Adaptive Lazy Collision Checking for Optimal Sampling-based Motion Planning

Donghyuk Kim[1] and Youngsun Kwon[1] and Sung-eui Yoon[2]

***Abstract*— Lazy collision checking has been proposed to reduce the computational burden of collision checking which is considered as a major computational bottleneck in sampling-based motion planning. Unfortunately, in complex environments with many obstacles, lazy collision checking can cause an excessive amount of optimistic thrashing problems and significantly degrade the performance of the planner. In this paper, we present an adaptive lazy collision checking method to alleviate the optimistic thrashing, thus broaden the applicability.**

Our method delays collision checking on the regions predicted to be in the configuration free space, while checking early on the other regions to reduce the optimistic thrashing. To identify such regions, we adopt a configuration free space approximation represented by a set of hyperspheres which can be constructed without significant proximity computation.

To demonstrate benefits of our approach we have compared against prior methods including RRT*, PRM* and lazy PRM* across benchmarks with varying dimensions. Overall, our method shows meaningful performance improvement up to three times higher in terms of convergence speed over the tested methods. We also discuss properties and theoretical analysis of the proposed algorithm.

I. INTRODUCTION

Sampling-based motion planning has been well studied for past several decades thanks to its benefits of scalability to the high dimensional problem and probabilistic completeness. RRT [1] and PRM [2] are examples of most dominant algorithms in current research trends and a large number of follow-up works have been proposed upon them for various improvements such as higher performance for finding an initial solution with biased sampling [3], [4] and for the narrow passage problem [5], [6].

When it comes to the optimal motion planning, Karaman et al. presented RRT*, PRM*, and RRG [7], which guarantee the asymptotic optimality. As a follow-works, biased sampling techniques [8], [9], [10] have particularly drawn much attention because of its noticeable performance improvement and simplicity in terms of integration with the aforementioned works. Nonetheless, additional computational overheads of adopting optimal planners are not negligible in practice [11]. Collision checking, in particular, has been identified as the most time-consuming part in sampling based motion planning among other components [12], [13]. To overcome the computational bottleneck, many prior techniques have been proposed. Some of them introduced efficient collision detection methods with information of local collision free space [14], [15], GPU acceleration [16], and lazy evaluation [12].

Recently, lazy collision checking was introduced to improve the optimal motion planning method [13] by delaying the expensive collision checks. While this technique, named lazy RRG/PRM* improves the performance, it has been also identified to give rise to excessive false negative errors of collision checking, so-called optimistic thrashing [13]. Depending on the shape of configuration free space, it could significantly degrade the performance of optimal motion planning.

Main contributions. To alleviate the optimistic thrashing of the prior lazy optimal planner, we propose an adaptive lazy collision checking named adaptive lazy PRM* in this work. Adaptive lazy PRM* is built upon lazy PRM* [13], while adaptively delaying collision checks in consideration of the approximate configuration free space (Sec. IV). Specifically, our method delays collision checks on the regions predicted as collision-free, while explicitly checking on other regions to maintain the efficiency of lazy collision checking. To predict such regions, we propose to use configuration free space approximate utilizing empirical collisions found in the nature of sampling-based motion planning [17].

To demonstrate benefits of the proposed algorithm, we have compared it against RRT*, PRM* and lazy PRM* across three different benchmarks. We have found that our adaptive lazy PRM* shows meaningful performance improvement, 20% to 250% faster convergence rate over the prior methods (Sec. VI). We also discuss the optimality and properties of our method in Sec. V.

II. RELATED WORK

In this section, we review prior techniques that are directly related to our problem.

A. *Collision Checking*

In sampling-based motion planning algorithms, collision checking has been considered as one of the main bottlenecks in terms of the computational cost [18]. Its cost depends on the dimensionality of a given problem and varies over the complexity of obstacles, while other components such as sampling and nearest neighbor search are much less affected by this kind of environmental factors.

In order to reduce the computational burden of collision checking, prior methods adopt additional geometry operation [14] or implicitly exploit geometric information

[1]Donghyuk Kim (`donghyuk.kim@kaist.ac.kr`), Youngsun Kwon (`youngsun.kwon@kaist.ac.kr`) are with the School of Computing and [2]Sung-eui Yoon (Corresponding author, `sungeui@gmail.com`) is with the Faculty of School of Computing, Korea Advanced Institute of Science and Technology(KAIST) at Daejeon, Korea 34141

by analyzing the distribution of samples or sampling history [17], [19]. Rather than directly reducing the overhead of collision checking itself, Bialkowski et al. [20] proposed an algorithm that asymptotically converges the complexity of collision checking to a constant time amortized as the number of iteration increases. Free space approximation approaches to reduce the number of samples in configuration obstacle space [17] also have been studied and given helpful insights on additional performance improvement of collision detection.

B. Lazy Evaluation of Collision Checking

Lazy collision checking in sampling-based motion planning [12], [13] is a technique to delay collision checking at the expansion step, and then performs on demand only for edges along promising solution paths. The approach provides better performance, when the majority of given sampling space is not necessary to explore.

Its potential drawback is known as "optimistic thrashing" [13], which occurs when the planner optimistically treats an infeasible solution path as a feasible one. It can then give rise to frequent roll-backs of graph updates, resulting in degrading the overall performance of the planner. Lazy collision checking, thus, has been considered to have clear strength and weakness depending on given problems.

Built on top of those previous works, we focus on how optimistic thrashing affects the overall performance of optimal motion planner with lazy collision checking, and suggest a simple, yet efficient technique to improve the overall performance by reducing optimistic thrashing dramatically without using expensive additional proximity queries.

III. Background

In this section, we explain basic concepts that our method is built upon. We begin by defining motion planning problems related to ours.

Motion planning problem. Given a robot, let $\mathbf{X} \subseteq \mathbb{R}^{\mathbf{d}}$ be the configuration space, where $x \in \mathbf{X}$ is a configuration of the robot. Let $\mathbf{X}_{obs} \subset \mathbf{X}$ to denote the *obstacle region*, and the *free* region is then defined by $\mathbf{X}_{free} = \mathbf{X} \backslash \mathbf{X}_{obs}$. A collision-free path that connects x_{init} to x_{goal} consists of a sequence of collision-free configurations and can be expressed as a continuous function, $f : [0 : 1] \rightarrow \mathbf{X}_{free}$, where $f(0) = x_{init}$ and $f(1) = x_{goal}$. The motion planning problem is then to compute such a feasible collision-free path given initial and goal configurations.

Optimal motion planning problem. Let $g : f \rightarrow \mathbb{R}$ be a cost function that associates a non-zero cost, which is the integral of the cost function over the given path of f. To be specific, $g(f)$ can be denoted by $\int_0^1 g(\Delta f(t))dt$, where $t \in [0 : 1]$ is a time parameter and $\Delta f(t)$ is a measurable trajectory passing $f(t)$. The optimal motion planning problem is then to compute a collision-free path, f^*, in a similar way to that of the motion planning problem, but its resulting path f^* is computed to have the minimum cost among all the possible paths in the given configuration space $\mathbf{X}$.

Algorithm 1: PRM* (Naïve and Lazy)

```
1  $V \leftarrow x_{init}$
2  $E \leftarrow \emptyset$
3  while $Termination\ condition\ is\ not\ satisfied$ do
4      $x_{rand} \leftarrow Sample()$
5      if $IsCollisionFree(x_{rand})$ then
6          Insert $x_{rand}$ to $V$
7          $X_{near} \leftarrow Near(x_{rand})$
8          foreach $x_{near} \in X_{near}$ do
9              if $IsCollisionFree(x_{rand}, x_{near})$ then
10                 Insert $(x_{rand}, x_{near})$ to $E$
11         $UpdateShortestPathTree(G)$
12 return $ShortestPath(G)$
```

A. Preliminaries

We build our method upon PRM*, which is one of optimal sampling-based motion planning approaches proposed by Karaman et al. [7]. Graph-based algorithms such as PRMs and RRGs have tolerance for disconnection, thus can expect computational benefits even in situations where structural changes can frequently occur in **G** since those maintain all of the connections to its near neighbors.

Proximity queries and notations used throughout the paper are explained in below. For more details and generalizations, refer to related papers [1], [7].

G = (V, E) : A graph **G** consists of a vertex set **V** and an edge set **E**, which is a subset of $\mathbf{V} \times \mathbf{V}$. A vertex $x \in \mathbf{V}$ corresponds to a configuration $x \in \mathbf{X}_{free}$, and $e \in E$ represents a motion that connects x_i and x_j, denoted by (x_i, x_j). A definition of motion follows motion capabilities and constraints of the given problem.

Cost(x) : Given a configuration x, returns a cost from the initial configuration x_{init} to x along the shortest path on **G**.

Nearest(x) : returns a node $x' \in \mathbf{G}$ that is nearest to the given configuration x, i.e., $\underset{x' \in V}{\text{argmin}}(\|x' - x\|)$.

Near(x) : returns a set of nodes that lie within a ball of radius r_{near} centered at the node x, i.e., $\{x' \in V | \ \|x' - x\| \leq r_{near}\}$. For the optimality, threshold r_{near} is set to $\gamma(\frac{\log|V|}{|V|})^{1/d}$, where d is a dimension of the problem, and γ is a user defined constant [7]. K-nearest neighbor search also can be an alternative for **Near**$(\cdot)$, in that case k is defined as $\lambda(\mathrm{e} + \frac{\mathrm{e}}{d}) \cdot log(|V|)$. where λ is a constant greater than 1 to guarantee almost-sure asymptotic optimality.

IsCollisionFree(x) : returns a boolean value *true* if a given configuration $x \in X_{free}$, *false* otherwise.

IsCollisionFree(x_{from}, x_{to}) : Likewise, returns *true* if there is no collision on the given edge (x_{from}, x_{to}), *false* otherwise. With discrete collision detection which most of motion planners use, this function can be divided into a sequence of point collision checking, i.e., $\bigwedge_{i \in [0,1]} IsCollisionFree(f(i))$, where $f(0) = x_{from}$ and $f(1) = x_{to}$ as shown in Alg. 2.

A pseudocode for naïve and lazy PRM* are shown in Alg. 1. The difference between both algorithms can be found

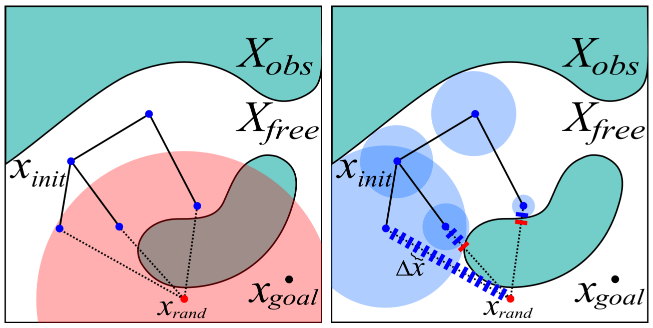

Fig. 1. The left image shows an example of connecting x_{rand} to its near neighbor set, X_{near}, which correspond to blue points within the red circle. The dotted lines indicate edges to be checked for collision. The right image shows a subset of X_{free} denoted as a set of light blue circles, each of which has a center at $x \in V$ and radius corresponding to the distance to the closest X_{obs}. A sequence of bars along the edge indicates points to be checked for collision, where Δx is the resolution of a discrete collision checker.

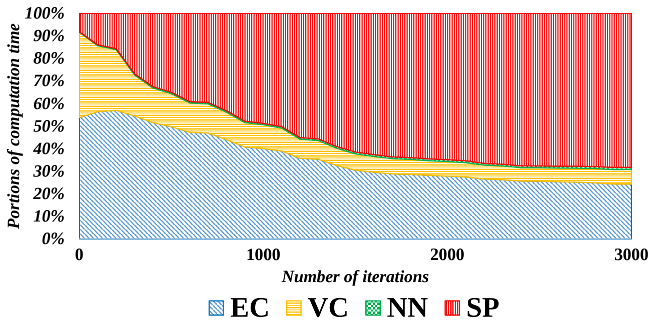

Fig. 2. Breakdown of computation time taken for each component in lazy PRM* tested on benchmark of Fig. 4. Each shortened key in legend stands for computation time of edge collision checking (EC), vertex collision checking (VC), nearest neighbor search (NN) and update of DSPT (SP). Results are averaged over 10 times of trials and normalized for easy comparison.

in highlighted lines (9, 11). In lazy PRM*, $IsCollisionFree(\cdot)$ (:9) always returns $true$ to delay collision checking during the expansion, and actual checking is done along the solution path in $UpdateShortestPathTree(\cdot)$ (:11). This function also updates a DSPT(Dynamic Shortest Path Tree) [13], [21] to maintain shortest paths from x_{init} to all $x \in G$. DSPT dynamically updates a shortest path tree with better amortized time complexity than existing methods such as Dijkstra's algorithm and A*, when edges are dynamically inserted and removed.

B. Motivations

Lazy collision checking in conjunction with optimal sampling-based motion planner can successfully improve the overall performance over regular optimal planners in general cases [13]. Nonetheless, lazy evaluation of collision checking adopted in such lazy planners has been known to have the problem of "optimistic thrashing" [13]. In this case, the lazy planner optimistically treats infeasible paths without collision checks as feasible ones which causes a rollback of graph updates. As a result, the performance of planner can be lowered down by the false negative error under the optimistic assumption of lazy collision checking.

For the details, DSPT, the graph update method used in lazy PRM*, can be divided into two sub-routines, $Decrease(\cdot)$ and $Increase(\cdot)$. For each configuration $x \in V$, $\mathbf{Cost}(x)$ can only decrease or remain unchanged if a vertex or an edge is newly inserted into G. Such a procedure is handled in $Decrease(\cdot)$. On the other hand, $\mathbf{Cost}(x)$ can only increase if an edge $e \in E$ turns out to be infeasible and then removed from G, i.e., false negative error on the delayed collision checking. This kind of case is covered by $Increase(\cdot)$.

The computation overhead of $Increase(\cdot)$ is more expensive than that of $Decrease(\cdot)$. Specifically, $Increase(\cdot)$ has the complexity of $O(klog(|V|))$, while $Decrease(\cdot)$ has $O(log(|V|))$, where k is defined by structural properties of the graph G [21]. As a result, $Increase(\cdot)$ becomes the dominant factor between them, by causing optimistic thrashing.

Fig. 2 shows relative computational overheads of different components in lazy PRM* where SP stands for both $Increase(\cdot)$ and $decrease(\cdot)$. As shown in the graph, the cost of graph update gradually increases as the number of iteration goes higher, and it rather seems to be comparable against the collision detection. Accordingly, we can observe that the major bottleneck is shifted to the graph update(SP) from collision checking(CD) in lazy PRM*.

The left image in Fig. 1 shows our motivation where collision checks between x_{rand} and its near neighbors X_{near} are illustrated. With lazy collision checking, a planner optimistically accepts all of three edges depicted by dotted lines to aggressively expand the search graph G. It is, however, obvious that some of them will turn out to be infeasible, thus likely to cause additional overheads, i.e., the invocation of $Increase(\cdot)$,

In the right image in Fig. 1, each configuration $x \in \mathbf{V}$ is associated with a light blue circle. The circle represents a collision-free hypersphere with a radius of distance to the closest X_{obs}. Also, a set of configurations to be checked for $IsCollisionFree(\cdot)$ is denoted by blue and red rectangles, where blue bars are collision-free, reds otherwise. We can observe that although we know only a subset of X_{free}, it is possible to predict which configurations are likely to be collision-free without performing collision detection since a configuration x placed within at least one of free circles cannot be in X_{obs}.

In this context, we propose an adaptive lazy collision checking algorithm for optimal motion planning to adaptively delay collisions only in regions that are predicted to be free. To obtain a configuration free space information as shown in Fig. 1, we use a configuration free space approximation method inspired by inexact ball tree [17], as a guidance for the proposed algorithm to efficiently avoid optimistic, but problematic expansion of graph $\mathbf{G}$ in advance. In what follows, we explain details of our algorithm and discuss its properties.

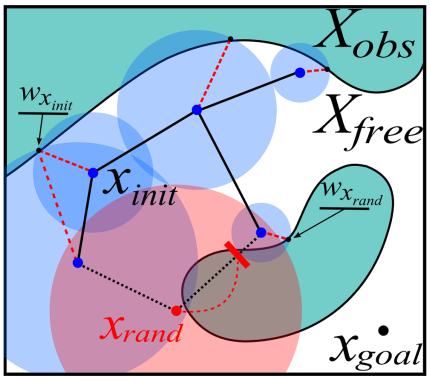

Fig. 3. An example of adaptive collision checking. For a sample configuration x_{rand}, the expansion to near neighbors requires only a single point collision checking at the red bar since other area are predicted as a collision-free space by our free space approximation, $\widetilde{F}_{free}$. After the expansion, the witness of x_{rand} is inherited from the near neighbors.

IV. Algorithms

In this section, we elaborate each component defined in the proposed algorithm and underlying practical meaning.

A. Configuration Free Space Approximation

We first define our approximate configuration free space, which is inspired by inexact ball tree [17]. The inexact ball tree algorithm was originally proposed to approximate configuration free space to reject samples in a wide open area for rapid exploration of RRT. On the other hand, we use it for our adaptive lazy evaluation, while utilizing its scalability, efficiency and non-use of proximity computation, e.g., computing a distance to the closest $\mathbf{X}_{obs}$.

The approximate, free configuration space or simply approximate free space $\widetilde{F}$, is defined by a set of hyperspheres in the given configuration space $\mathbf{X}$. Each hypersphere, $\widetilde{F}_{x_i}$, represents a local approximate free space, as shown by light blue circles in Fig. 3. Each hypersphere is centered at a configuration $x_i \in V$ with a radius r_{x_i}, and defined by the following equations:

$$\begin{aligned} \widetilde{F} &= \bigcup \widetilde{F}_{x_i},\ \forall x_i \in V, \\ \widetilde{F}_{x_i} &= \{x \mid r_{x_i} > \|x - x_i\|\},\ x_i \in V. \end{aligned} \tag{1}$$

To compute the radius of a hypersphere centered at configuration x_i, we utilize a set of local empirical collisions. The set includes configurations $x \in \mathbf{X}_{obs}$ checked for an edge collision checking adjacent to x_i, where such information is available in the nature of discrete collision checkers during the execution.

Alg. 2 shows an example of edge collision checker, *IsCollisionFree*$(\cdot)$. In this function *Discretize*$(\cdot)$ splits the given edge (x_{from}, x_{to}) into a sequence of point collision checking of k different configurations in a discrete manner, according to the collision checking resolution. Therefore, only a single x_{inter} in $\mathbf{X}_{obs}$ will be captured for our approximation if such a configuration exists. Note that the implementation detail can vary according to the collision checking pattern, e.g., bisection method [12].

Algorithm 2: *IsCollisionFree*

```
  Input: $x_{from}$, $x_{to}$, an edge $(x_{from}, x_{to})$
1 $k, \Delta x \leftarrow Discretize(x_{from}, x_{to})$
2 for $i = 1$ to $k$ do
3     $x_{inter} \leftarrow x_{from} + i \cdot \Delta x$
4     if $\neg IsCollisionFree(x_{inter})$ then
5         return $false$
6 return $true$
```

Lastly, we define a witness, $w_x \in \mathbf{X}_{obs}$ associated with a configuration x to be a configuration which is found closest to x during the execution. The hypersphere associated with a configuration x, $\widetilde{F}_x$ therefore, has a radius of the distance to w_x, i.e., $r_x = \|w_x - x\|$.

For both input configurations x_{init} and x_{goal}, their witness and corresponding radius r are initialized with *Null*. For a new sample configuration x_{rand}, the initial values are inherited from its near neighbors at the end of iteration, i.e., $w_{x_{rand}} = argmin_{x_{near} \in X_{near}}(\|w_{x_{near}} - x_{rand}\|)$, where X_{near} is a set of near neighbor of x_{rand}.

In this approach, we do not perform any collision detection dedicated to computing witnesses, while utilizing collision information which is naturally available during the planning process.

B. Adaptive Lazy Collision Checking

In this section, we look into how our approximate free space $\widetilde{F}$ can be used for improving the performance of motion planning in conjunction with the proposed adaptive lazy collision checking. The pseudo-code of the modified collision checker is shown in Alg. 3. In a nutshell, the adaptive collision checker performs collision checking only for configurations outside of $\widetilde{F}$ under the optimistic assumption that configurations inside $\widetilde{F}$ is likely to be in $\mathbf{X}_{free}$. The assumption is highly likely to be valid as the cardinality of search graph, $|V|$ increases.

For each configuration x_{inter} to be checked for collision, we test whether the configuration is in either one of hyperspheres associated with two endpoints x_{from} and x_{to}, i.e., $x_{inter} \in \widetilde{F}_{x_{from}} \bigcup \widetilde{F}_{x_{to}}$. If so, we assume that the configuration is in the configuration free space, and delay its collision check. Otherwise, we check collision explicitly to prevent optimistic thrashing. Once we have empirical collision information, we also update witness of x_{from} and x_{to} if possible.

In Fig. 3, we can observe how the edge collision checking for two edges, denoted by dotted lines connected from x_{rand}, is processed by our adaptive lazy collision checking. *IsCollisionFree*$(\cdot)$ denoted by the red bar is the only explicit point collision checking in this example since the other regions are covered by our approximate free space (light blue circle). As a result, we can identify infeasible edges earlier then reject to prevent optimistic thrashing.

While our method can reduce the optimistic thrashing, it is clear that the proposed algorithm requires additional collision checking compared to lazy PRM* since we consider more collision checking in expansion step.

```
Algorithm 3: IsAdaptiveCollisionFree
  Input: $(x_{from}, x_{to})$
1 $k, \Delta x \leftarrow Discretize(x_{from}, x_{to})$
2 for $i = 1$ to $k$ do
3     $x_{inter} \leftarrow x_{from} + i \cdot \Delta x$
4     $d_{from} \leftarrow \|x_{inter} - x_{from}\|$
5     $d_{to} \leftarrow \|x_{inter} - x_{to}\|$
6     if $(d_{from} < R_{x_{from}}) \vee (d_{to} < R_{x_{to}})$ then
7         continue
8     else if $\neg IsCollisionFree(x_{inter})$ then
9         return $false$
10 return $true$
```

```
Algorithm 4: THE PROPOSED ALGORITHM
1 $V \leftarrow x_{init}$
2 $E \leftarrow \emptyset$
3 while Termination condition is not satisfied do
4     $x_{rand} \leftarrow Sample()$
5     if $IsCollisionFree(x_{rand})$ then
6         Insert $x_{rand}$ to $V$
7         $X_{near} \leftarrow Near(x_{rand})$
8         foreach $x_{near} \in X_{near}$ do
9             if $IsAdaptiveCollisionFree(x_{near}, x_{rand})$
               then
10                Insert $(x_{near}, x_{rand})$ to $E$
11        $UpdateDSPT(G)$
12        while $\neg ValidateSolution(G)$ do
13            $UpdateDSPT(G)$
14 return $ShortestPath(G)$
```

Nonetheless, we show the overall performance improvement of our adaptive lazy collision checking with analysis and experimental results in Sec. V and Sec. VI, respectively.

Note that when $r_x = \infty, \forall x \in V$, our method works exactly like lazy PRM*, and reduces to naïve PRM* with $r_x = 0$.

C. Resulting Algorithm

The overall pseudo-code of the proposed algorithm is shown in Alg. 4, where modified lines from lazy PRM* [13] are highlighted (:9, 12) for easy comparison. *IsAdaptiveCollisionFree*$(\cdot)$ replaces the prior edge collision checking between x_{rand} and X_{near}(:9).

ValidateSolution$(\cdot)$(:12) lazily evaluates feasibility of a solution path until we found a valid solution like lazy PRM*, while updating our approximation $\widetilde{F}_{free}$ whenever we found empirical collision information.

UpdateDSPT$(\cdot)$ updates DSPT to reflect the structural change of G such as vertex and edge insertion(:11) or edge deletion(:13). Thus, the false negative error of lazy collision checking causes much overhead on the validation loop(:12-13). For details about lazy collision checking with DSPT, refer to the lazy RRG/PRM* [13] and the original work on DSPT [21].

V. Experiments

A. Experiment setup

We experimentally show the performance of the proposed algorithm in comparison with RRT*, PRM* and lazy PRM* [13]. For fair comparison, all of four different algorithms are implemented with DSPT(Dynamic Shortest Path Tree) for the shortest path computation by following the existing testing protocol [13]. Also, all the tested methods are built upon the same proximity subroutines such as collision detector, nearest neighbor search (G-NAT in OMPL [22]) and sampling rejection technique [8] on V-REP simulator [23]. We use r-NN for our $Near(\cdot)$ with a user-defined parameter $\gamma = 1.1$.

We test three different benchmarks on the same machine that has 3.34GHz i7 processor with 16GB of main memory. The overall results are given in Tab. I. Tested methods produce solution paths that convergence to the optimal one as a function of computation time, but with different convergence rates. These benchmarks and the result tested on them are shown in Figs. 4, 5, and 6. Note that the PRM* generates a solution path with a set of samples which is generated by a single batch at a sampling phase [24]. We, therefore, only report the cost of final solution for PRM* in our experiment.

B. Experiment results

The first scene (Fig. 4) is intended to show a basic motion planning problem where a solution cannot be directly found across the walls. The other benchmarks (Figs. 5, 6 consist of both wide-open area and narrow passages, where optimistic thrashing can frequently occur due to the complex shape of the configuration free space. It also requires heavier computation on collision checking due to the complexity of workspace environment.

In Fig. 4 and 5, we can observe that RRT* outperforms other algorithms in terms of the cost of solution path at the early phase. This kind of result can be achieved by relatively faster growth pattern of a tree-based algorithm and light-weight construction process. Across all the tested benchmarks, however, the proposed method outperforms the other tested algorithms regrading the convergence speed toward the optimal solution.

As shown in the Tab. I, computation time ratio for collision checking and graph update (CD and SP, respectively) shows how the main computational bottleneck of each algorithm shift in different benchmarks. Even though the lazy PRM* efficiently tackles down the major computational bottleneck, i.e., collision checking(CD) and even outperforms the single-query planner RRT*, the cost of graph update(SP) becomes comparable to the collision checking. This kind of phenomenon can be observed, particularly in 3D L-maze scene where the shape of solution path has a zig-zag pattern. It is due to the false negative error of edge expansion under the optimistic assumption which causes the more loop on the validation phase(Line number :12-13 in Alg. 4). Moreover, unlike the computational cost of the collision checking which tends to be static through the execution, that of graph update

TABLE I

Overall statistics measured in the given time budget for each benchmark. We report the number of vertices ($|V|$) in G, final solution cost (**Cost**), computation time ratio for collision detection (**CD**), graph update in $UpdateDSPT(\cdot)$ (**SP**) and success rate (Success). RRT* and PRM* has zero for **SP** since they run in a non-lazy manner. Results are averaged over 10 trials.

Benchmark	3D L-maze					2D-Conference					6D-Sponza				
Evaluation	$\|V\|$	Cost	CD	SP	Success	$\|V\|$	Cost	CD	SP	Success	$\|V\|$	Cost	CD	SP	Success
RRT*	10419	4.070	99%	0%	90%	12802	2.288	99%	0%	100%	4573	7.943	99%	0%	20%
PRM*	3064	4.218	99%	0%	100%	1759	2.322	99%	0%	100%	2530	10.300	99%	0%	80%
Lazy PRM*	4507	4.012	27%	61%	100%	11881	2.283	25%	52%	100%	6533	7.838	56%	22%	100%
Proposed	16778	3.887	58%	20%	100%	16832	2.273	54%	19%	100%	12813	6.323	74%	7%	100%

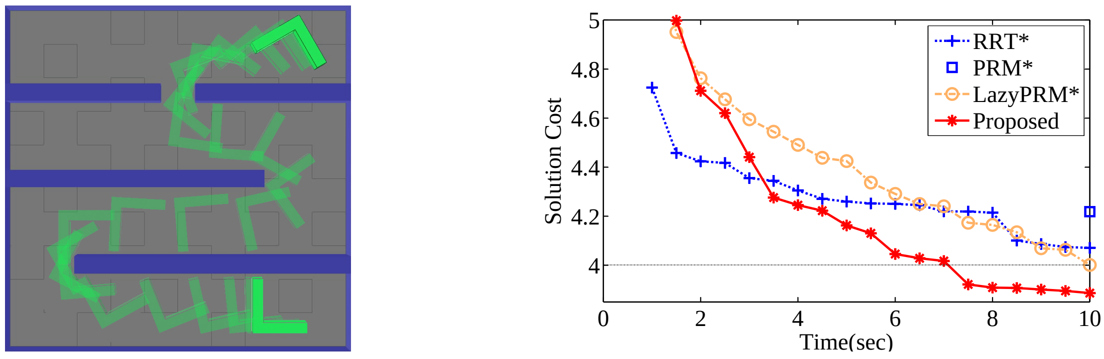

Fig. 4. A 3-DOF L-shaped rigid body planning in a maze-like environment. It shows a drastic decrease of **SP**, and the performance improvement.

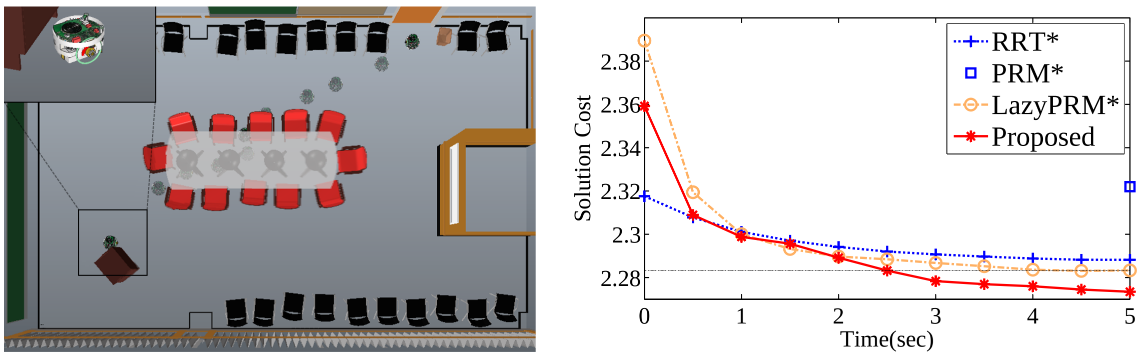

Fig. 5. A 2-DOF robot planning problem in a conference room which has both wide-open area and narrows passages. To find an optimal homotopy, the robot should pass under the table where chair/table legs compose narrow passages, resulting in a performance degradation by false negative errors of lazy collision checking.

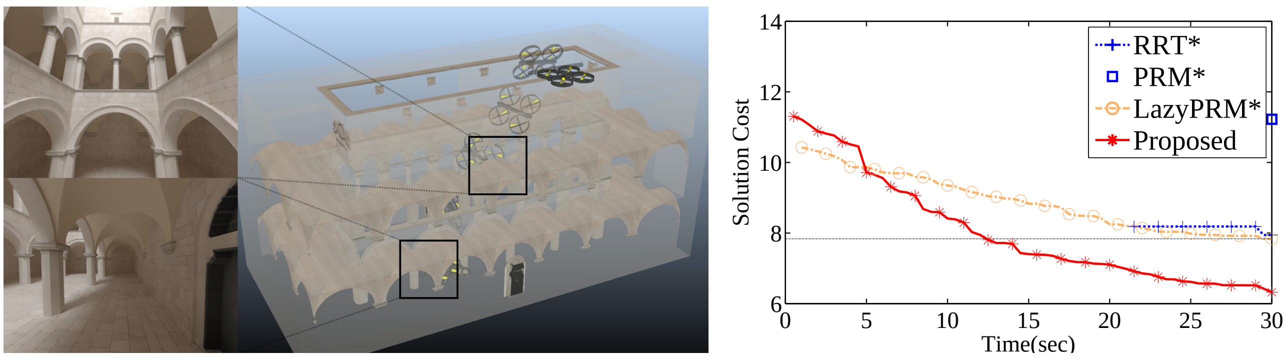

Fig. 6. A 6-DOF quadrotor planning in a *Sponza* scene consisting of 66K triangles. The Sponza scene is a three-story Gothic building with a several pillars, which causes optimistic errors and relatively heavy computational overhead of collision checking. In this model, our method achieves a final cost of 6.32, which is faster by a factor of two over the other methods. Note that the result of RRT* is not reported before 20s, since all of 10 trials is not able to find even an initial solution up to that point.

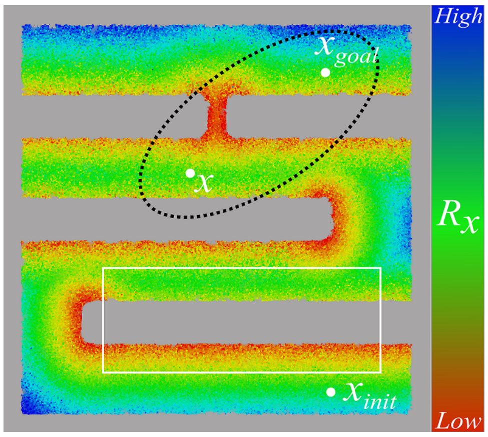

Fig. 7. An illustration of biased convergence of $\widetilde{F}$ with a heat map showing r_x, approximate minimum distance to X_{obs} where r_x is encoded by red colors for low values and blue for high. We can observe that r_x in white rectangle area is biased to the closest obstacle towards x_{goal}. Dotted enclosed elliptical area indicates a region in which promising edges can exist in lazy manner.

asymptotically increase as the number of samples in G grows as we have seen in Sec. III-B. It is the another reason why the major computational bottleneck, shifts from collision checking to the graph update depending on the shape of given configuration space. The proposed method, however, successfully reduces the overhead of graph update(SP) even in the Fig. 6 benchmarks by early rejecting the problematic expansions while preserving the efficiency of lazy collision checking. This demonstrates the robustness of the proposed method as well as the low computational overhead of the proposed algorithm.

VI. Analysis

In this section, we discuss asymptotic optimality and the biased convergence of free space approximation in our algorithm.

A. Asymptotic Optimality

Let $E_{proposed}$ refer to the edges in $G_{proposed}$ constructed by the proposed method. E_{PRM^*} and $E_{lazyPRM^*}$ are defined similarly for naïve PRM* and lazy PRM*, respectively. For the sake of the simplicity, we assume that a sequence of random samples in different methods is identical and ignore the cases of divergence by $UpdateDSPT(\cdot)$.

As depicted in Alg. 4, the proposed algorithm follows identical steps of PRM* except only for the collision checking during expansion. This makes E_{PRM^*} a subset of $E_{proposed}$, since our adaptive collision checking generates only false negative – accepting infeasible edges, while not rejecting valid edges – which results in $E_{proposed}$ has more edges than E_{PRM^*}. Likewise, lazy PRM* accepts all of the edges under the optimistic assumption during the expansion, which makes our method in-between PRM* and lazy PRM* in terms of the cardinality of E. Therefore, we have a relationship of $E_{PRM^*} \subset E_{proposed} \subset E_{lazyPRM^*}$. Consequently, the optimality and completeness of PRM* and lazy PRM* are applied to the proposed algorithm [13].

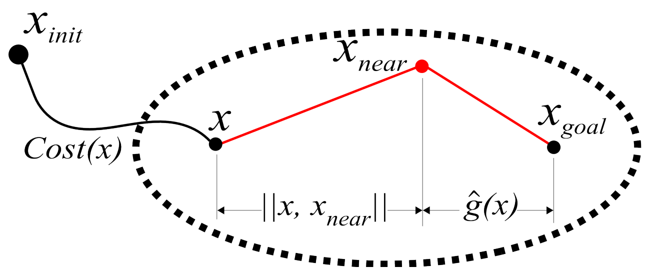

Fig. 8. An illustration of Eq. 2 corresponding to the elliptical area in Fig. 7. It denotes a restricted domain for edge expansions from x to a subset of near neighbors caused by lazy collision checking.

B. Biased convergence of $\widetilde{F}$

Interestingly, the behavior of the proposed algorithm is not only governed by configuration free space approximation $\widetilde{F}$, but also affects the convergence of $\widetilde{F}$.

When we update the approximate free space in the non-lazy collision checking with uniform sampling, $\widetilde{F}_x$ is updated and trimmed by witnesses in all possible directions. On the other hand, when $\widetilde{F}_x$ is updated with lazy collision checking, we consider edges from an arbitrary configuration x to those nearby nodes, only when those edges are expected to be a part of the solution path.

To be specific, for a configuration x, only edges from x to $x_{near} \in X'_{near}$ satisfying Eq. 2 can reduce $Cost(x_{goal})$; no other edges from x are never considered for the graph expansion.

$$\begin{aligned} X'_{near} = \{x_{near} \in X_{near} \mid Cost(x) + \|x, x_{near}\| \\ + \hat{g}(x_{near}) < Cost(x_{goal})\}, \end{aligned} \tag{2}$$

where $Cost(x)$ and $\hat{g}(x_{near})$ stand for a cost-to-come from x_{init} to x and an admissible cost-to-go from x_{near} to x_{goal}, respectively. Fig. 8 shows Eq. 2 in an illustrative way. The proposed algorithm uses DSPT(Dynamic Shortest Path Tree) structure for finding the shortest path in search graph G at the end of each iteration as shown in line 12-13 in Alg. 4. For this reason, all of the edges newly added to the solution path satisfy Eq. 2 implicitly in the nature of the proposed algorithm. $Cost(x)$ for all $x \in V$ are also updated during the DSPT update which is necessary for shortest path computation. For the details, refer to the original paper [21].

Furthermore, collision checking in Alg. 2 always returns an empirical collision close to x_{from}, which leads to a biased result of $\widetilde{F}$. Fig. 7 shows such an example where heat map shows distribution of r_x, the distance from each configuration to the closest $\mathbf{X}_{obs}$ approximated by the proposed method.

These properties seem to harm the completeness of $\widetilde{F}$. However, it results in narrowing down the domain of empirical collision set for $\widetilde{F}_x$. For a configuration x, probabilistically more edges towards promising directions will be checked during execution, and thus biased $\widetilde{F}_x$ can provide better understanding for a configuration x than the non-lazy case. Consequently, it can be beneficial for adaptive collision checking since biased $\widetilde{F}_x$ reflects the majority of outgoing edges from a configuration x.

VII. Conclusion

Our motivation for this work began with the questions, how false negative errors of lazy collision checking affects the performance of sampling-based motion planner, and how we can alleviate it. To this end, we have proposed an adaptive lazy collision checking with configuration free space approximation. The approximate free space information guides which areas should be checked early to avoid the optimistic thrashing while preserving the efficiency of lazy collision checking.

Our method provides meaningful results over the prior methods across all the tested benchmarks with varying dimensions. We also have shown properties of the configuration free space approximation in conjunction with lazy collision checking.

As a future work, it is worth studying the representation of configuration free space for better accuracy and flexibility. Furthermore, as shown in Fig. 2, a possible research direction could be to improve a graph manipulation and nearest neighbor search for higher scalability with respect to the nature of optimal sampling-based planning.

References

[1] Steven M LaValle, "Rapidly-exploring random trees a new tool for path planning", Tech. Rep. 98-11, Iowa State University, 1998.

[2] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark 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.

[3] J. Kuffner and S.M. LaValle, "RRT-connect: An efficient approach to single-query path planning", in *IEEE Int'l. Conf. on Robotics and Automation*, 2000, pp. 995–1001.

[4] Shawna Thomas, Marco Morales, Xinyu Tang, and Nancy M Amato, "Biasing samplers to improve motion planning performance", in *IEEE Int'l. Conf. on Robotics and Automation*. IEEE, 2007, pp. 1625–1630.

[5] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner", in *IEEE Int'l. Conf. on Robotics and Automation*, 2008, pp. 3743–3750.

[6] Junghwan Lee, OSung Kwon, Liangjun Zhang, and Sung-eui Yoon, "SR-RRT: Selective retraction-based RRT planner", in *IEEE Int'l. Conf. on Robotics and Automation*, 2012, pp. 2543–2550.

[7] Sertac Karaman and Emilio Frazzoli, "Sampling-based algorithms for optimal motion planning", *Int'l. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[8] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2011, pp. 2640–2645.

[9] Donghyuk Kim, Junghwan Lee, and Sung-eui Yoon, "Cloud RRT*: Sampling cloud based RRT*", in *IEEE Int'l. Conf. on Robotics and Automation*, 2014, pp. 2519–2526.

[10] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004.

[11] Jingru Luo and Kris Hauser, "An empirical study of optimal motion planning", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2014, pp. 1761–1768.

[12] R. Bohlin and E.E. Kavraki, "Path planning using lazy prm", in *IEEE Int'l. Conf. on Robotics and Automation*, 2000, vol. 1, pp. 521–528 vol.1.

[13] Kris Hauser, "Lazy collision checking in asymptotically optimal motion planning", in *IEEE Int'l. Conf. on Robotics and Automation*, 2015.

[14] M. Foskey, M. Garber, M. Lin, and D. Manocha, "A voronoi-based hybrid motion planner", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2001, vol. 1, pp. 55–60.

[15] Joshua Bialkowski, Michael Otte, and Emilio Frazzoli, "Free-configuration biased sampling for motion planning", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*. IEEE, 2013, pp. 1272–1279.

[16] Jia Pan, Christian Lauterbach, and Dinesh Manocha, "g-planner: Real-time motion planning and global navigation using gpus", in *AAAI Conference on Artificial Intelligence*, 2010.

[17] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space", *arXiv preprint arXiv:1109.3145*, 2011.

[18] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[19] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2011, pp. 4307–4313.

[20] Joshua Bialkowski, Sertac Karaman, Michael Otte, and Emilio Frazzoli, "Efficient collision checking in sampling-based motion planning", in *Int'l. Workshop on the Algorithmic Foundations of Robotics*, pp. 365–380. 2013.

[21] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni, "Fully dynamic algorithms for maintaining shortest paths trees", *Journal of Algorithms*, vol. 34, no. 2, pp. 251–281, 2000.

[22] Mark Moll, Ioan A. Şucan, and Lydia E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization", *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, September 2015.

[23] M. Freese E. Rohmer, S. P. N. Singh, "V-REP: a versatile and scalable robot simulation framework", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2013.

[24] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs", in *Robotics and Automation (ICRA), IEEE International Conference on*. IEEE, 2015, pp. 3067–3074.