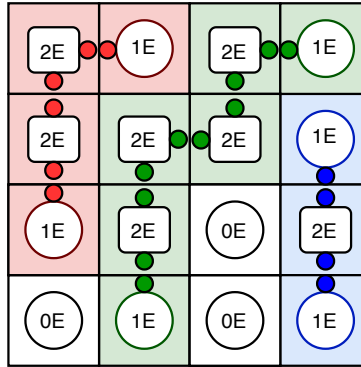


Fig. 5: Modified representation of a 4x4 puzzle with SAT constraints.

all the paths would not necessarily be the optimal or shortest one. Not only that but also achieving these kinds of solutions is thoroughly dependable with the conditions of the problem, which here are the maps of test cases, the initial positions of the agents, and the goals. Furthermore, this traditional problem is exerted in the *Flow Free* game. Thus, the approach is introduced here could be considered as an AI to solve this game.

Based on the definition of the problem discussed in the previous section, the non-overlapping MAPF on a $m \times m$ grid with n agents in this search space is considered. Each agent and its goal or desired destination in this problem specified by a unique pair of origin and goal cells. It is unnecessary to besmear every cell in the grid since the approach initially likes to find the best possible path.

MCTS is a prospective avenue to deal with this problem due to the convenience of the MCTS to perform well in problems with high time or (and) memory complexity. Nevertheless, in this case, classic MCTS has to construct and comply with a complicated tree with a high *Branch Factor*, which would precipitate that MCTS could not effectuate as it should. The erected tree's size would be b^{m^2} , where b represents the Branch Factor. Although due to the computation and memory limitations to explore the whole tree, the MCTS is suitable to approach these scenarios. As mentioned, MCTS needs to be improved to solve the emblazoned problem effectively. Thus, we should employ a new representation of the problem to let MCTS cope with it better.

We allude to a unique configuration of the $m \times m$ space to be a *ScreenShot* of the grid. Each *ScreenShot* defined by the idiosyncratic configuration of agents filling the 2-D grid space with paths and is a particular state of the algorithm struggling to solve the problem. In the method, each node initially represents a *ScreenShot*. A *Child Node* is related to its *Parent Node* whenever a single cell in the grid is different. To describe precisely, one of the agents moved toward its destination. The movement is not necessarily accurate, and one of the grid cells is only occupied. It is shown in the following relations:

$$\begin{aligned}
 &\exists v' \in V : \\
 &\chi'(v') \neq \chi(v') \\
 &\forall v'' \in V, v'' \neq v' : \chi'(v'') = \chi(v'')
 \end{aligned} \tag{5}$$

χ' represents a colouring function in a Child Node based on a definition explained in the problem definition section. In summary, equation 5 describes that a node can be a child of another if they differ in the colour of only one cell and other cells have the same colours in both nodes. Fig. 6 represents the construction of the tree and the relationships of nodes in the proposed algorithm. In this figure, a parent node and its children have been drawn. In each child of this node, a unique movement has occurred. The tree-shape relationship among the agents' manoeuvres utterly shows the flow of the algorithm that is trying to construct an efficient and accurate tree to accomplish its purpose.

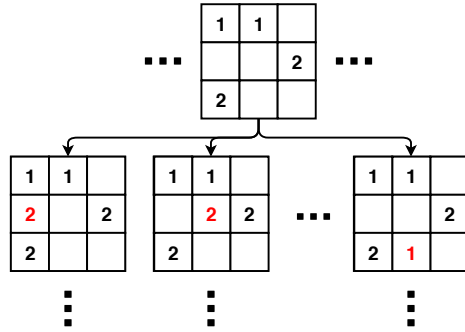


Fig. 6: Representation of the tree and relationships of nodes in the proposed algorithm [10].

In The explicated problem, there are many agents in the environment. Classic MCTS randomly chooses one of them and begins to direct it to its destination. Since the MCTS would arbitrarily opt for the direction, it would be practicable that the agent goes to each of its adjacent cells. Therefore, these two explained factors would yield a substantial value of Branch Factor, which considerably limits the performance and then the algorithm's accuracy. The new Branch Factor of the tree in this representation is $(b - 1) \times m^2$, which is larger than b . We consider other additional constraints for the nodes to overcome this situation and define a more advanced representation. In this representation, we store v and c , where $\chi(v) = c$ with the following description for its Child Node:

$$\begin{aligned}
 &\exists v' \in V, v \neq v', \chi(v') = 0 \\
 &\chi'(v') = c, MD(v', v) = 1
 \end{aligned} \tag{6}$$

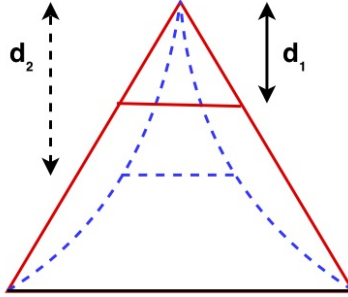


Fig. 7: Optimization in the tree exploration based on lower *Branch Factors* [10].

Note that χ' in (6) represents the coloring function in the Child Node. Relations in equation (6) describe that each Child Node is different from its parents in a single vertex v . v is a neighbour of a specific node v' while keeping the *Screen Shot* connected. This modification reduces the Branch Factor significantly since the algorithm would not randomly choose among all of the cells and only pick one of the cells, which is in the juxtaposition of one of the agents' current positions. So, the maximum value for b is 4 ($b \leq 4$). As another optimization, we strive to explore the nodes with lower Branch Factor [10]. Thus, starting nodes have a lower Branch Factor. This optimization is shown in Fig. 7. As depicted in Fig. 7, the explorer tree depth (d_2) in the modified version is increased, and thus the accuracy of the MCTS is improved. This modification does not ignore or suppress nodes in the MCTS tree since MCTS is gradually adapted to trace all nodes of the tree.

In the rest of this section, we explain the optimizations that we have employed in our work.

Average or Maximum in Backpropagation Due to the effects of the backpropagation in MCTS, the *Average* or *Maximum* of the children's values could be passed to their parents. We realize that the choice between these two procedures ultimately depends on the test case. It means that sometimes average is better than maximum, while in some test cases not. It is mainly because of the ratio of the number of factors to the size of the grid. If the number of agents is less against the grid (sparse grid), then the average method performs better since it solves the problem or goes forward in the tree gradually. Nevertheless, in a dense grid, the algorithm reaches the correct node abruptly. Along these lines, we decided to scrutinize both procedures during our test cases with the algorithm in two distinct models and utilize each of them depending on the test case. All of the arguments explored above have been shown in Figure 8. In the left search tree, the average of the children's values has been considered the value of their parent node, while in the right one, the maximum is chosen.

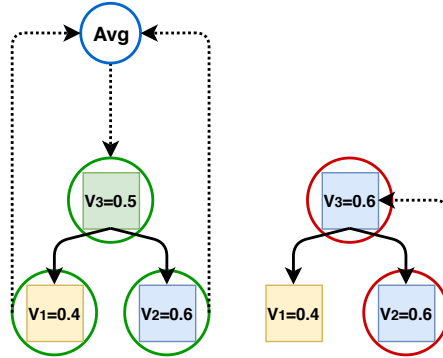


Fig. 8: Using average or maximum to propagate value of nodes through the search tree

Optimized Rollout Function The *Rollout Function* is a sub-stage of the MCTS algorithm that involves random tree exploration. In the simulation step of MCTS, the algorithm starts to explore the tree search by random movements. These movements need to be saved in a place temporarily. In our problem, it will create a grid for saving every movement from one node to another. Understandably, since every node of the tree is a state of the problem, each node is represented by a new grid with a size of $N \times N$ which causes a $\mathcal{O}(N^2)$ complexity for each node to build the state grid. Now, every child is different from its parent in only one cell. So, solely instead of constructing a new $N \times N$ grid for each state, a grid can be created and then modified by each node when it is created and craves to change the state of the problem [35]. In the Rollout function, storing all of the nodes' states is unnecessary since, for the evaluation, only the final state is needed. This optimization reduces the steering cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(1)$. Moreover, this technique is applicable in the scenarios where the child node differs from its ancestor insignificantly. In such applications, it is relatively easier to construct a new child by modifying the parent node.

Optimized Value Function Before explaining how we optimized the Value Function, it is crucial to consider that each agent tries to reach its destination by scrutinizing two prominent factors that lead the algorithm to solve the problem. First, the agent must reach its destination and establish the connection without violating the problem's rules. Second, the agent must contemplate that all other agents should reach their goals without violating any rules. Therefore, we define a coefficient called α that determines how much each agent should care about the two mentioned factors. To be more exact, we define three distinct values for α , which are 0, 0.5, and 1. If α is 1, agents consider other agents' success more important than themselves. If α is 0.5, agents care about the factors equally, and eventually, when α is 0, it means that agents crave to reach their goal first and then think about others' situations. After that, we define six different mod-

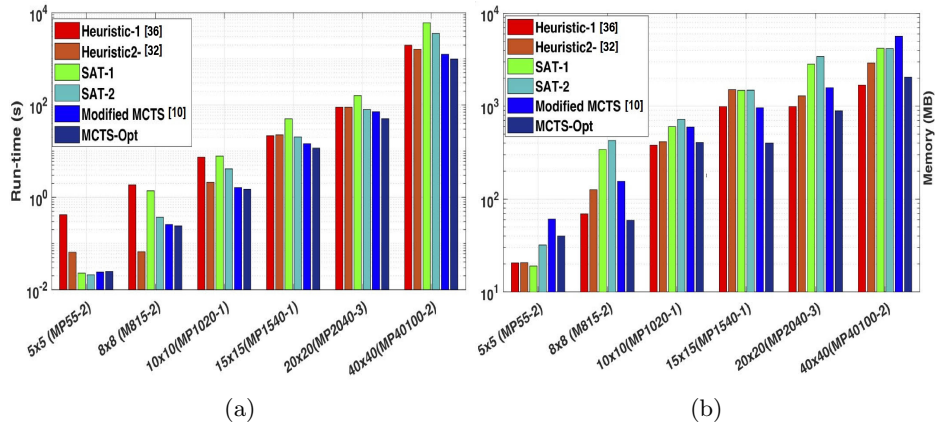


Fig. 9: Run-time comparison (a), Memory-consumption (b) among studied approaches

els with these three values of the coefficient and two different backpropagation model. Each model has one of them, and then, in each test case, one of these models performs better and ultimately depends on the test case features.

5 Experimental Results and Evaluation

In this section, the results of the proposed method have been indicated and then compared with other discussed methods.

5.1 Setup

We implemented our proposed MCTS and the SAT version of [10] with *Java 12* platform and *Z3 4.8* library, respectively.

5.2 Test Case Representation

We run the same test cases to avoid any concerns about executing different test cases with divergent difficulties. These test cases are divided into six major groups, with varying grid sizes as 5, 8, 10, 15, 20, and 40, as shown in the results table. Each grid size contains some distinctive number of agents to create different test cases with the same size and the peculiar number of agents. After the test cases were randomly generated with a *C#* program written by the authors of this paper, all of them were tested 50 times with our MCTS algorithm to ensure that they are not unsolvable.

Table 1: Evaluation of different algorithms for solving *Numberlink* puzzles

Grid-Size (m^2)	Instance	Agent No (Color)	Heuristic-1 [36]		Heuristic-2 [32]		SAT-1		SAT-2		Modified MCTS [10]		MCTS-Opt	
			Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)
5x5	MP53-1	3	0.391	18.4	0.059	15.4	0.013	14	0.011	21	0.037	83.9	0.021	52.9
	MP55-2	5	0.421	20.5	0.065	20.6	0.023	19	0.021	32	0.024	60.7	0.255	40.6
8x8	MP88-2	8	1.419	52.1	0.490	48.1	0.982	210	0.286	375	0.318	251.1	0.301	60.1
	MP810-2	10	1.562	58.1	0.519	70.16	1.183	292	0.318	427	0.295	210.2	0.229	45.2
10x10	MP815-2	15	1.872	69.1	0.676	126	1.391	341	0.371	480	0.259	155.1	0.240	59.1
	MP1010-1	10	5.238	128.4	1.261	150.6	4.182	546	2.850	638	1.351	450.2	1.291	149
15x15	MP1015-1	15	6.970	239	1.912	181.9	6.427	598	3.711	612	1.508	567.7	1.373	132.2
	MP1020-1	20	7.408	380	2.128	414	7.904	601	4.163	719	1.630	595.7	1.517	406
20x20	MP1515-1	15	12.719	1028	5.821	780	27.223	1202	14.131	1471	6.071	2108	4.155	810
	MP1520-1	20	17.782	1515	9.870	1355	41.841	1250	17.183	1383	9.118	2253	6.982	891
40x40	MP1540-1	40	21.661	987	22.621	1506	50.182	1481	20.471	1490	14.606	960.3	11.625	401.8
	MP2015-3	14	52.152	1690	38.721	1682	91.327	2104	48.551	2011	36.721	2344	20.104	1201
40x40	MP2020-3	20	68.122	1985	59.921	1739	112.41	2191	60.019	3028	42.332	2375	33.135	991
	MP2040-3	40	90.177	989	90.251	1290	160.21	2831	80.259	3429	71.343	1583	50.450	890
40x40	MP4010-2	10	319.88	2501	304.21	2210	741.09	4141	330.81	5881	143.789	7837	107.65	2591
	MP4020-2	20	570.54	5921	515.91	2730	1141.7	4239	618.82	6008	367.361	8541	259.76	3712
40x40	MP4050-2	50	1007.0	1802	928.52	3521	3083.8	4490	1251.9	5512	886.196	5281	699.51	1991
	MP40100-2	100	1980.4	1690	1626.0	2921	6013.7	4189	3541.4	4162	1257.686	5648	988.54	2051

5.3 Results and Analysis

Here, given the description of various approaches to approach the *Numberlink* puzzle, we present the run-time and memory consumption on extensive selected test cases with the specific properties described above. Note that the proposed MCTS algorithm with the explained optimizations performs much more efficiently in memory and compensates for the unnecessary delays in the classic MCTS implementation.

According to the results shown by Table 1, improved SAT solvers outperforms the MCTS method in small-size puzzles in terms of Memory and Run-time. The overhead of many simulation iterations in MCTS is slightly more time/memory-consuming than the whole tree search process. For clarification and making the results on Table 1 easy to read, we highlight each approach’s result entry that got the lowest memory consumption and execution time for each test case. As indicated in Table 1, most of the cells of MCTS-Opt, which is our proposed method, are bolded, particularly in the Time column. It means that this approach is outperforming other methods, especially for more significant test cases.

However, in the SAT solvers, the run-time increases dramatically when the puzzle and agents’ size are hiked and perform even worse than Heuristic approaches. There is an exponential hike observed in terms of memory consumption by increasing the grid size, specifically in the SAT methods. It could be explained since, in the SAT method, the whole grid and attributes for all of the cells are stored in every stage, which leads to a large memory consumption in large-scale grid test cases. On the other hand, it could be inspected that in MCTS methods, memory usage is confined even for extensive tests by employing proposed optimizations. Interestingly, by increasing the number of agents, memory usage is rising much more than other methods. It is basically because MCTS memory storage highly depends on saving the agents’ states, and increasing the

agent number increases the number of these states. Regarding the heuristic approaches, they generally tend to perform worse in small-size puzzles and better in larger grids than SAT in run-time efficiency. Moreover, Fig. 9 compares different approaches in terms of run-time and memory consumption on instances with maximum agent numbers. Based on Fig. 9, we see MCTS-Opt for grid size more than 10×10 solve the puzzles faster. It should be mentioned that the results are averages of fifty times executions of each method on each test case.

6 Conclusion

In this paper, we model the *Numberlink* puzzle, which is a traditional version of *Flow Free* game with a specified mathematical definition, as the non-overlapping MAPF problem. Afterwards, we study the problem to pinpoint promising possible approaches. We specify two SAT-based, two heuristics, and MCTS approaches as propitious ones to be studied and compared. Our final proposed method, MCTS-Opt, is based on MCTS due to its advantages, such as low time and memory complexity. The leverage about the time and memory complexity of MCTS is considerably remarkable since there are many agents, and each agent can choose to move forward in distinct directions. Consequently, this approach works suitably in the context of the problem regarding the size of the problem or, more precisely, the size of the tree search.

Even the classic MCTS has substantial errors with long execution time. Therefore, we propose several optimizations and modifications that help MCTS to solve the problem. These contributions indicate the impartial study and the effects of different enhancements applied to the proposed method by comparing them. One of our future works is to investigate and compare learning-based methods such as Reinforcement Learning (RL) better to visualize the non-overlapping MAPF problem’s possible solutions. Also, we would like to investigate more enhancements for the MCTS method to introduce new variations and examine them with the same proposed problem.

Finally, we present a comprehensive comparison of the possible solutions for the problem and showcase the effectiveness and efficiency of the enhanced MCTS. Results indicate that our approach outperforms other solutions in both Run-time Memory consumption in almost all the test cases, solving a large 40×40 grid with 100 agents in 988.5 seconds. Investigated and proposed solutions can potentially be employed to solve similar problems.

Acknowledgements

This research is connected to the GenZ strategic profiling project at the University of Oulu, supported by the Academy of Finland (project number 318930), and CRITICAL (Academy of Finland Strategic Research, 335729). Part of the work was also carried out with the support of Biocenter Oulu, spearhead project ICON.

References

1. Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning a. *Artificial Intelligence*, 155(1-2):93–146, 2004.
2. Shaull Almagor and Morteza Lahijanian. Explainable multi agent path finding. In *To appear in Int'l Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, 2020.
3. Hang Ma, Craig Tovey, Guni Sharon, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *Thirtieth AAAI Conf on AI*, 2016.
4. Yngvi Björnsson and Kári Halldórsson. Improved heuristics for optimal path-finding on game maps. *AIIDE*, 6:9–14, 2006.
5. Adi Botea, Bruno Bouzy, Michael Buro, Christian Bauckhage, and Dana Nau. Pathfinding in games. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
6. Pavel Surynek and Petr Michalík. The joint movement of pebbles in solving the $(n^2 - 1)$ -puzzle suboptimally and its applications in rule-based cooperative path-finding. *AAMAS*, 31(3):715–763, 2017.
7. Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.
8. Edward J Powley, Daniel Whitehouse, and Peter I Cowling. Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 234–241. IEEE, 2012.
9. Chun Yin Chu, Hisaaki Hashizume, Zikun Guo, Tomohiro Harada, and Ruck Thawonmas. Combining pathfinding algorithm with knowledge-based monte-carlo tree search in general video game playing. In *2015 IEEE CIG*, pages 523–529. IEEE, 2015.
10. M. S. Kiarostami, M. Reza Daneshvaramoli, S. K. Monfared, D. Rahmati, and S. Gorgin. Multi-agent non-overlapping pathfinding with monte-carlo tree search. In *2019 IEEE Conference on Games (CoG)*, pages 1–4, Aug 2019.
11. Mohammad Sina Kiarostami, Mohammadreza Daneshvaramoli, Saleh Khalaj Monfared, Aku Visuri, Helia Karisani, Simo Hosio, Hamed Khashehchi, Ehsan Futuhi, Dara Rahmati, and Saeid Gorgin. On using monte-carlo tree search to solve puzzles. In *2021 7th International Conference on Computer Technology Applications, ICCTA 2021*, page 18–26, New York, NY, USA, 2021. Association for Computing Machinery.
12. Mohammad Sina Kiarostami, Saleh Khalaj Monfared, Mohammadreza Daneshvaramoli, Negar Yousefian, Mahsa Massoud, Aku Visuri, Simo Hosio, Dara Rahmati, and Saeid Gorgin. Unlucky explorer: A complete non-overlapping map exploration. In *2021 3rd World Symposium on Software Engineering (WSSE 2021)*, 2021.
13. Ross Graham, Hugh McCabe, and Stephen Sheridan. Pathfinding in computer games. *The ITB Journal*, 4(2):6, 2003.
14. Aaron Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Zig-zag numberlink is np-complete. *Journal of Information Processing*, 23(3):239–245, 2015.
15. Kouichi Kotsuma and Yasuhiko Takenaga. Np-completeness and enumeration of number link puzzle. *IEICE Tech Report COMP2009-49, IEICE*, 2010.

16. Henry E Dudeney. *536 Puzzles and curious problems*. Courier Dover Publications, 2016.
17. Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.
18. David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.
19. Roni Stern. *Multi-Agent Path Finding – An Overview*, pages 96–115. Springer International Publishing, Cham, 2019.
20. Omri Kaduri, Eli Boyarski, and Roni Stern. Algorithm selection for optimal multi-agent pathfinding. In *Proceedings of the 30th Int Conf on Automated Planning and Scheduling*, pages 161–165. AAAI Press, 2020.
21. Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Int and AI in games*, 4(1):1–43, 2012.
22. Miroslav N Velev and Randal E Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. *Jrnl of Symbolic Compt*, 35(2):73–106, 2003.
23. Paul Stephan, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, 1996.
24. Carla P Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *AIPS*, pages 208–213, 1998.
25. Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
26. Bruno Dutertre. Yices 2.2. In *International Conference on Computer Aided Verification*, pages 737–744. Springer, 2014.
27. Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the national conference on artificial intelligence*, pages 1194–1201, 1996.
28. Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.
29. PPG Kramer and Jan Van Leeuwen. *Wire routing in NP-complete*, volume 82. Unknown Publisher, 1982.
30. Jingjin Yu and Steven M LaValle. Multi-agent path planning and network flow. In *Algorithmic foundations of robotics X*, pages 157–173. Springer, 2013.
31. Ryo Yoshinaka, Toshiki Saitoh, Jun Kawahara, Koji Tsuruma, Hiroaki Iwashita, and Shin-ichi Minato. Finding all solutions and instances of numberlink and slitherlink by zdds. *Algorithms*, 5(2):176–213, 2012.
32. Thomas Dybdahl Ahle. Program for generating and solving numberlink, 2012.
33. Bernard A Galler and Michael J Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.
34. Dennis Yurichev. Sat/smt by example, 2019.
35. Mohammadreza Daneshvaramoli, Mohammad Sina Kiarostami, Saleh Khalaj Monfared, Helia Karisani, Keivan Dehghannayeri, Dara Rahmati, and Saeid Gorgin. Decentralized communication-less multi-agent task assignment with cooperative monte-carlo tree search. In *2020 6th International Conf on Control, Automation and Robotics (ICCAR)*, pages 612–616. IEEE, 2020.
36. Ben Torvaneye. Solving the "flow free" game, 2019.