Multi-Agent non-Overlapping Pathfinding with Monte-Carlo Tree Search

Mohammad Sina Kiarostami*, Mohammad Reza Daneshvaramoli*, Saleh Khalaj Monfared*,
Dara Rahmati*, Saeid Gorgin‡*
*School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
‡Iranian Research Organization for Science and Technology (IROST), Tehran, Iran
{skiarostami, daneshvaramoli, monfared, dara.rahmati}@ipm.ir, gorgin@irost.ir

***Abstract*—In this work, we propose a novel implementation of Monte-Carlo Tree Search (MCTS) algorithm to solve a multi-agent pathfinding (MAPF) problem. We employ an optimization of MCTS with low time-complexity and acceptable reliability to approach the MAPF problems with no time constraint. To examine the efficiency and performance of the proposed approach, the *NumberLink* problem as a MAPF is investigated. We show that the addressed problem could be characterized as multi-agent pathfinding problem with no overlapping paths for the agents. Furthermore, we define this problem to be a simplified and special case of Multi-commodity flow problem (MCFP). Our MCTS solution utilizes a modified search-tree structure to efficiently solve the problem based on a 2-dimensional search space which performs in quadratic time complexity ($\mathcal{O}(m^4)$ where input size is m^2) and linear memory complexity ($\mathcal{O}(m^2)$). To evaluate our algorithm, we investigate the efficiency of the proposed solution for the well-known *Flow Free* puzzle. Our implementation solves a large 40×40 Numberlink puzzle in 21 minutes. To the best of our knowledge, there is no other efficient solution for this puzzle where the size of the problem is considerably large.**

***Index Terms*—Monte Carlo, Pathfinding, Multi-agent, Flow Free, Numberlink**

I. Introduction and Background

Monte-Carlo Tree Search (MCTS) is widely used in development of Artificial Intelligence's (AI) applications [1], [2]. This algorithm is successfully employed in specific pathfinding problems due to its low time-complexity and comparative reliability. MCTS procedure is conceptually simple which is thoroughly discussed in [3]. As tree-based approach, MCTS constructs a tree in a sequential and asymmetric manner at the beginning. At each stage of the algorithm, a tree policy is employed to find the most preferable node of the current tree. The tree policy strives to balance the considerations of exploration and exploitation. These considerations are handled by the defined parameters. A simulation is then run from the selected node and the search tree is updated according to the result. The *Child Nodes* are chosen and explorations are performed during this simulation according to the default policy. A great advantage of MCTS is that the values of intermediate states do not have to be evaluated. Furthermore, limited knowledge of problem is required since the solution is estimated by the reward criterion [3].

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

Multi-agent pathfinding (MAPF) problems are extensively used in AI applications, modern and commercial games [4], [5]. The general problem as initially defined by [6] is described by finding the collision-free paths for set of agents (K) which should be moved from $Source^i$ to $Destination^i$ where $i \in K$. This is simply illustrated in Fig 1. In this figure, two balls should not be collided when moved to their desire destination. Hence, based on time-spot state-machine a solution could be found to handle the problem (e.g the ball at the right side would wait until the other ball is successfully moved to its destination and then it is transported to its goal spot.) Moreover, the black colored squares are known to be obstacle in the map. Note that, in this problem a great degree of freedom is provided by time constraint which lets the agents to change their space configurations in time spots. The MAFP problems are shown to be NP-Complete [7]. There have been several proposed algorithms to solve MAPF problems [8], [9], [10]. Among all of these works, the heuristic algorithms have been conventionally used to solve the problems due to their simplicity. The A^* algorithm [11] is the most known algorithm to approach the application specific MAPF problems. However, these algorithms suffers heavy computational/time complexity for large scale and multi-agent problems and some propose MCTS methods to solve some instance of MAPF problems due to their efficient complexity and acceptable accuracy in many applicable different puzzles [12]. For instance, [13] gives a modified implementation of MCTS namely Monte-Carlo Fork Search (MCFS) which efficiently solves different puzzles like *N-Puzzle* where conventional solutions seem to be impractical in large-scale scenarios.

In this work, we have focused on special cases of MAPF problem. Here, the scope of the problem is defined with no time constraint. This means that there are no overlap between chosen paths of agents at any time. Hence, the problem is reduced to a single space configuration in single state machine with no freedom degree of time, which generally is a stronger case of the MAPF problem. A simple example of this problem and its solution is demonstrated in Fig. 2. This problem has been called *Numberlink* [14] and has proven to be NP-Complete [15], [16]. Note that, by *Numberlink* we do not mean the version with the additional condition which requires to cover all of the nodes. This condition makes the problem

completely irrelevant to the MAPF. There has been some efforts to solve the problem by proposing different algorithms based on 2-dimensional search [17]. However, no efficient algorithm is proposed when the size of the problem tends to large numbers.

The exact definition of our problem which could be referred as a *MAPF with no-overlapping* is as follow:
Consider a $K = \{k_1, k_2, k_3, ...k_n\}$ as a set of paths, 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 $V = \{v_1, v_2, v_3, ...v_{m^2}\}$. The objective is to find a coloring function $\chi : V \rightarrow \mathbb{N}$ which satisfies the following:

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

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

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) $k_1, k_2, k_3, ...k_n$ 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 $f_j(u, v) \in \{1, 0\}$ 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:

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

Moreover, to apply the no-overlapping restriction the capacity constraint should be extended to the vertex as follow:

$$\sum_{w \in V} f_{iv} \neq c(v)\ \forall v \in V \backslash \{s, t\} \tag{3}$$

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*.

For the sake of illustration, Fig. 2 represents a simple demonstration of the addressed problem and its solution. This figure also shows a simple configuration of the *Flow Free* game [18]. It's worthy to mention that any solution for a *Numberlink* puzzle is clearly an acceptable solution for a *MAPF* problem with the same configuration. To solve regular *Numberlink* naively, instead of space-time search in the conventional MAPF, only space search is performed to find the acceptable configuration for the paths. However, an exhaustive space search for an acceptable solution in this case, is infeasible in terms of computational/memory complexity for large-scale problems.

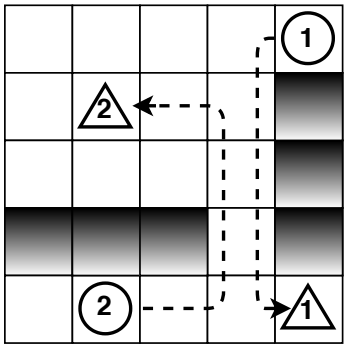

Fig. 1. Simple collision-free multi-agent pathfinding problem, circles are sources and triangles are the determined destinations.

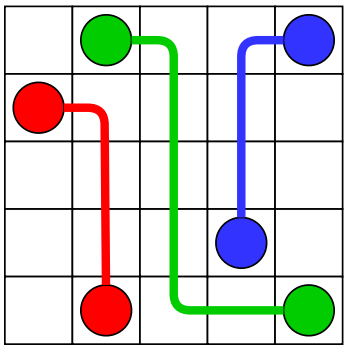

Fig. 2. Demonstration of a presented MAPF with no overlapping problem.

The remaining of this article describes the exact definition of the problem and the proposed solution, in section II. In section III, we present some evaluations of the proposed algorithm for different configurations of the *Numberlink* puzzle. Finally, we conclude the article and discuss the possible future works in section IV.

II. Proposed Method

In this section, we briefly describe the computational aspect of the problem to be solved and then discuss our proposed MCTS algorithm.

Based on the definition of the problem discussed in the previous section, the multi-agent non-overlapping pathfinding on a $m \times m$ grid is considered. With n agents in the grid search space, each specified by a unique pair of origin and goal cells. As described in (1), the path for agents should be connected. Moreover, there must not be any overlapping between the paths, since it violates the conditions in (1). Also, it worthy to mention that the paths are not necessarily the

shortest one which can be addressed as another optimization problem.

Most of the proposed MCTS-based solutions for pathfinding collision-free problems build a tree search to estimate the final correct answer [19], [20]. In the case of the addressed problem for a $m \times m$ grid, a huge tree is constructed leading to a tree size of b^{m^2}, where b represents the *Branch Factor*. Due to the computation and memory limitations to explore the whole tree, the MCTS is highly suitable to approach this scenarios. We refer a single configuration of the $m \times m$ space to be a *ScreenShot* of the grid. Each *ScreenShot* defined by unique configuration of agents filling the 2-D space with paths. In proposed MCTS, each node initially represents a *ScreenShot*. A *Child Node* is related to its *Parent Node* whenever a single cell in the grid is different. This is shown in the following relations:

$$\begin{gathered} \exists\ v' \in V : \\ \chi'(v') \neq \chi(v') \\ \forall v'' \in V,\ v'' \neq v' : \chi'(v'') = \chi(v'') \end{gathered} \tag{4}$$

Where χ' represents a coloring function in a *Child Node* based on a definition explained in (1).

Fig. 3 represents the construction of the nodes and tree relations in the proposed algorithm.

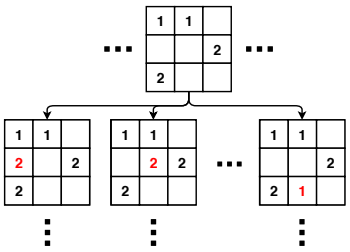

Fig. 3. Representation of the nodes and relation trees in the proposed algorithm.

This prospective yields to a large value of *branch factor*, which considerably limits the performance of the algorithm. To overcome this limitation, as other additional constraints for the nodes, we store v and c where $\chi(v) = c$ with the following description for its *Child Node*:

$$\begin{gathered} \exists v' \in V,\ v \neq v', \chi(v') = 0 \\ \chi'(v') = c, MD(v', v) = 1 \end{gathered} \tag{5}$$

Note that χ' in (5) represents the coloring function in the *Child Node*. Relations in (5) 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. In this case the maximum value for b is 4 ($b \leq 4$). As another optimization in the proposed MCTS, we strive to explore the nodes with lower *Branch Factor*. Thus, starting nodes have lower *Branch Factor*. This optimization is shown in Fig. 4. As depicted in Fig. 4, the explorer tree depth ($d2$) in the modified version is increased and thus the accuracy of the MCTS is

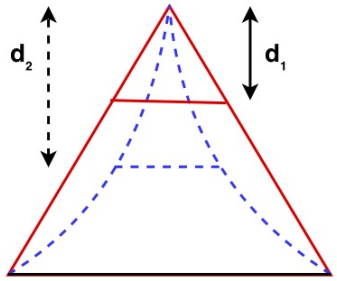

Fig. 4. Optimization in the tree exploration based on lower *Branch Factors*

improved. Note that this modification doesn't ignore/suppress nodes in the MCTS tree.

Algorithm 1 describes the main proposed MCTS. Each iteration of the procedure in algorithm 1 gives the best estimated *child*. The *computational budget* is the iteration value for the algorithm which is manually set to I. The *simulationResult* function at each iteration is performed in $\mathcal{O}(m^2)$. Moreover, the MCTS procedure is executed in $\mathcal{O}(m^2)$ in worst case scenario (full search in the problem space). Thus, the time complexity of the proposed MCTS algorithm could be calculated as follows:

$$\mathcal{O}(m^2) \times \mathcal{O}((m^2 + n) \times I) = \mathcal{O}(m^4) \tag{6}$$

Note that, in the above equation n represents the number of agents which appears due to the final search for all agents in the final stage of the algorithm.

Algorithm 1 Monte-Carlo Tree Search with No-Overlapping

```
1: procedure MCTS(root)
2:     while within computational budget do
3:         leaf = selection(root)
4:         expandedLeaf = expansion(leaf)
5:         simulationResult = rollout(expandedLeaf)
6:         backpropagation(simulationResult, expandedLeaf)
       end while
7:     return bestChild(root)
```

As for the memory complexity in our proposed algorithm, the program needs $\mathcal{O}(Im^2) = \mathcal{O}(m^2)$.

III. Evaluation

A. Setup

The hardware setup used in our evaluation is a machine running Ubuntu 16.04 equipped with an *Intel XEON E5 2697 V3 CPU* clocked at 2.6 GHz with 128 GB of DDR3 RAM. We have implemented the algorithm with Java version 1.8.

B. Results

The Table I represents our evaluation on different generated *Numberlink* instances. The puzzles are generated by the use of an algorithm proposed by [17]. The puzzles are chosen with different grid size and agent number to present a wide range of configurations. Note that the simulation is executed 50

TABLE I
PROPOSED MCTS EVALUATION ON DIFFERENT *Numberlink* PUZZLES

Grid Size(m^2)	Instance	Agents Num(n)	Full Search Space(n^{m^2})	Time(s)	Allocated Memory(MB)	Normalized Cost
5×5	MP53-1	3	3^{25}	0.037	83.9	3.1×10^{-6}
	MP55-1	5	5^{25}	0.024	60.7	1.4×10^{-6}
8×8	MP88-2	8	8^{64}	0.318	251.1	7.9×10^{-5}
	MP810-2	10	10^{64}	0.295	210.2	6.2×10^{-5}
	MP815-2	15	15^{64}	0.259	155.1	4.01×10^{-5}
10×10	MP1010-1	10	10^{100}	1.351	450.2	6.08×10^{-4}
	MP1015-1	15	15^{100}	1.508	567.7	8.5×10^{-4}
	MP1020-1	20	20^{100}	1.630	595.7	9.7×10^{-4}
15×15	MP1515-1	15	15^{225}	6.071	2108	1.2×10^{-2}
	MP1520-1	20	20^{225}	9.188	2253	2.07×10^{-2}
	MP1540-1	40	40^{225}	14.606	960.3	1.4×10^{-2}
20×20	MP2015-3	15	15^{400}	36.721	2344	8.6×10^{-2}
	MP2020-3	20	20^{400}	42.332	2375	0.1
	MP2040-3	40	40^{400}	71.343	1583	0.11
40×40	MP4010-2	10	10^{1600}	143.789	7837	1.12
	MP4020-2	20	20^{1600}	367.361	8541	3.13
	MP4050-2	50	50^{1600}	886.196	5281	4.68
	MP40100-2	100	100^{1600}	1257.686	5648	7.1

times for each instance and the results are averaged for each configuration. As shown in the table, the time and memory results are highly correlated to the configuration of the puzzles whereas some puzzles are relatively harder to solve. As a general rule, by increasing the number of the agents, the memory usage reduces surprisingly. This is due to the fact that the *Branching Factor* decreases when large number of agents is set for the problem which results in a more accurate exploration. This is also shown in figure 4.

As another interesting point, the evaluation shows a fair increase in memory-time consumption when the size of the problem tends to large integers. Also, as shown in the table, exponential increase in the size of the problem yields near-linear increase in the computation cost. Note that the computation cost, here is determined by normalized value of *allocated memory* $\times$ *execution time*.

IV. DISCUSSION AND CONCLUSION

In this work, we have proposed a novel implementation of Monte-Carlo Tree Search (MCTS) algorithm to solve a multi-agent pathfinding (MAPF). Our MCTS algorithm is empowered by different levels of optimizations to lower the time complexity and improve the reliability. To approach a MAPF problem with no time constraint, we have examined *NumberLink* problem as a MAPF problem. We have shown that this problem could be characterized as multi-agent pathfinding problem with no overlapping paths for the agents. Furthermore, our solution utilizes a modified search-tree structure to efficiently solve the problem based on 2-dimensional space-search which performs in $\mathcal{O}(m^4)$ and $\mathcal{O}(m^2)$ for time and memory, respectively. Rather than a significant decrease in the memory and time complexity, using MCTS implementations in the similar puzzles will require a minimum knowledge of the problem structures, since it inherently tends to find the optimum solutions based on the defined reward and cost functions. Our Implemented approach could also be applied to other different games such as *Line Puzzle: Pipe Art*, *Knots Puzzle* and *Draw Line: Classic*. As our future work, we would like to apply the MCTS with the proposed modifications to other MAPF problems to show the scalability and the efficiency of MCTS in other real-life applications. Also, as an interesting application this method could be studied to be applied as designer for traffic lines with minimum overlap.

ACKNOWLEDGMENT

The authors of the paper would like to express their special thanks of gratitude to Prof. Hamid Sarbazi-Azad, head of the school of computer science of IPM.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[2] D. Perez, S. Mostaghim, S. Samothrakis, and S. M. Lucas, "Multiobjective monte carlo tree search for real-time games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 347–360, Dec 2015.

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[4] D. Silver, "Cooperative pathfinding." *AIIDE*, vol. 1, pp. 117–122, 2005.

[5] M. Buro and T. M. Furtak, "Rts games and real-time ai research," in *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, vol. 6370, 2004.

[6] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1-4, p. 477, 1987.

[7] P. Surynek, "An optimization variant of multi-robot path planning is intractable," in *24th AAAI Conference on Artificial Intelligence*, 2010.

[8] S. Brand and R. Bidarra, "Multi-core scalable and efficient pathfinding with parallel ripple search," *computer animation and virtual worlds*, vol. 23, no. 2, pp. 73–85, 2012.

[9] Z. Bnaya, R. Stern, A. Felner, R. Zivan, and S. Okamoto, "Multi-agent path finding for self interested agents," in *Sixth Annual Symposium on Combinatorial Search*, 2013.

[10] D. Šišlák, P. Volf, and M. Pechoucek, "Accelerated a* trajectory planning: Grid-based path planning comparison," in *19th International Conference on Automated Planning and Scheduling (ICAPS), Thessaloniki, Greece, Sept*. Citeseer, 2009, pp. 19–23.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[12] J. Barraquand and J.-C. Latombe, "A monte-carlo algorithm for path planning with many degrees of freedom," in *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 1712–1717.

[13] B. Bouzy, "Monte-carlo fork search for cooperative path-finding," in *Workshop on Computer Games*. Springer, 2013, pp. 1–15.

[14] Nikoli, "Numberlink," 1989. [Online]. Available: http://www.nikoli.co.jp/en/puzzles/numberlink.html

[15] A. Adcock, E. D. Demaine, M. L. Demaine, M. P. O'Brien, F. Reidl, F. S. Villaamil, and B. D. Sullivan, "Zig-zag numberlink is np-complete," *Journal of Information Processing*, vol. 23, no. 3, pp. 239–245, 2015.

[16] P. Kramer and J. Van Leeuwen, *Wire routing in NP-complete*. Unknown Publisher, 1982, vol. 82.

[17] R. Yoshinaka, T. Saitoh, J. Kawahara, K. Tsuruma, H. Iwashita, and S.-i. Minato, "Finding all solutions and instances of numberlink and slitherlink by zdds," *Algorithms*, vol. 5, no. 2, pp. 176–213, 2012. [Online]. Available: http://www.mdpi.com/1999-4893/5/2/176

[18] Big Duck Games LLC, "Flow Free," 2012. [Online]. Available: https://www.bigduckgames.com

[19] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[20] M. Naveed, D. E. Kitchin, and A. Crampton, "Monte-carlo planning for pathfinding in real-time strategy games." PlanSIG, 2010.