International Core Journal of Engineering Volume 8 Issue 7, 2022
ISSN: 2414-1895 DOI: 10.6919/ICJE.202207_8(7).0127

Basic Optimal Solvers in Multi-Agent Pathfinding Problems: A literature Review

Tianxiao Xu[1], Yongqi Peng[2]

[1] School of Mathematics and Sciences, Tongji University, Shanghai 200092, China

[2] School of Mathematics and Sciences, Lanzhou University, Lanzhou 730000, China

*Corresponding author's e-mail: rickyxu1204@163.com

Abstract

Multi-agent pathfinding (MAPF) is an area of enormous research interest. Numerous techniques and algorithms under this topic have been put forward and continuously optimized. Recent researches in MAPF cover various subjects such as artificial intelligence, robotics, theoretical computer science and operations research. In this paper, these techniques and algorithms are examined and placed together in the broader context of the MAPF research domain. Same result might be exported through different techniques but the complexity and efficiency vary according to different circumstances. Finally, comparing them and further optimizing the algorithms are proposed in future work.

Keywords

Multi-agent Pathfinding; Machine Learning; Tree Method; Conflict-based; Reinforcement Learning.

1. Introduction

MAPF could be comprehended as follows. Multi-agent refers to a process of more than one simulation process, while pathfinding requires the agents find paths not colliding with each other spatially or temporally. As the progress going on, a cost function is calculated. It could be the total time usage of all agents, the total distance covered by all agents, etc., which is to be programmed and optimized by all algorithm. A* is a basic methods from which many MAPF algorithms find their inspiration, in which integer linear programming, tree method, various searching method are put into effect. Such algorithms typically cannot produce a solution within polynomial time[1]. In order to make it possible for computer to give an answer within an acceptable time, more means of machine learning on known patterns and topology changes are performed.

2. Two Kinds of Effective Methods for Small Grafts that are Dense with Agents

2.1 A*-Based: A Basic Search Method with its Continuation M*

A* search algorithm (A*) is an algorithm for finding the lowest cost of passage for paths with multiple nodes on the graph plane. The advantages of best-first search and Dijkstra's algorithm are combined in this algorithm: while performing a heuristic search to improve the efficiency of the algorithm, it is guaranteed to find an optimal path (based on the heuristic function)[2]. Similar to Dijkstra's algorithm, in A*, a priority queue is also used, but in this case the function f(n) is used as the priority, and the node with the lowest cost (f(n)) to the end is always chosen as the next node to be expanded and removed. The A* algorithm calculates the priority of each node by means of the following function. f(n)=g(n)+h(n). g(n) is the cost of node n from the starting point. And h(n) is a heuristic function that estimates the cost of the cheapest path from node n to the goal. Based on the actual problem, the

ISSN: 2414-1895 DOI: 10.6919/ICJE.202207_8(7).0127

choice of h(n) also differs, commonly used h(n) are Euclidean distance ($\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$), Taxicab geometry ($|x_1-x_2|+|y_1-y_2|$), Chebyshev distance ($\max(|x_1-x_2|,|y_1-y_2|)$) and so on (for N1(x1, y1) and N2(x2, y2)).

M* is similar to A*. In the expansion step, M* considers only the bounded neighbors of the node, a subset of the node's neighbors in the graph determined by the set of paths. To keep the collision sets up to date, the information about a collision is returned along all searched paths that reach the collision. The heuristic function is given in the referential paper[3]. The drawback of A* is obvious. The information of the processed nodes is stored into a linear or tree data structure, in which all states remained to be processed are located. Thus, the disadvantages of A* are obvious. Each agent selection affects other agents, so the total number of possible states increases exponentially as the number of agents increases linearly. These huge number of branches then make the computer's computing time and memory usage an intractable problem[4].

2.2 Constraint Programming(CP): Transforming Problems with Constraints into Resolvable Ones

Constraint Satisfaction Problem (CSP) is to represent the entities in its problem as a homogeneous set of finite conditions on variables, while Constraint Optimization Problem (COP) is a nonlinear programming problem with constraints. One of the algorithmic ideas of solving the above problems is Constraint Programming (CP). After being transformed into CSP or COP problem, the given problem would use a general constraint solver to find a solution. The main advantage of using CP is reflected in the Satisfiability problem. The constraint calculation algorithm that has been widely used is already very efficient under continuous optimization, and as of the 2019 study, the method can already solve the Satisfiability problem for over a million orders of magnitude of variables[1].

In the studied papers, the researchers tested the efficiency of different models for solving the same problem by limiting the number of variables entering the SAT solver and comparing the optimal search-based solver CBS. the results show that while each model is the fastest on some instances, it is the newer models with enhancements that stand out the most. If CBS solves a particular instance, then it tends to be the fastest. However, it solves the fewest instances overall within a given time constraint. The improvement of reducing the number of variables entering the SAT solver improves the above model. Also, it generates unsatisfiable instances to find the Makespan optimal solution, but then it generates satisfiable instances to minimize the sum of costs[5].

3. Two Kinds of Methods Effective for Large Grafts

3.1 Increasing Cost Tree(ICT): A Basic Tree Searching Method

With CBS algorithm mentioned in the next section, this algorithm is given as a basis. The basic method is as follows: the complete paths of different agents are nested with each other and finally form the complete solution of the whole problem. Such a search algorithm is accomplished by building two levels.

The high level is searched on a search tree, which is named the Increasing Cost Tree (ICT). In this tree, each node stores the cost of each agent's path at that node. First, in the high level, the tree is searched orderly (based on the value of specific functions[5]) to ensure the best solution is found first. The key point is as follows: the nodes in the ICT are flagged if they have been visited, every time a flagged node is traversed, the high-level stage calls the low-level stage to make a judgment. If valid solution represented by this node if found, the node will be flagged again. Second, when performing the low-level, the algorithm searches through the flagged possible solutions (which formed a space), where the cost of different agents is determined through the records in the nodes from the high level of ICT. At the end of the low-level phase, by finding the paths of all the individual agents, a feasible non-conflicting solution is discovered. Unlike the A* search, neither of the two levels of ICT make direct use of heuristic information. An effective pruning technique is also introduced in the study[5], which ensures that ICT nodes that do not represent any valid solution can be identified quickly.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202207_8(7).0127

3.2 Conflict-based-research(CBS): A Method Adding and Processing Constraints

When dealing with different MAPF problems, CBS is found to be very effective in some cases compared to other algorithms but ineffective in others[6]. In short, the influencing factors are as follows, for very narrow channel problems, A* (and ICTS) based algorithms can perform exponential work, while CBS can solve the problem much faster by applying and resolving a small number of conflicts.

The main point of CBS is that every time a set of constraints are added, paths that meets the requirement of all the constraints should be found[7]. If there is a conflict in the path of agents, it is invalid, and for this conflict, the conflict can be eliminated by adding a newly created constraint[8]. The procedure of CBS is also divided into two levels. As for the high level, the program finds conflicts, based on which constraints are added. At the lower level, paths are found for each agent that that the requirement of all the constraints.

The higher level of CBS is a search constraint tree (CT)[6]. a CT is a binary tree in which each node N contains constraints, a single solution that meets those constraints, and the cost of the solution. The set of constraints of the root of a CT stays empty. In CT, a child node inherits the constraints of its parent node, and adds a new constraint for an agent. The solution is found by the following search: when the solution of a node is valid, the node is a target node and the set of satisfying paths for all the agents does not conflict[9]. A best-first search is performed in the high level CBS among the above target nodes, and the best-ranking judgment condition is its cost. The algorithm is given in the referential paper[6,10].

4. Latest Methods Related to Machine Learning

4.1 Monte-Carlo Tree Search(MCTS): Basic Model of Machine Learning--monte-carlo Method for MAPF

Monte-Carlo Tree Search (MCTS) is one of the underlying algorithms widely used in Artificial Intelligence (AI). This algorithm has high accuracy and low time complexity, so it can be applied in specific pathfinding problems.

The MCTS program is conceptually simple. The focus of MCTS is to analyze the optimal action and expand the search tree randomly on the basis of some function in the space of searching [11]. The essence of MCTS is to make the game tree expand in the direction wanted to expand most by adjusting the parameters. In the case of a game competition, for example, in each competition the MCTS algorithm selects actions by randomly choosing them and carrying them to the end, and then uses the results finally happening in each competition to re-weight the nodes in the tree built in the game so that the possibility of better nodes being selected in future competitions would rise. Each round of MCTS consists of four steps: I. Selection: Starting from the root, and select child nodes in certain sequence until some leaf node is reached. II. Expansion: Unless this leaf node is the specific node that ends the game (e.g., win/lose/tie), create one (or more) child nodes and select a node from it. III. Simulation: Randomly proceed backwards from this selected node to the end (this randomization can be very simple). IV. Backward propagation: Use the final result of the match to loose the weight, bias, etc. in the nodes which have been gone through from the selected node to the final result. The algorithm is given in the referential paper[11].

As an algorithm with a tree as the underlying logic, MCTS uses an algorithmic strategy on a tree at each stage of the algorithm. In MCTS, a tree is constructed at the beginning in a sequential and asymmetric manner. At each stage of the algorithm, the most desirable node in the current tree is searched first. The strategy balances depth and breadth by defining parameters. The simulation is then performed starting from the selected node and the search tree is updated based on the results. During this simulation, the algorithm performs the selection and deferral of child nodes according to the aforementioned strategy. One of the most distinct advantage of MCTS is that the amount of state in the middle of the process does not need to be incorporated into the computation. In addition, due to

ISSN: 2414-1895 DOI: 10.6919/ICJE.202207_8(7).0127

the aforementioned reward rationing method, many of the detailed parameters of the problem are not needed in the computation.

4.2 Reinforcement-Based MAPF: Machine Learning Method Using Reinforcement Learning

Faced with the exponential growth of the state-action space, algorithms need to invest a large amount of training data. In this regard, many algorithms focus on decentralized policy learning, where each agent learns its own line of travel[12-14]. The efficiency of this training can be improved by the combined training of multiple agents. One method is to train some agents in order to predict the behavior of other agents. In most cases, some form of centralized learning is involved, in which the sum of the experiences of all agents can be used to make an inspiration for a certain action of all agents. When learning the network output centrally, training can be in a higher speed and a more stable state by sharing the weights and parameters of some layers of the neural network. Imitation Learning (IL) is used as a potential algorithm[15].

5. MAPF with Deadlines: Specific Occasions of Problems in Reality

In practical applications, as of time is a problem that all agents should consider. the target of this problem of MAPF-DL is to maximize the number of agents that reach a given target point from a given starting point within a given deadline, while avoiding mutual collisions. It can be shown that MAPF-DL is np -hard optimal solution problem. There are two types of algorithms for solving MAPF-DL[16]. The first class is the reduction of MAPF-DL to a flow problem and the corresponding linear programming formulation for which the reduction effect arises. The second class is a new algorithm based on combinatorial search. For each of these algorithms, the corresponding improvements can be made with the addition of DL. After an experimental comparison on several MAPF-DL instances, the results show that all algorithms are well adapted to the problem instance and each algorithm[17] outperforms the original algorithm in different scenarios[16].

6. Conclusion

This work discusses various solutions to the MAPF problem, where A* and CP are more efficient when facing smaller graphs and a larger number of agents, while CBS and ICTS are more efficient for larger graphs. MAPF combined with RL is another class of solutions to face this type of problems. Combined with the cooperation problem between agents and the latest time limit problem when dealing with practical problems, this work also finds that there can be different algorithms corresponding to them respectively.

References

[1] Stern R (2019). Multi-agent path finding–an overview. Artificial Intelligence: 96-115.

[2] Goldenberg M, Felner A, Stern R, Sharon G, & Schaeffer J (2012). A* variants for optimal multi-agent pathfinding. In: the 5th Annual Symposium on Combinatorial Search. Niagara Falls. pp. 157-158.

[3] Wagner G, Choset H (2011). M*: A complete multirobot path planning algorithm with performance bounds. In 2011 IEEE/RSJ international conference on intelligent robots and systems. San Francisco. pp. 3260-3267.

[4] Sharon G, Stern R, Felner A, & Sturtevant N R (2015). Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence, 219: 40-66.

[5] Barták R, & Švancara J (2019). On SAT-based approaches for multi-agent path finding with the sum-of-costs objective. In Twelfth Annual Symposium on Combinatorial Search. California. pp. 10-17.

[6] Sharon G, Stern R, Felner A, & Sturtevant N (2012). Meta-agent conflict-based search for optimal multi-agent path finding. In International symposium on combinatorial search. Vol. 3, No. 1: 97-104.

[7] Felner A, Stern R, Shimony S, Boyarski E, Goldenberg M, Sharon G, ... & Surynek P (2017). Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In International Symposium on Combinatorial Search. Vol. 8, No. 1: 29-37.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202207_8(7).0127

[8] Boyarski E, Felner A, Stern R, Sharon G, Tolpin D, Betzalel O, & Shimony E (2015). ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In: Twenty-Fourth International Joint Conference on Artificial Intelligence. Buenos Aires. pp. 740-746.

[9] Li J, Felner A, Boyarski E, Ma H, & Koenig S (2019). Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. Macao. pp. 442-449.

[10] Barer M, Sharon G, Stern R, & Felner A (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In: Seventh Annual Symposium on Combinatorial Search. Prague. pp. 19-27.

[11] Kiarostami M S, Daneshvaramoli M R, Monfared S K, Rahmati D, & Gorgin S (2019). Multi-agent non-overlapping pathfinding with monte-carlo tree search. In: 2019 IEEE Conference on Games. London. pp. 1-4.

[12] Okumura K, Tamura Y, & Défago X (2019). winpibt: Extended prioritized algorithm for iterative multi-agent path finding. https://doi.org/10.48550/arXiv.1905.10149.

[13] Devo A, Costante G, & Valigi P (2020). Deep reinforcement learning for instruction following visual navigation in 3d maze-like environments. IEEE Robotics and Automation Letters. Vol. 5, No. 2: 1175-1182.

[14] Sharon G, Stern R, Goldenberg M, & Felner A (2013). The increasing cost tree search for optimal multi-agent pathfinding. Artificial intelligence, 195: 470-495.

[15] Sartoretti G, Kerr J, Shi Y, Wagner G, Kumar T S, Koenig S, & Choset H (2019). Primal: Pathfinding via reinforcement and imitation multi-agent learning. IEEE Robotics and Automation Letters. 4(3): 2378-2385.

[16] Ma H, Koenig S, Ayanian N, Cohen L, Hönig W, Kumar T K, ... & Sharon G (2017). Overview: Generalizations of multi-agent path finding to real-world scenarios. https://doi.org/10.48550/arXiv. 1702.05515.

[17] Ma H, Wagner G, Felner A, Li J, Kumar T K, & Koenig S (2018). Multi-agent path finding with deadlines. https://doi.org/10.48550/arXiv.1806.04216.