Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

Investigation of Heuristic Search Techniques for Path Planning

Subramanian Raghavan
School of Computing and Engineering
University of Huddersfield
Huddersfield, United Kingdom

***Abstract*— Greedy Best First Search (GBFS) and A* search algorithms are studied for path planning by defining their core behavior, then explaining them with suitable examples. We compare both search techniques in terms of their accuracy, performance, and resource requirements. We also cover delete relaxation method and its impact on path planning.**

Keywords—Greedy best-first search, A*, best-first search algorithms, delete-relaxation, path planning

I. INTRODUCTION

In the field of Robotics, finding the optimal path from the current point in space to the destination or goal is the key for a field agent (i.e., Robot) to work effectively. The optimal path should ideally be also the shortest path to reach the destination if the limited precious resources (e.g., memory, power etc.) must be judiciously used. However, such ideal conditions fail to exist in real world where an autonomous, self-navigating, field agent must encounter unknown or inevitable hurdles such as humans, natural obstructions which it may fail to interpret correctly. Path planning, therefore, becomes essential in enabling a robot to do its intended job correctly and without encountering mishaps or straying away from the destination. An effective path planning should allow the robot to reach its destination in minimum time avoiding obstacles while also conserving its resources. There are various search techniques developed over the years that allow such path planning systems to be implemented. Of these, Greedy best-first search (GBFS) [1], A* [2] are generally considered as two of the important algorithms to search a given state-space. They are both variants of the best-first search algorithms. GBFS and A* are part of a group of algorithms whose search mechanism is unidirectional and whose heuristics expansion based. For domains where it is difficult to reach the goal from all states, best-first searches are usually preferred. Although they are both variants of best-first search algorithms, GBFS and A* work differently and are suited for different real-time scenarios. While choosing one over the other, the system designer needs to understand their core capabilities and their limitations before deciding on the right algorithm for path planning. This paper aims to provide insight into each of these algorithms by defining and explaining the core behavior of these two algorithms, with examples, while also comparing them. The investigation will also cover Delete relaxation as it is an important method to relax planning tasks that are created with GBFS and A*.

II. A* ALGORITHM: DEFINITION, EXPLANATION

A* is a best-first search which is represented as (1):

$$f(n) = h(n) + g(n) \quad (1)$$

where,

n=node (or state) at which A* will calculate f value

h(n) = heuristic function which is the distance left to cover between this node (n) and goal node

g(n) = cost incurred to reach this node n

In A* search, the least expensive path from the starting node to a given node 'n' is coursed. Whenever a node needs to be extended, the algorithm will save the cost incurred to reach the node (i.e., g(n)) along with an index to its parent node. The closer a node is to the goal, the more weightage it is given in comparison to other, far away nodes.

For a Heuristic function to be admissible or accepted, it is important that the function estimates the distance left to cover between this node (n) and goal node (g) correctly. This distance is the shortest distance between two points in a plane and given by the Euclidean formula. Thus, for a 2D plane, the formula to calculate the shortest distance becomes as given by (2):

$$h(n) = \text{SQRT}\left((n_x - g_x)^2 + (n_y - g_y)^2\right) \quad (2)$$

If the Heuristic function does not calculate this correctly, it is said to either *underestimate* or *overestimate* the distance and the calculated distance is not *optimal*. Such values which are not optimal are said to be not admissible.

This process continues till goal node is found (i.e., nodes cannot be expanded any further). The least expensive cost path from the starting node till the goal node is then found by tying back the indexes from the goal node to the starting node.

If C* is the expense incurred to arrive at the solution and one which is optimal and n be the given node, then it follows that: -

- if f(n) < C*, it implies that Node n will be expanded by A*, and
- if f(n) > C*, it implies that Node n will not be expanded by A*.

III. A* ALGORITHM: STRENGTHS AND WEAKNESSES

Assuming h(n) is admissible, it can be safely concluded that A* will always find the most optimal solution. That is, A* will always calculate the lower bound on the path to the Goal state. But this also means that A* relies on the effectiveness of the heuristic function for its efficiency. When faced with situations where heuristics plateau, A*'s performance relies

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

on the efficiency of its tiebreaker to identify the nodes which are least expensive to expand (i.e., least h-value). In almost every type of A* logic implemented; the challenge faced by planners has been with its performance-especially in universes which undergo noticeable change frequently. Specifically, considering the processing time of the A* algorithm, the count of nodes identified for the next move, and the length of their paths etc.

IV. GREEDY BEST FIRST SEARCH: DEFINITION, EXPLANATION

The simplest of algorithms falling in the category of satisficing search is the Greedy Best-First Search (GBFS). It is also the algorithm that is most considered by the planners.

Instead of finding a solution which is assured and optimal, A Greedy based search will try to identify the required search path in shortest possible time. If GBFS finds the solution, it stops search process at that node returning the solution. Else it will calculate the heuristic value and expand the child node whose heuristic value is lowest. That is, GBFS supposes that states that have lower values of Heuristics lie on a path that is least expensive to reach the nearest goal state. Thus, in every step, heuristic value is calculated and then compared to all the states generated till that point (but not expanded yet). This trade-off ensures that a possible search path gets identified quickly proving extremely beneficial sometimes.

GBFS's evaluation function(f(n)) is usually determined by the heuristic function(h(n)) and expressed as (3):

$$f(n) = h(n) \quad (3)$$

In other words, state which will be identified for expansion will be determined by the Evaluation function. The state giving the lowest such value is expanded. This process continues till the goal state is finally generated.

V. GBFS STRENGTHS, WEAKNESSES, AND WAYS TO IMPROVE ITS EFFICIENCY

Due to its search behavior (i.e., greedy), GBFS will never be able to ensure that the generated search plan is cost-optimal. Due to this inaccuracy GBFS may end up having states with same heuristic value.

For complex search problems, GBFS tends to generate a huge number of heuristic plateaus (i.e., heuristics with same values). This makes task of identifying the least expensive state to expand next difficult or accurate. In such scenarios, GBFS fails. For example, in Fig. 1 [3], let node S be the starting node. Let, G be the target node or the goal node. Assume the subtree of nodes be ST. Then, D shown in the figure correspond to the depth of ST. Let all nodes of ST have a Heuristic value of 6. Since GBFS considers only Heuristic value in its evaluation, it will search the entire ST before it decides to expand node N to reach G-even though all nodes in ST have the same Heuristic value of 6 (i.e., heuristic plateau). The greater the value of D, the more time GBFS will take to reach G. For instance, when D is 10, 1024 nodes would need to be expanded and when D's value increases to, say, 20, 10,00,000 nodes would have been expanded in ST as the search reaches N.

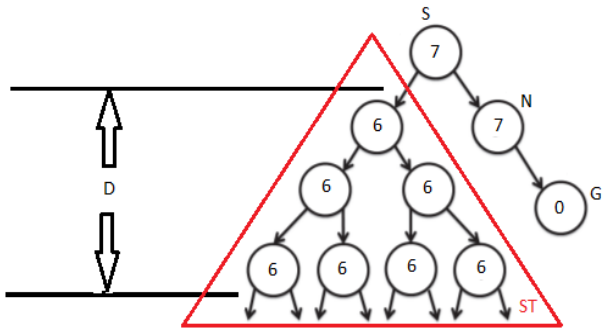

Fig. 1 Nodes and sub tree being evaluated in a path search

GBFS may not be considered a clearly stated algorithm but instead a collection of such algorithms (A* for instance) where the tie-breaking policy decides how to get differentiated.

If a planner based on GBFS fails to come up with an effective guided search, the problem could be usually with its tiebreaker or the implemented Heuristic. In such scenarios, the performance of GBFS degrades. As a solution, many times the standard search algorithm is enhanced by two methodologies namely "preferred operators" and "multi-heuristic best-first search" [4]. These methodologies make the algorithm better informed and as a result, the state space is searched more effectively. This process introduces variation in the original search by imparting knowledge to the standard algorithm and is also known to be "knowledge-based" [5]. The other way to make the GBFS more effective is by selecting the nodes at random and not the ones the Heuristic has identified. This is also known as "random exploration" [5].

VI. DELETE-RELAXATION: ANALYSIS, EXAMPLES, PROS AND CONS

A classical planning problem in STRIPS framework is represented as (4):

$$'\Pi = (Po, Ac, Is, Go) \quad (4)$$

where,

Π=tuple representing the planning problem (or task)

Po=set of facts or conditions (i.e., propositional variables)

Ac=set of actions or operations which is a triple of precondition (e.g., pre(ac)), add list (e.g., add(ac)), delete list (e.g., del(ac)), each a subset of Po

Is=Initial State (Is ⊆ Po)

Go=Goal (Go ⊆ Po)' [6], [7].

Conjunctive conditions on action, precondition and goal can consist of both positive and negative literals. However, in classical planning, delete-relaxation [8], [9] mentions action precondition and the goal will constitute only positive conjunctions. In the formula shown above, Conjunctive conditions can be represented as fact sets.

Delete relaxation considers delete lists as empty. Therefore, plans which follow such consideration are termed as relaxed plans. This means, negative "delete" literals [8], [9], e.g., logical *not*, will not be processed by a planner implementing delete-relaxation heuristic if they are present in either action precondition or problem goal description. The delete-relaxation heuristic function is usually denoted by "h+" [10]. It returns the cost calculated for an optimized relaxed

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

plan. It is considered as well-informed heuristic for classical planning.

To illustrate this, let us consider the car-driving example [7]. Let there be three locations the car can move to. Let these locations be A, B, C. Let us assume, to reach C from A, a car can only move on one-way in a line i.e., from A->B->C. Any car that moves on this path will consume one unit worth of fuel. The car has a tank that can only hold one unit of fuel-therefore refueling must happen in B. We know from (4), Π = (Po, Ac, Is, Go)

Therefore,

Po = {carA, carB, carC, fuel}

Is = {carA, fuel}

Go = {carZ}

Ac=a_{AB}{pre{CarA,fuel},add{CarB},del{carA,fuel}},a_{BC}{pre{CarB,fuel},add{CarC},del{carB,fuel}},a_{REFUEL}{pre{carB},add{fuel},del{}}. To be considered as a delete relaxation problem, del{carA, fuel} and del{carB, fuel} will have to be empty, i.e., del{}.

An example from PDDL planner's problem strips-gripper [11] is given below where it is seen that logical *not* conjunctions are not processed in delete-relaxation:-

`:effect (and (carry ?obj ?gripper)(not (at ?obj ?room)) (not (free ?gripper)))` – without delete-relaxation

`:effect (and (carry ?obj ?gripper))` – with delete-relaxation

Two major drawbacks of delete relaxation stem from the fact that it does not consider negative conjunctions i.e., ignores delete lists (as seen above). In many domains heuristic output does not change (i.e., its value remains the same) across states. For example, let us consider the logistics domain problem [4]. Under this example, the delete relaxation fails to account for the "moving back" action of the transportation vehicle. Once the vehicle starts charting the required course, it does not need to perform "moving back" action because it is in all the regions of the course at the same time.

Putting it in another way, let us say there is a vehicle which needs to move over a line of path twice - first to pick up items for delivery and then to traverse the same path back to deliver the items. Here, we can see that the value of the heuristic remains the same (i.e., line of path) until the vehicle reaches the point of actual pick up.

This means within states there are heuristic values that remain constant. The other drawback is "resource persistence" [12]. That is, planners implementing delete relaxation cannot track and control the consumption of resources that are limited and will not last forever. Frequently, planning activity involves identifying ways to make consumption of non-replenishable resources, such as energy and money, more economical. The planner should therefore find prudent ways to use such resources otherwise the quantity of resources it started with will not be sufficient to reach the desired goal. It thus becomes one of the important tasks of the planner to manage agents working with limited or non-replenishable resources. Relaxed plans (e.g., delete-relaxation) consider any resource to last (persist) for infinity. They are, therefore, not suitable for planning for agents which have limited resources (e.g., memory, power etc.).

VII. GBFS AND A*: A COMPARISON

A* algorithm ensures that the solution is optimal (assuming admissible heuristic), whereas GBFS favours conserving time over optimality and hence fails to guarantee an assured solution. This core difference between the two extend even when they are used with delete-relaxation heuristic as shown below:-

1. A* algorithm, unlike GBFS, requires the Heuristic to be both admissible and consistent. A* also assures optimality. But computing delete-heuristic (h+), which is optimal, is hard [10]. We also know that GBFS tends to expand the state with shortest path to the goal-and it does so without considering historical knowledge or any other criteria such as consistency and admissibility of the heuristic. Thus, h+ works much better with GBFS than with A*.
2. Since delete-relaxation works well for satisficing planning [10] and because GBFS is used in satisficing planning where A* isn't, delete-relaxation works better for GBFS than for A*.
3. When GBFS and A* both are used with delete-relaxation heuristic, GBFS may overapproximate the overall effort to reach the goal in comparison to A* and by a large margin [10].
4. GBFS requires lesser computer memory than A* for computing the search path. Because delete-relaxation heuristic does not factor in resource consumption while performing its calculations, it may consume more resources when working with A* algorithm. This may slow down A* even more than GBFS when either use delete relaxation heuristic.
5. When both algorithms are used with delete-relaxation heuristic, the quality of solution given by GBFS is inferior to given by A*.

VIII. CONCLUSION

An analysis of two common best-first search algorithms i.e., Greedy best-first search (GBFS) and A* was conducted. We identified that GBFS's focus is on finding the solution fast rather than guaranteeing it or making the solution optimal. We found out that there is high likelihood of GBFS failing to determine an effective search whenever the heuristics plateau-especially when the number of states to transition is high. Some recent techniques such as "preferred operators" and "multi-heuristic best-first search" were highlighted which tend to impart additional knowledge making GBFS better informed and more effective. The trade-offs to the optimality of A* were identified to be its relatively slower performance and higher resource requirements when compared to GBFS. The main drawbacks of delete-relaxation heuristics i.e., ignoring empty delete lists and an inherent inability to consider resources as finite, were showcased with car-driving and logistics-domain examples. The impact of delete-relaxation heuristic on GBFS and A* was analyzed while comparing characteristics of GBFS and A*. In general, delete-relaxation heuristic was found to work better with GBFS than with A*.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

REFERENCES

[1] J. E. Doran, D. Michie, and D. G. Kendall, "Experiments with the Graph Traverser program", *Proceedings of the Royal Society of London. Series A*, vol. 294, pp. 235-259, Sep. 1966.
[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, pp. 100-107, 1968.
[3] R. A. Valenzano, and F. Xie, "On the Completeness of Best-First Search Variants That Use Random Exploration", in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
[4] C. Domshlak, J. Hoffmann, and M. Katz, "Red–black planning: A new systematic approach to partial delete relaxation", *Artificial Intelligence*, vol. 221, pp. 73-114, Apr. 2015.
[5] R. Valenzano, N. Sturtevant, J. Schaeffer, and F. Xie, "A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration", in *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014, pp. 375-379.
[6] (2005) Stanford Research Institute Problem Solver. [Online]. Available: https://en.wikipedia.org/wiki/Stanford_Research_Institute_Problem_Solver
[7] M. Fickert, J. Hoffmann, and M. Steinmetz, "Combining the Delete Relaxation with Critical-Path Heuristics: A Direct Characterization", *Journal of Artificial Intelligence Research*, vol. 56, pp. 269-327, May. 2016.
[8] D. V. McDermott, "Using regression-match graphs to control search in planning", *Artificial Intelligence*, vol. 109, pp. 111-159, Apr. 1999.
[9] B. Bonet, and H. Geffner, "Planning as heuristic search", *Artificial Intelligence*, vol. 129, pp. 5-33, Jun. 2001.
[10] A. Torralba, and C. Croitoru, AI Planning 9. Delete Relaxation Heuristics, Part II: Pretending Things Can Only Get Better, A. Torralba, and C. Croitoru, Ed. Saarbrücken, Germany: Saarland University, Computer Science, 2018-2019.
[11] The PDDL Editor website. [Online]. Available: http://editor.planning.domains/#
[12] A. Coles, M. Fox, D. Long, and A. Smith, "A hybrid relaxed planning graph-LP heuristic for numeric planning domains", in *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 2008, pp. 52-59.
[13] H. T. Dinh, A. Russell, and Y. Su, "On the Value of Good Advice: The Complexity of A* Search with Accurate Heuristics", in *Proceedings of the National Conference on Artificial Intelligence*, 2007, pp. 1140-1145.
[14] M. Fickert, "A Novel Lookahead Strategy for Delete Relaxation Heuristics in Greedy Best-First Search", *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, pp. 119-123, Oct. 2020.
[15] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning : Theory and Practice*, Elsevier Science, Elsevier Ltd., 2004.
[16] E. Hansen, and R. Zhou, "Anytime Heuristic Search", *The Journal of Artificial Intelligence Research*, vol. 28, pp. 267-297, 2007.
[17] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, *An Introduction to the Planning Domain Definition Language*, Morgan & Claypool Publishers, 2019.
[18] M. Heusner, T. Kellar, and M. Helmert, "Understanding the Search Behaviour of Greedy Best-First Search", *Tenth Annual Symposium on Combinatorial Search*, vol. 8, pp. 47-55, Jun. 2017.
[19] J. Hoffmann, "Local Search Topology in Planning Benchmarks: An Empirical Analysis", *IJCAI International Joint Conference on Artificial Intelligence*, pp. 453-458, Jan. 2001.
[20] V. Martell, and A. Sandberg, "Performance Evaluation of A* Algorithms", B. CS. thesis, Blekinge Institute of Technology, Karlskrona, Sweden, 2016.