Life-long Planning

Setu Durgesh Vinay Annam (1217258939)
Hsin-Jung Lee (1216278531)
Arizona State University

Satrajit Maitra (1215097780)
Rishti Gupta (1217211814)
Arizona State University

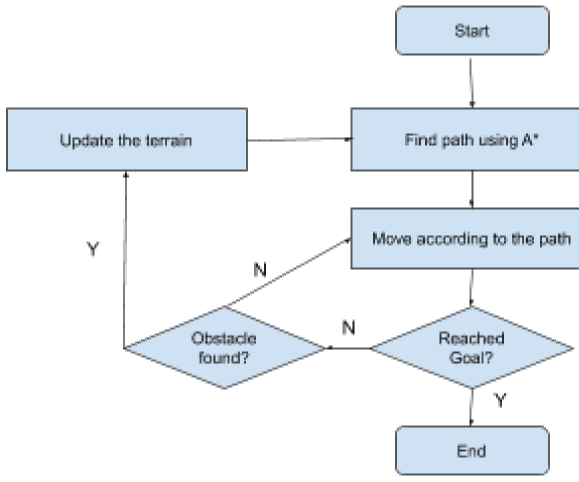
Abstract— **Incremental heuristic search methods use heuristics to focus their search and reuse information from previous searches to find solutions to series of similar search tasks much faster than solving each search task from scratch. In this project, we applied Lifelong Planning A* to pacman navigation in unknown terrain, including goal-directed navigation in unknown terrain and mapping of unknown terrain. We also implemented the D* Lite algorithm in the pacman domain. We compared the performances of Lifelong Planning A* and D* Lite.**

I. INTRODUCTION

Most traditional search methods reuse information from previous searches to find solutions to a series of similar search tasks much faster rather than solving each search task from scratch as mentioned in (Frigioni, MarchettiSpaccamela, & Nanni 2000).
Heuristic search methods, such as A* [1] (Nilsson 1971), is a best-first search algorithm, which uses the heuristics ($h(x)$) and traversal costs ($g(x)$). The two values are obtained from the expanded nodes, which maintains a priority queue representing a set of nodes of the graph which is to be traversed. The nodes are expanded in the same order as they are present in the queue. It makes the search for a path faster than the algorithms that do not use heuristics or any previous information from the graph.
LPA* or Lifelong Planning A* [1], is a modification on A* that maintains information between successive searches, given the start node and the end node of the route do not change, that is they remain static. Thus, LPA* does not compute all information for the graph from the scratch. For this, it maintains an extra variable for traversal costs. It checks the nodes consistency, using heuristics to detect the relevant nodes for new route computation. In this project, we apply LPA* to pacman navigation in unknown terrain. The pacman could use conventional graph-search methods when replanning its paths after discovering previously unknown obstacles. However, the resulting planning times can be on the order of minutes for the large terrains that are often used, which adds up to substantial idle times (Stentz 1994).
Building on LPA*, we also implemented D* Lite, a novel replanning method that is derived from LPA* by exchanging the start and goal vertex and reversing all edges, which can exempt already traversed nodes in the graph to be traveled again. D*Lite updates its priority queue without constant reordering, which helps for a fast and optimized response. This feature is derived from D*, an algorithm which is very similar to A*. The only difference is that the edge costs are variable.
To gain insight into its behavior, we presented various performance comparisons between LPA* and D* Lite. Our theoretical properties show that LPA* is efficient and similar to A*, a well known and well understood search algorithm. Our experimental properties show that D* Lite is at least as efficient as D*. We also present an experimental evaluation of the benefits of combining incremental and heuristic search across different navigation tasks in unknown terrain, including goal-directed navigation and mapping.

II. IMPLEMENTATION COMPARISON

A* [2] creates the start and goal node and inserts them into the map. The keys are created for the nodes. These keys are inserted in the proper priority queue and the g(goal) node is initially set to ∞. The map and the queue are initially empty. The s(start) node is set to *open*, and the node with the smallest key is selected for expansion. If the selected node for expansion is the goal node, one last update is done to check the neighbors and then the g(goal) node is set to *closed* and the algorithm terminates. Otherwise, the status of the node being expanded is set to *closed*, its previous node (backpointer) is updated, the value of f, g *and* h of its neighbors is updated and the priority queue is modified to follow the adjustments made in the map. Then, the node with the smallest key is selected for expansion by the algorithm in a loop procedure and it is checked at each iteration, whether the node selected is the goal node or not. A* can be extended to an unknown terrain (dynamic environment) by simply replanning the path whenever it sees an obstacle. While replanning, the agent chooses the current location as the start node and searches the goal.



LPA* [2] initializes the start node and sets its key, $g_{start} = \infty$, $rhs_{start} = 0$, and remaining all are set to ∞. A priority queue is used as the fringe and initially the key is inserted in the priority queue. To maintain consistency, the key with the lowest value in the queue is smaller than the key of the goal or $g_{goal} \neq rhs_{goal}$, the key with the lowest value maintains consistency, the neighbouring nodes updates their rhs values, their back pointers and updating the priority queue with the inconsistent vertices. When obstacles are found, the edge cost changes, leading to the changes in consistency, rhs and back pointers are made for those nodes that are affected. This results in the calculation of other routes. This way LPA* maintains a terrain map and fringes from successive searches for static start and goal nodes.

D* Lite [2] initially sets the following values: $goal = start$, $km = 0$, $g_{goal} = \infty$, $rhs_{goal} = 0$. The priority queue is initially empty. The key for the goal node is inserted in the queue. The start node is created and this is set: $g_{start} = rhs_{start} = \infty$. While the lowest key of the queue is smaller than the key of the start node or while the rhs_{start} differs from g_{start}, the algorithm checks the consistency and updates the node with the lowest key. The rhs values and the previous nodes(backpointers) of its neighbors will also get updated. This while-loop is executed till a route is calculated. Then, the start node varies till the goal is reached. If obstacles are detected by the agent in the path, these are modified: $km = km + h(last, start)$, $last = start$ and the affected nodes are updated. This changing of km helps directly in future computation of keys and plays a vital role to avoid unnecessary reordering of queues and thus guarantees optimal paths. The while-loop mentioned previously is executed again to compute a new path. Hence, D*Lite is able to maintain maps and queues for successive calculations with dynamic start nodes and static end nodes.

III. TECHNICAL APPROACH

Lifelong planning A* search is based upon the Sevn Koenig and Maxim Likhachev algorithm. We are implemented based upon Seven Koenig and Mxim Likhachev pseudocode. The pseudocode uses the priority queue. U.Top() returns a vertex of the smallest priority of all elements in the priority queue. U.TopKey() return the smallest priority of all vertices in priority queue U. U.Pop() returns the vertex and delete the vertex with the smallest priority in priority K. U.Update (s,k) changes the priority of vertex s in priority queue U to k. U.Remove(s) removes the node s.

Here is the brief procedures to implement lifelong planning A* [1]:

CalculateKey(s):
The function returns the key. The key is defined as $min(g(s), rhs(s)) + h(s); min(g(s), rhs(s))$.

Initialize():
The function sets start and goal to be infinity, sets the priority queue to be empty and sets all the rhs values of the nodes to be = 0. The function also inserts the start node along with its heuristic value into the queue.

Update Vertex(u):
The function removes (pops) the lowest key value node from the priority queue. If the node chosen for expansion is not the goal node, its neighbor's values are modified and inserted in the priority queue.
ComputeShortestPath():
The function computes the shortest path considering the heuristic values whenever the agent re-plans and returns the same.

Main():
The main function is a runner that is used to call other functions in order to calculate the required path and do the replanning whenever the environment changes.

Here is the brief procedures to implement D* Lite [1]:

CalculateKey(s):
The function returns the key. The key is defined as: $min(g(s), rhs(s)) + h(s_{start}, s) + km; min(g(s), rhs(s))$.

Initialize():
The function sets start and goal to be infinity, sets the priority queue to be empty, $km = 0$ and sets all the rhs values of the nodes to be = 0. The function also

inserts the start node along with its heuristic value into the queue.

Update Vertex(u):
The function removes (pops) the lowest key value node from the priority queue. If the node chosen for expansion is not the goal node, its neighbor's values are modified and inserted in the priority queue.

ComputeShortestPath():
The function computes the shortest path considering the heuristic values whenever the agent re-plans and returns the same.
M*ain():*
The main function is a runner that is used to call other functions in order to calculate the required path and do the replanning whenever the environment changes.

Application on Pacman Domain:
In pacman domain, the cost of every action is 1. The pacman can move in four directions. For every algorithm, initially the pacman knows only about the size of the layout and the boundary walls of the environment. At any time the pacman has the information about its current locations and its neighbouring nodes' latest information. It can know its own location, so It maintains the memory of the obstacles and avoids them whenever it encounters them again. For the implementation sake, we adopted from the position search agent from project 1 code. A position search agent will try to move towards the goal location, based on the search algorithm implemented. We are defining the goal location as a single food item placed as a goal location. To analyze the implementation of the algorithms we are using the layout shown in the source paper of the algorithm. To find the path chosen by the pacman, we use the red to gray gradient on the layout, red representing the start and gray representing the end. To represent the obstacles that caused the pacman to replan are shown in green boxes.

IV. RESULTS

A* Search:
Fig 1. Depicts the pacman moving towards the goal in accordance to A* algorithm.

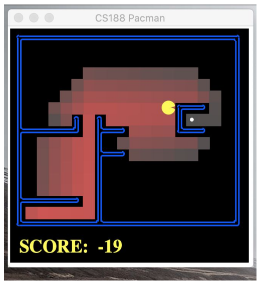


Fig. 1. Implementation on pacman (A*)

Nodes Expanded	110
Score	481
Path Cost	29

Table I: Results of A*

A* Replanner (Dynamic A*):
Fig 2. Depicts the pacman moving towards the goal in accordance to Dynamic A* algorithm.

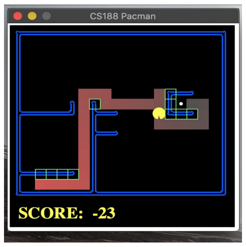


Fig. 2. Implementation on pacman (Dynamic A*)

Nodes Expanded	611
Score	477
Path Cost	33

Table II: Results of Dynamic A*

The A* algorithm finds the shortest path from the start to the goal. In the first iteration the manhattan displacement is the path chosen by the pacman if no obstacles are found in the way. Dynamic A* algorithm will help the agent to detect the obstacle along the path, and thus will continue moving along the manhattan path after it tackles the obstacle. The obstacles found are always in the minimum heuristic distance path. For this setting the A* algorithm expands 110 nodes, and the shortest path is same as that of the d* lite. But in a dynamic environment the A* replanner expands the nodes in the order of the number of obstacles it finds.

LPA* Search:
Fig 3. Depicts the pacman moving towards the goal in accordance to LPA* algorithm.

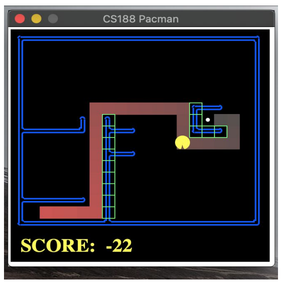


Fig. 3. Implementation on pacman (LPA*)

Nodes Expanded	2907
Score	479
Path Cost	31

Table III: Results of LPA*

The LPA* algorithm finds the shortest path from the new start to the goal points when it sees an obstacle. There are 15 obstacles found and the start point is changing every time the obstacle is found. Since there are more obstacles and the expansions are also high. Optimally the LPA* expands approximately twice the number of expansions than that of a d* lite algorithm.

D* Lite Search:
Fig 4. Depicts the pacman moving towards the goal in accordance to D* Lite algorithm.

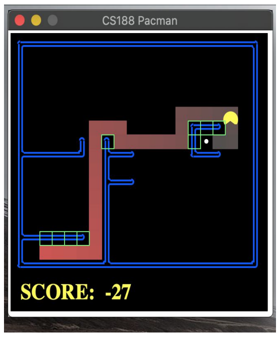


Fig. 4 Implementation on pacman (D* Lite)

.

Nodes Expanded	898
Score	479
Path Cost	31

Table IV: Results of D* Lite

D* Lite algorithm finds the shortest path from the new start to the goal point when it sees an obstacle, from the goal to the new start state that is the current state. The optimal D* search path will be similar to that of the A* algorithm. The node expansions are also comparable to that of the A* dynamic search.

Comparing the performance of LPA* and D* Lite:
These both are used for incremental searching but the main difference is the start and goal states are constant in the LPA*. The g values are adjusted for the shortest path when an obstacle is found and the edge costs are updated. Whereas the D* Lite will move to the new start state, where the obstacle is found and the replanning is done for the goal to the new start point. This has less unexplored locations in the terrain than in the case where search is from start to goal. This improvement is called the *Search Direction*. Another improvement is handling the *Priority Queue.* Since, the D* Lite algorithm moves the agent to a new point, the h values are updated to all the nodes. This affects the key values stored in the priority queue. To take care of these changes in the priority queue, k_m is used as an accumulated h for all the nodes.

V. CONCLUSIONS & DISCUSSIONS

In this project, we have implemented D* Lite and A* (D*). [1] D* Lite is a novel replanning method for robot navigation in unknown terrain that implements the similar navigation strategies as in dynamic A* (D*). Both D* Lite and A* use heuristic to focus the search and search from the goal vertex towards the current vertex of the robot and use similar ways to reduce the reorder of the priority queue. D* Lite builds on Lifelong planning A*. It is very strong similarity to A*. D* Lite is different with D*. D* Lite provides a solid algorithmic foundation for research on replanning methods in artificial intelligence and robotics and incremental search methods.

VI. TEAM EFFECTIVENESS

The goal was accomplished by equal and divided efforts of the whole team. The group under the support and guidance of Dr. Tony Zhang worked to compare results of three different algorithms, namely Dynamic A*, LPA* and D*Lite. The team decided to split up the responsibilities to achieve the required objective early and efficiently. The paper provided by the professor was read by everyone, and was taken up by one of our team members to implement. One member managed the layouts to be tested. The other two members handled the report to be submitted. Thus, the divided efforts of the team helped us to fulfil the necessary.

VIII. ACKNOWLEDGMENTS

We are grateful to Prof. Tony Zhang for his constant encouragement and support towards the project. We would also humbly extend our thanks to all the team members and TA's whose constant hard-work and persistence brought a successful end to the project.

REFERENCES

[1] Koenig, S., & Likhachev, M. (2002). D^* lite. Aaai/iaai, 15.
[2] da Rosa, W. C., de Bessa, I. V., & Cordeiro, L. C. (2016). Application of Global Route-Planning Algorithms with Geodesy. arXiv preprint arXiv:1610.04597.
[3] Stentz, A. (1995, August). The focussed d^* algorithm for real-time replanning. In IJCAI (Vol. 95, pp. 1652-1659).