Lazy Lifelong Planning for Efficient Replanning in Graphs with Expensive Edge Evaluation

Jaein Lim
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0150
Email: jaeinlim126@gatech.edu

Siddhartha Srinivasa
School of Computer Science & Engineering
University of Washington
Seattle, WA, 98195-2355
Email: siddh@cs.uw.edu

Panagiotis Tsiotras
School of Aerospace Engineering
Institute for Robotics & Intelligent Machines
Georgia Institute of Technology
Atlanta, Georgia 30332–0150
Email: tsiotras@gatech.edu

***Abstract*— We present an incremental search algorithm, called Lifelong-GLS, which combines the vertex efficiency of Lifelong Planning A* (LPA*) and the edge efficiency of Generalized Lazy Search (GLS) for efficient replanning on dynamic graphs where edge evaluation is expensive. We use a lazily evaluated LPA* to repair the cost-to-come inconsistencies of the relevant region of the current search tree based on the previous search results, and then we restrict the expensive edge evaluations only to the current shortest subpath as in the GLS framework. The proposed algorithm is complete and correct in finding the optimal solution in the current graph, if one exists. We also show the efficiency of the proposed algorithm compared to the standard LPA* and the GLS algorithms over consecutive search episodes in a dynamic environment.**

I. Introduction

The ability to replan quickly and reliably is essential for every decision-making agent with partial knowledge operating in a dynamic or complex environment. Incremental search methods [1]–[3] allow replanning by storing the previous search tree in order to identify the inconsistent portion of the tree when the graph changes in order to efficiently repair the current tree. Any identified inconsistencies are propagated onward to make the search tree consistent again with respect to the current graph changes without having to solve the problem from scratch. In particular, Lifelong Planning A* (LPA*) [2] efficiently restricts repairs to only the optimal path candidate guided by a consistent heuristic and a priority queue similar to that of the A* algorithm [4]. This means that LPA* heuristically delays the expansion of inconsistent vertices until repairing becomes necessary in order to find the new optimal solution with respect to the current graph.

Unfortunately, the design of LPA* is tailored to reducing the number of vertex expansions to find the new optimal solution, and it is indifferent to the number of edge evaluations. LPA* evaluates all changed edges before repairing propagation commences, regardless of whether these changes are relevant to the current problem or not. Figure 1 illustrates this point. This property of LPA* often results in unnecessarily excessive edge evaluations to find the new optimal solution, causing significant overhead in problem domains where edge evaluation dominates computation time. For example, in motion planning problems [5]–[8], an edge evaluation consists of multiple collision checks in the configuration space, solving two-point boundary value problems, or propagating the system dynamics with a closed-loop controller.

The issue of excessive edge evaluations has been explicitly addressed within the lazy search framework in order to reduce the actual number of edge evaluations by delaying these evaluations as much as possible [9]–[15]. The main idea of the lazy search framework is to delay the actual evaluation of the edges using a n-step lookahead ($n > 0$), by prioritizing the expansion of the subpath constrained with an n-number of heuristically evaluated edges.

In [14] it was shown that the number of edge evaluations decreases as the lookahead steps increase. In fact, using an infinite lookahead step (LazyPRM [9], LazySP [13]), i.e., restricting the edge evaluations to the shortest path to the goal (instead of subpaths), is proven to be edge optimal, that is, the number of edge evaluations is minimized. The edge optimality of LazySP comes at the expense of many vertex expansions. This is because the heuristic tree is grown beyond a possibly infeasible edge, and therefore, the subtree must be repaired when the edge is revealed to be infeasible upon evaluation. On the other hand, zero-step lookahead algorithms, such as the A* algorithm, do not grow the subtree beyond any infeasible edge, therefore minimizing the number of vertex expansions. In [14] the relationship between the number of lookahead steps and the total computation time to solve the problem has been studied extensively to highlight the tradeoffs between vertex rewiring and edge evaluation in different problem domains. The Generalized Lazy Search (GLS) encompasses various lookahead strategies with a user-defined algorithmic toggle between vertex rewiring and edge evaluation [15]. With a proper choice of the toggle from the search and the evaluation, GLS hence reduces to LazySP [13], LRA* [14] or LWA* [10].

In this paper, we seek to remedy the excessive edge evaluations of LPA* by borrowing ideas from the lazy search framework of [9]–[11], [13]–[15]. We extend GLS to incorporate lifelong planning behavior, by maintaining a lazy LPA* search tree with non-overestimating heuristic edges. In other words, we restrict the actual edge evaluations of LPA* to only those edges that could possibly be part of the optimal path in the current graph. The proposed algorithm, Lifelong-GLS (L-GLS) is complete and finds the optimal solution in the current graph. Compared to GLS, the proposed algorithm can possibly find the optimal solution faster by reusing previous search results. Compared to LPA*, our algorithm reduces significantly the number of edge evaluations.

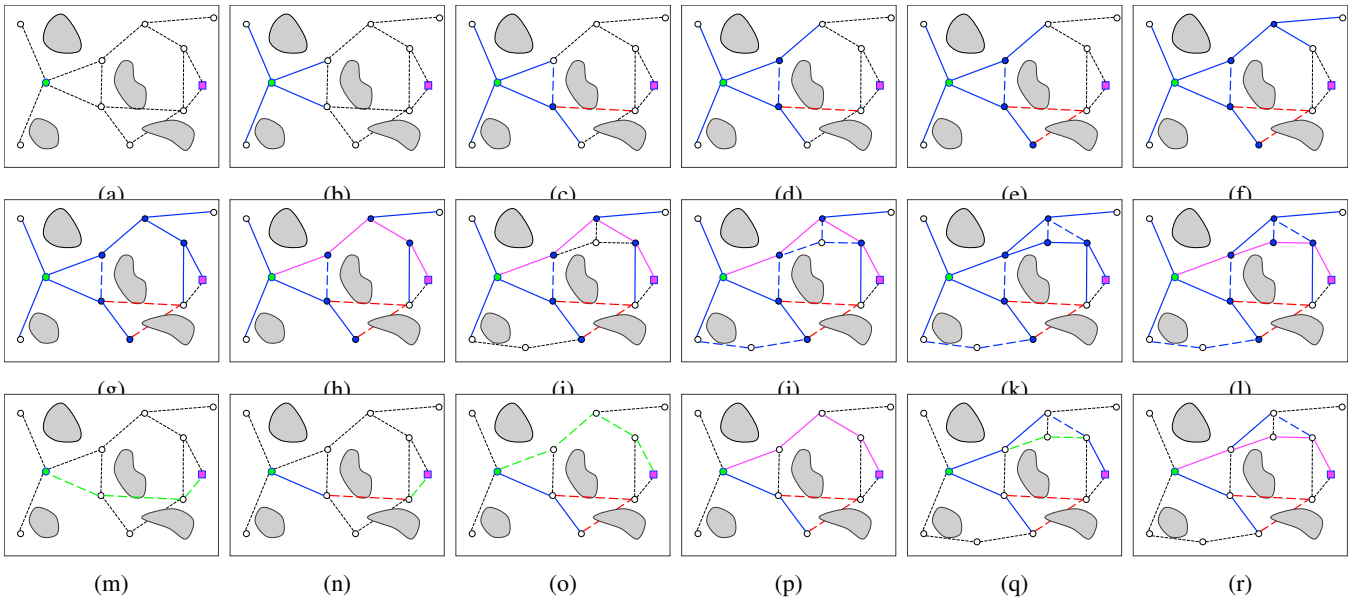


Fig. 1: The propagation of the LPA* search from (a) to (g) while it searches to find the shortest path from the start vertex (●) to the goal vertex (■) given the graph. After finding the optimal solution in (h), new vertices and edges are added in (j). The cost-to-come inconsistencies are propagated in (k) and LPA* finds the new optimal solution in (l). The colored lines are the evaluated edges, where the bold edges are part of the current search tree and the dashed edges are not. Blue and red represents free and obstacle, respectively. The expanded vertices are shown with blue dots. All the incident edges of the expanded vertices are evaluated regardless of their relevancy to the current problem. The L-GLS search is shown from (m) to (r), where edge evaluations are restricted to the heuristically estimated shortest path candidates in dashed green.

II. Problem Formulation

We first introduce the variables and relevant notation that will be used throughout the rest of the paper.

A. Lazy Weight Function

Let $G = (V, E)$ be a graph with vertex set V and edge set E. For a vertex $v \in V$, we denote the predecessor vertices of v with $pred(v)$ and its successor vertices with $succ(v)$. For each edge $e \in E$, a weight function $w : E \to (0, \infty]$ assigns a positive real number, including infinity, to this edge, e.g., the distance to traverse this edge, and infinity if traversing the edge is infeasible. Also, we denote an admissible heuristic weight function with $\widehat{w} : E \to (0, \infty)$, which assigns to an edge a non-overestimating positive real number such that $\widehat{w}(e) \leq w(e)$ for all $e \in E$. We assume that evaluating the true weight w is computationally expensive, but the heuristic edge $\widehat{w}$-value is easy to compute. Let $E_{\text{eval}} \subseteq E$ be the set of all evaluated edges, that is, all edges whose w-values have been computed. Let a lazy weight function $\overline{w} : E \to (0, \infty]$ defined by

$$\overline{w}(e) := \begin{cases} w(e), & \text{if } e \in E_{\text{eval}}, \\ \widehat{w}(e), & \text{otherwise.} \end{cases} \tag{1}$$

B. Optimal Path

Define a path $\pi = (v_1, v_2, \ldots, v_m)$ on the graph $G = (V, E)$ as an ordered set of distinct vertices $v_i \in V$, $i = 1, \ldots, m$ such that, for any two consecutive vertices v_i, v_{i+1}, there exists an edge $e = (v_i, v_{i+1}) \in E$. Throughout this paper, we will interchangeably denote a path as the sequence of such edges. With some abuse of notation, we denote the cost of a path as $w(\pi) := \sum_{e \in \pi} w(e)$. Likewise, we denote $\overline{w}(\pi) := \sum_{e \in \pi} \overline{w}(e)$ for the lazy cost estimate of the path π. Let $v_{\text{s}}, v_{\text{g}} \in V$ be the start and goal vertices, respectively. Let Π be the set of all paths from v_{s} to v_{g} in G. Then, the shortest path planning problem seeks to find $\pi^* := \operatorname{argmin}_{\pi \in \Pi} w(\pi)$.

C. Lazy LPA* Search Tree

We maintain a lazy LPA* search tree to update the inconsistencies that arise from both graph changes and edge value discrepancies between the heuristic weight and the actual weight. The lazy LPA* search tree is identical to the standard LPA* search tree [2], except that lazy LPA* uses the lazy weight function $\overline{w}$ instead of the actual weight function w. For completeness of discussion, next we define the variables of the lazy LPA*.

For each vertex, we store the two cost-to-come values, namely, the g-value and rhs-value to identify the inconsistent vertices, similarly to LPA*. A vertex v whose $g(v) = rhs(v)$ is called consistent, otherwise it is called inconsistent. An inconsistent vertex is locally overconsistent if $g(v) > rhs(v)$ and locally underconsistent if $g(v) < rhs(v)$. The g-value is the accumulated cost-to-come by traversing the previous search tree, whereas the rhs-value is the cost-to-come based on the g-value of the predecessor and the current $\overline{w}$-value of the current edge. Hence, the rhs-value is potentially better informed than the g-value, and it is defined as follows:

$$rhs(v) := \begin{cases} 0, & \text{if } v = v_{\text{s}}, \\ \min_{u \in pred(v)} (g(u) + \overline{w}(u, v)), & \text{otherwise.} \end{cases} \tag{2}$$

Additionally, the rhs-value minimizing the predecessor of v

is stored as a backpointer, denoted with

$$bp(v) := \underset{u \in pred(v)}{\operatorname{argmin}} \left(g(u) + \overline{w}(u, v)\right). \tag{3}$$

Hence, the subpath from v_s to v is retrieved by following the backpointers from v to v_s. The queue Q prioritizes inconsistent vertices using the key $k(v) = [\min(g(v), rhs(v)) + h(v) \;;\; \min(g(v), rhs(v))]$, with lexicographic ordering, where $h(v)$ is a consistent heuristic.

III. Lifelong-GLS Algorithm

The proposed algorithm, Lifelong-GLS (L-GLS), consists of two loops: the inner loop and the outer loop. In the inner loop, the lazy LPA* search tree updates the new shortest path from v_s toward v_g in the current graph G based on the previous search results. The lazily evaluated LPA* search tree uses the lazy estimates of the edge values when it propagates the inconsistencies to find the shortest subpath to the goal in the current graph. The first unevaluated edge on the shortest subpath returned by the lazy LPA* is then evaluated. If the evaluation results in inconsistency, then the lazy LPA* search tree is updated and returns the next best subpath for evaluation. If all the edges on the current shortest path to the goal returned by the lazy LPA* are already evaluated, then L-GLS has found the optimal solution and exits the inner loop. In the outer loop, L-GLS waits for graph changes. When the edges of G change, L-GLS assigns admissible heuristic values to the corresponding edges instead of evaluating them, to make sure that the lazy estimate of the path cost does not overestimate the optimal path cost. Then, the inner loop begins again to search for the new optimal path. Hence, only a subset of the changed edges that could be on the shortest path in the current graph are actually evaluated.

A. Details of the Algorithm and Main Procedures

Next, we describe the step-by-step procedure of L-GLS in greater detail. Before the first search begins, all g-values of the vertices are initialized with ∞ similar to the regular LPA*, and all lazy estimates of edge values are assigned with admissible heuristic values. The first search begins by setting $rhs(v_s) = 0$ and inserting v_s in the priority queue Q. In the main search loop (Line 35-39 of Algorithm 1) the lazy LPA* search tree is grown with ComputeShortestPath(Event) until an Event is triggered by the expansion of a leaf vertex which just became consistent upon this expansion (Line 16 of Algorithm 1). Then, the subpath to this leaf vertex which triggered the Event is returned for evaluation (Line 37 of Algorithm 1). Then, EvaluateEdges evaluates the unevaluated edges along the subpath and updates the lazy estimates with their true weights. If the evaluation of an edge results in a different value than the previous lazy estimate, then EvaluateEdges returns the edge for the lazy LPA* to update this change accordingly by UpdateVertex. The inconsistency is propagated by the lazy LPA* again until the next time the Event is triggered. If the path to the goal is found, and all the edges along this path are evaluated in the current graph, then the path is indeed the optimal path in the current graph. This procedure repeats again when the graph changes.

Algorithm 1 Lifelong-GLS(G, v_s, v_g)

```
1:  procedure CalculateKey(v) return
2:      [min(g(v), rhs(v)) + h(v) ; min(g(v), rhs(v))];
3:  procedure UpdateVertex(v)
4:      if v ≠ v_s then
5:          bp(v) = argmin_{u∈pred(v)} (g(u) + w̄(u, v));
6:          rhs(v) = g(bp(v)) + w̄(bp(v), v);
7:      if v ∈ Q then Q.Remove(v);
8:      if g(v) ≠ rhs(v) then
9:          Q.Insert((v, CalculateKey(v)));
10: procedure ComputeShortestPath(Event)
11:     while Q.TopKey ≺ CalculateKey(v_g) or
12: g(v_g) ≠ rhs(v_g) do
13:         u = Q.Pop();
14:         if g(u) > rhs(u) then
15:             g(u) = rhs(u);
16:             if Event(u) is triggered then
17:                 return path from v_s to u;
18:             for all v ∈ succ(u) do UpdateVertex(v);
19:         else
20:             g(u) = ∞;
21:             for all v ∈ succ(u) ∪ {u} do
22:                 UpdateVertex(v);
23: procedure EvaluateEdges(π̄)
24:     for each e ∈ π̄ do
25:         if e ∉ E_eval then
26:             w̄(e) ← w(e);
27:             E_eval ← E_eval ∪ {e};
28:             if w̄(e) ≠ ŵ(e) then return e;
29: procedure Main()
30:     for all e ∈ E do w̄(e) ← ŵ(e);
31:     E_eval ← ∅;
32:     rhs(v_s) = 0;
33:     UpdateVertex(v_s);
34:     while true do
35:         repeat
36:             π̄ ← ComputeShortestPath(Event);
37:             (u, v) ← EvaluateEdges(π̄);
38:             UpdateVertex(v);
39:         until v_g ∈ π̄ and π̄ ⊆ E_eval
40:         Wait for changes in E;
41:         L ←the set of edges that changed;
42:         for all e = (u, v) ∈ L do
43:             w̄(e) ← ŵ(e);
44:             E_eval ← E_eval\{e};
45:             UpdateVertex(v);
```

Algorithm 2 Candidate Event Definitions [15]

```
1: procedure ShortestPath(v)
2:     if v = v_g then return true;
3: procedure ConstantDepth(v, depth α)
4:     π̄ ← path from v_s to v;
5:     α_v ← number of unevaluated edges in π̄;
6:     if α_v = α or v = v_g then return true;
```

The procedure UPDATEVERTEX is identical to that of the regular LPA*. The only difference is that when UPDATEVERTEX(v) is called, the rhs-value of the vertex v is updated based on the lazy estimate of the incident edge values. This is done to avoid edge evaluations of the irrelevant incident edges of v. When a minimizing predecessor is found lazily, then the vertex assigns its backpointer to this predecessor. Finally, the key of this vertex is updated with CALCULATEKEY to be prioritized in the queue Q.

The choice of an EVENT function determines the balance between the vertex expansion (Line 13 of Algorithm 1) and the edge evaluation (Line 37 of Algorithm 1), as in the GLS framework. For example, if one chooses the SHORTESTPATH as the EVENT, then the algorithm becomes a version of Lifelong-LazySP [13]. That is, the lazy LPA* repairs its inconsistent part of the tree all the way up to the goal, then returns the shortest path to the goal for evaluation. This minimizes the number of edge evaluations of the inner loop. On the other hand, if one chooses the CONSTANTDEPTH of GLS as the EVENT, then the algorithm becomes a version of Lifelong-LRA* [14]. The tree repairing (vertex expansion) of the lazy LPA* is reduced, since the inconsistency propagation is restricted not to exceed a certain depth before evaluating the edges. This comes at the expense of possibly more edge evaluations. Some candidate EVENT definitions of GLS [15] are reproduced in Algorithm 2.

IV. ANALYSIS

We now present some of the properties of L-GLS. We also prove the completeness and correctness of the algorithm, based on the inherited properties from both the LPA* and the GLS algorithms. First, let us state two facts that are invariant during the main search loop.

Invariant 1: The lazy estimate of an edge never overestimates the true edge value, that is, $\overline{w} \leq w$.

Proof: Since $\overline{w}(e) = w(e)$ for all $e \in E_{\text{eval}}$, and $\overline{w}(e) = \widehat{w}(e) \leq w(e)$ for all $e \notin E_{\text{eval}}$, it follows that $\overline{w}(e) \leq w(e)$ for all $e \in E$. ■

Invariant 2: The output subpath $\overline{\pi}$ from v_{s} to v of COMPUTESHORTESTPATH(EVENT) is optimal with respect to $\overline{w}$, that is, $\overline{\pi} = \text{argmin}_{\pi \in \Pi_v} \overline{w}(\pi)$, where Π_v is the set of paths from v_{s} to v.

Proof: COMPUTESHORTESTPATH with an EVENT returns the path $\overline{\pi}$ from v_{s} to v, when the triggering vertex v is expanded. Right before the expansion, v was locally overconsistent. Theorem 6 of LPA* [2] states that whenever COMPUTESHORTESTPATH selects a locally overconsistent vertex for expansion, then the g-value of v is optimal with respect to $\overline{w}$. ■

Now we show the completeness and correctness of the inner loop of L-GLS. The first theorem is due to the completeness of GLS [15], which we restate here.

Theorem 3: Let EVENT be a function that on halting ensures there is at least one unevaluated edge on the current shortest path or that the goal is reached. Then, the inner loop (Line 35-39) of L-GLS implemented with EVENT on a finite graph terminates.

Proof: Suppose the path to the goal has not been evaluated, such that COMPUTESHORTESTPATH(EVENT) returns at least one unevaluated edge to evaluate. Since there is a finite number of edges, the inner loop will eventually terminate. ■

Theorem 4: L-GLS finds the shortest path with respect to the current graph when the inner loop (Line 35-39) terminates.

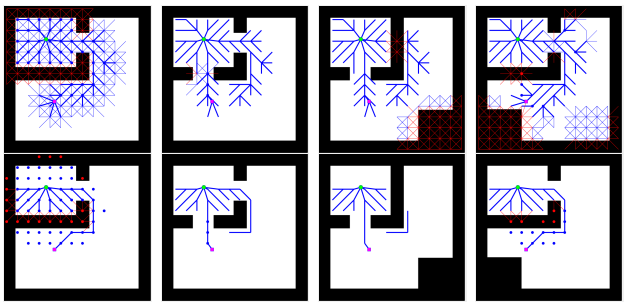
Proof: Let π^* be the optimal path with respect to w in the current graph, that is, $w(\pi^*) = \min_{\pi \in \Pi} w(\pi)$, where Π is the set of all paths from v_{s} to v_{g}. L-GLS terminates its inner-loop when $v_g \in \overline{\pi}$ and $\overline{\pi} \subseteq E_{\text{eval}}$, where $\overline{\pi}$ is the output subpath of COMPUTESHORTESTPATH(EVENT). Then, we have

$$\overline{w}(\overline{\pi}) = \sum_{e \in \overline{\pi}} \overline{w}(e) \leq \sum_{e' \in \pi^*} \overline{w}(e') \leq \sum_{e' \in \pi^*} w(e') = w(\pi^*), \tag{4}$$

where the first inequality holds by Invariant 2, and the second inequality follows by Invariant 1. Hence, $\overline{w}(\overline{\pi}) \leq w(\pi^*)$, and since $\overline{\pi} \subseteq E_{\text{eval}}$, we have $w(\overline{\pi}) = \overline{w}(\overline{\pi}) \leq w(\pi^*)$. But $w(\pi^*) \leq w(\overline{\pi})$, since π^* is the optimal path. Therefore, $\overline{\pi}$ must be the optimal path with respect to w. ■

V. NUMERICAL RESULTS

In this section, we present numerical results comparing L-GLS to LPA* and GLS to demonstrate the efficiency of L-GLS in scenarios where the shortest path planning problem is solved consecutively in a dynamic environment. The search is performed on the same graph with evenly distributed vertices, in which two vertices are adjacent if they are within a predefined radius. The graph topology does not change throughout the experiment, and only the edge values change due to underlying environment changes. We present search results of path planning problems in $\mathbb{R}^2$ for the sake of visualization, and then we present search results of piano movers' problems in $\mathbb{R}^3$ and of manipulation problems in $\mathbb{R}^7$ using PR2, a mobile robot with 7D arms.



(a) (b) (c) (d)

Fig. 2: LPA*(top row) and L-GLS(bottom row) search results to find the shortest path from start vertex(●) to goal vertex(■) per environment change, from left to right: (a) first search, (b) second search, (c) third search, and (d) final search. Lines(／／) are the evaluated edges, and dots(●●) are the expanded vertices during the current search. Bold lines(／) are the edges belonging to the current search tree. Blue and red represents free and obstacle, respectively.

During the 2D experiments, the environment changed three times after the shortest path was found in each of the changed environments (see Figure 2). We recorded the number of vertex expansions and the number of edge evaluations in each search episode for the three algorithms: LPA*, L-GLS, and GLS for each search. We chose SHORTESTPATH for the EVENT function for both L-GLS and GLS.

Piano Movers	LPA*	L-GLS	GLS
First Query in scene 1			
# Edge Evaluation	24445	1867	1867
# Vertex Expansion	124	3612	3612
Total Time (s)	5.08	3.48	3.48
Second Query in scene 2			
# Edge Evaluation	46384	67	1173
# Vertex Expansion	31	277	2266
Total Time (s)	9.46	0.251	2.18
Third Query in scene 3			
# Edge Evaluation	32312	68	872
# Vertex Expansion	23	115	1671
Total Time (s)	6.59	0.112	1.61
PR2			
First Query in scene 1			
# Edge Evaluation	10709	879	879
# Vertex Expansion	205	1555	1555
Total Time (s)	6.25	1.04	1.04
Second Query in scene 2			
# Edge Evaluation	49251	20	81
# Vertex Expansion	9	33	139
Total Time (s)	28.4	0.023	0.094
Third Query in scene 1			
# Edge Evaluation	13024	147	879
# Vertex Expansion	195	363	1555
Total Time (s)	7.58	0.209	1.04

TABLE I: Number of edge evaluations, number of vertex expansions, and approximated planning time for different planners over three consecutive search queries in a dynamic environment.

In the first search, LPA* is equivalent to A*, and L-GLS is equivalent to GLS (See Figure 2.a). LPA* evaluated 390 edges and expanded 45 vertices, whereas L-GLS and GLS both evaluated 61 edges and expanded 314 vertices.

After the first search, only a small part of the environment changed (see Figure 2.b), opening a shorter passage to the goal. LPA* evaluated 18 edges corresponding to the change, then expanded 4 inconsistent vertices to find the shortest path in the current graph. L-GLS evaluated 4 edges that belong to the new shortest path to the goal, and expanded 4 inconsistent vertices. The GLS evaluated 7 edges and expanded 6 inconsistent vertices.

When the environment changed in the irrelevant region (see Figure 2.c), LPA* evaluated 153 edges corresponding to the environment change, but did not expand any vertices, as they were irrelevant to the current search. L-GLS did not do any additional operations to find the shortest path, since the path was already optimal. GLS was identical to the previous search with 7 edge evaluations and 6 vertex expansions.

Finally, the environment changed back to the first search episode with the addition to a new obstacle in the irrelevant region. The GLS search was identical to the first search episode with 61 edge evaluations and 314 vertex expansions. On the other hand, L-GLS evaluated only 11 edges and expanded 83 vertices. This is because the majority of the relevant edges were already evaluated during the previous searches, and the majority of the relevant vertices were already consistent. Similarly, LPA* expanded a fewer number of vertices and evaluated a fewer number of edges compared to the first search episode with 273 edge evaluations and 9 vertex expansions, since it utilized the previous search results.

(a) Scene 1 (b) Scene 2 (c) Scene 3

Fig. 3: The shortest paths of the Piano Movers' problems in dynamic environment.

We also implemented LPA* and L-GLS as an OMPL Planner [16] with the MoveIt! interface [17] for the 3D piano movers' problem and for the 7D manipulator experiment. All the algorithm implementations were in C++, and the experiments were run on an 2.20 GHz Intel(R) Core(TM) i7-8750H CPU Ubuntu 16.04 LTS machine with 15.5GB of RAM.

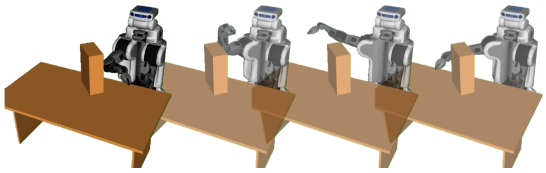
We find the shortest paths for the piano from the Apartment scenario in OMPL [16] from a start configuration to a goal configuration without colliding with the moving obstacles (see Figure 3). There were three consecutive searches in the environment, where the first search was on scene 1 (Figure 3 (a)), the second search was on scene 2 (Figure 3 (b)), and the third search was on scene 3 (Figure 3 (c)). The search was performed on a prebuilt graph with 8,000 vertices and 34,327 edges. The vertices were sampled using a Halton sequence in $\mathbb{R}^3$.

Similarly, we find the shortest paths for the right arm of PR2 robot from a start configuration to a goal configuration without collision in a dynamic environment where the obstacle moves (see Figure 4). There were three consecutive searches in the environment, where the first search was on scene 1 (Figure 4 (a)), the second search was on scene 2 (Figure 4 (b)), and the third search was on scene 1 again. The search was performed on a prebuilt graph with 30,000 vertices and 168,795 edges. The vertices were sampled using a Halton sequence in $\mathbb{R}^7$, bounded by the PR2 arm's joint-angle bounds. Two vertices are adjacent in this graph if the Euclidean distance between them is less than 0.9 rad.

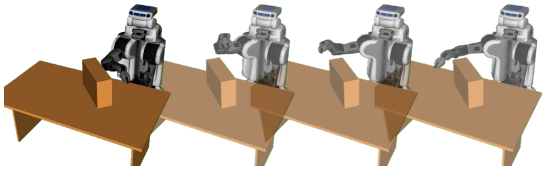
We compared three algorithms: LPA*, L-GLS, and GLS, in which the number of edge evaluations and the number of vertex expansions along with the approximated planning time are recorded for each search episode and tabulated in Table I. The approximate planning time was computed as the weighted sum of the number of edge evaluations and the number of vertex expansions. In addition, the similar search results are collected by varying lookahead values to illustrate their effects and they are plotted in Figure 5. Our proposed method outperforms the other two algorithms in terms of planning time.

VI. Conclusion

We have presented a new replanning algorithm to find the shortest path in a given graph efficiently using previous search results. The proposed algorithm maintains a lazy

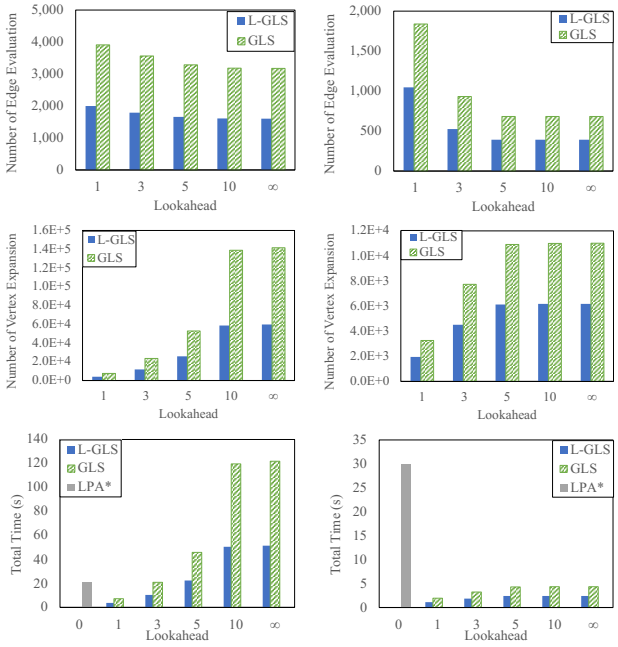


(a) Scene 1



(b) Scene 2

Fig. 4: The shortest paths of the right arm of PR2 robot for the same query in dynamic environment.



(a) Piano (b) PR2

Fig. 5: From top to bottom: the total number of edge evaluations, the total number of vertex expansions, and the total planning time over three consecutive search of L-GLS and GLS with for different lookahead values: (a) for the piano movers' problem and (b) for the mainpulator problem. LPA* evaluated 103,141 edges and expanded 178 vertices for the piano movers' problem, and evaluated 72,984 edges and expanded 409 vertices for the manipulator problem.

LPA* tree to efficiently repair the inconsistency of the existing search that arises either from external environment changes or internal discrepancies between the lazy estimate and the real weight of an edge cost. Based on the efficiency of LPA*, the propagation of vertex rewiring to repair any vertex inconsistencies is restricted only to the shortest path candidate. Similar to the GLS framework, only the edges in the current shortest path candidate are evaluated. The proposed algorithm reduces by a substantial amount the edge evaluations per search compared to LPA*, and it can find a new shortest path significantly faster than GLS, given a change in the graph.

Acknowledgement: We would like to thank Aditya Mandalika for setting up the benchmark testing for the Piano Movers' problem. This work has been supported by ARL under DCIST CRA W911NF-17-2-0181 and SARA CRA W911NF-20-2-0095 and NSF under award IIS-2008686.

References

[1] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *Journal of Algorithms*, vol. 21, pp. 267–305, 1996.

[2] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1, pp. 93 – 146, 2004.

[3] S. Aine and M. Likhachev, "Truncated incremental search," *Artificial Intelligence*, vol. 234, pp. 49 – 77, 2016.

[4] 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, July 1968.

[5] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Department, Iowa State University, Tech. Rep., 1998.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[8] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *IEEE International Conference on Robotics and Automation*, Karlsrühe, Germany, May 6–10 2013, pp. 5054–5061.

[9] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *IEEE International Conference on Robotics and Automation*, vol. 1, San Francisco, CA, April 24–28 2000, pp. 521–528.

[10] B. Cohen, M. Phillips, and M. Likhachev, "Planning single-arm manipulations with n-arm robots," in *Proceedings of Robotics: Science and Systems*, Berkeley, CA, July 12–16 2014.

[11] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *IEEE International Conference on Robotics and Automation*, Seattle, WA, May 26–30 2015, pp. 2951–2957.

[12] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE International Conference on Robotics and Automation*, Seattle, WA, May 26–30 2015, pp. 3067–3074.

[13] C. M. Dellin and S. S. Srinivasa, "A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors," in *Proceedings of the International Conference on Automated Planning and Scheduling*, no. 9, London, UK, 2016, pp. 459–467.

[14] A. Mandalika, O. Salzman, and S. S. Srinivasa, "Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges," in *Proceedings of the International Conference on Automated Planning and Scheduling*, Delft, Netherlands, 2018, pp. 476–484.

[15] A. Mandalika, S. Choudhury, O. Salzman, and S. S. Srinivasa, "Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, Berkeley, CA, 2019, pp. 745–753.

[16] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[17] D. Coleman, I. A. Şucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a MoveIt! case study," *Journal of Software Engineering for Robotics*, vol. 5, no. 1, pp. 3–16, May 2014.