2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Asymptotically-optimal motion planning using lower bounds on cost

Oren Salzman* and Dan Halperin*

Abstract— **Many path-finding algorithms on graphs such as A* are sped up by using a heuristic function that gives lower bounds on the cost to reach the goal. Aiming to apply similar techniques to speed up sampling-based motion-planning algorithms, we use effective lower bounds on the cost between configurations to tightly estimate the cost-to-go. We then use these estimates in an anytime asymptotically-optimal algorithm which we call Motion Planning using Lower Bounds (MPLB). MPLB is based on the Fast Marching Trees (FMT*) algorithm [1] recently presented by Janson and Pavone. An advantage of our approach is that in many cases (especially as the number of samples grows) the weight of collision detection in the computation is almost negligible compared to the weight of nearest-neighbor queries. We prove that MPLB performs no more collision-detection calls than an anytime version of FMT*. Additionally, we demonstrate in simulations that for certain scenarios, the algorithmic tools presented here enable efficiently producing low-cost paths while spending only a small fraction of the running time on collision detection.**

I. Introduction

Motion-planning algorithms aim to find a collision-free path for a robot moving amidst obstacles. A common approach is to use sampling-based techniques [2, C. 7]. These algorithms sample points in the robot's *configuration-space* (C-space) and connect near-by configurations to construct a graph called a *roadmap*. Often, a *low-cost* path is desired, where cost can be measured in terms of, for example, path length or energy consumption along the path.

Sampling-based algorithms rely on two central primitive operations: *Collision Detection (CD)* and *Nearest Neighbors (NN)* search. CD determines whether a configuration is collision-free or not and is often used to assess if a path connecting near-by configurations is collision-free. This latter operation is referred to as *Local Planning* (LP). An NN data structure preprocesses a set of points to efficiently answer queries such as "which are the points within radius r of a given query point?" In practice, the cost of CD, primarily due to LP calls, often dominates the running time of sampling-based algorithms, and is typically regarded as the computational bottleneck for such algorithms.

In their influential work, Karaman and Frazzoli [3] analyzed existing sampling-based algorithms (namely PRM [4] and RRT [5]) and introduced the notion of *asymptotic optimality* (AO); an algorithm is said to be AO if the cost of the returned solution converges to the cost of the optimal solution, when the algorithm is run for sufficiently long time. They proposed AO variants of PRM and RRT called PRM* and RRT*, respectively. However, the AO of PRM* and RRT* comes at the price of increased running time and memory consumption when compared to their non-optimal counterparts. To reduce this additional price, several improvements were proposed which modify the sampling scheme [6], [7], the CD [8], or relax the optimality to *asymptotic near-optimality* (ANO) [9], [10], [11], [12]. An algorithm is said to be ANO if, given an approximation factor ε, the cost of the returned solution is guaranteed to converge to within a factor of $1+\varepsilon$ of the cost of the optimal solution.

Following the introduction of PRM* and RRT*, other AO algorithms have been suggested such as RRT# [13], and the Fast Marching Trees (FMT*) [1] algorithm. FMT* which is reviewed in detail in Section II, was shown to converge to an optimal solution faster than PRM* or RRT*.

Contribution and paper organization. We show that by examining the roadmap induced by a set of samples, we can compute *effective lower bounds* on the cost-to-go of nodes (the cost to reach the goal). This is done without performing expensive LP calls and allows to efficiently guide the search performed by the algorithm. We call our scheme Motion Planning using Lower Bounds or MPLB for short.

An implication of our approach is that the weight of CD and LP in the overall running time of the algorithm becomes negligible when compared to that of NN calls[1]. Bialkowski et al. [8] introduced a technique which replaces CD calls by NN calls. Their scheme relies on additional data produced by the CD algorithm used, namely, a bound on the clearance of a configuration (if it is collision free) or on its penetration depth (if it is not collision free). However, such a bound is not trivial to compute for some prevalent C-spaces. In contrast, our algorithmic framework can be paired with existing off-the-shelf NN, CD and LP procedures. Thus, our results, which are more general (as they are applicable to general C-spaces), strengthen the conjecture of Bialkowski et al. [8] that NN computation, and not CD, may be the bottleneck of sampling-based motion-planning algorithms.

This work continues and expands our recent work [11], where we relaxed the AO of RRT* to ANO using lower bounds. The novel component here is that multiple nodes are processed *simultaneously* while in RRT* the nodes are processed one at a time. This allows to efficiently compute for all nodes an estimation of the cost-to-go, which in turn can be used to speed up motion-planning algorithms.

* Blavatnik School of Computer Science, Tel-Aviv University, Israel

This work has been supported in part by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

[1] Throughout this paper, when we say an NN call we mean a call to find all the nodes within radius $r(n)$ of a given node. $r(n)$ is a radius depending on the number of samples used and will be formally defined in Section II.

Our framework is demonstrated for the case where distance is the cost function. We apply our approach to FMT* which is reviewed in Section II. As we wish to work in an *anytime* setting, we introduce in Section III a straightforward adaptation of FMT* for anytime planning which we call aFMT*. We then proceed to present MPLB in Section IV. We analyze aFMT* and MPLB with respect to the amount of calls to the NN and LP procedures in Section V and report on experimental results in Section VI. Specifically, we demonstrate in simulations that for certain scenarios, MPLB produces lower-cost paths faster (by a factor of between two and three) than aFMT*. We conclude with a discussion and suggestions for future work in Section VII.

II. Terminology and Algorithmic Background

We begin by formally stating the motion-planning problem and several standard procedures used by sampling-based algorithms. We continue by reviewing the FMT* algorithm.

A. Problem definition and terminology

Let $\mathcal{X} \subseteq \mathbb{R}^d$ denote the C-space[2], and let $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ denote the free space. Let $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$ be the motion-planning problem where: $x_{\text{init}} \in \mathcal{X}_{\text{free}}$ is the initial free configuration of the robot and $\mathcal{X}_{\text{goal}} \subseteq \mathcal{X}_{\text{free}}$ is the goal region. We will make use of the following procedures: $\texttt{sample\_free}(n)$ returns n random configurations from $\mathcal{X}_{\text{free}}$; $\texttt{nearest\_neighbors}(x, V, r)$ returns all the nodes within the set V whose distance from x is less than r; $\texttt{collision\_free}(x, y)$ tests whether the straight-line segment connecting x and y is contained in $\mathcal{X}_{\text{free}}$; $\texttt{cost}(x, y)$ returns the cost of the straight-line path connecting x and y, namely, in our case, the distance. Given a tree $\mathcal{T}$ rooted at x_{init}, $\texttt{path}(v, \mathcal{T})$ returns the path from x_{init} to v. Let $\mathcal{G} = (V, E)$ be a weighted graph, where the weight of an edge $(x, y) \in E$ is $\texttt{cost}(x, y)$. Given such a graph $\mathcal{G}$, we denote by $\texttt{cost}_{\mathcal{G}}(x, y)$ the cost of the weighted shortest path from x to y. Let $\texttt{cost-to-come}_{\mathcal{G}}(x)$ be $\texttt{cost}_{\mathcal{G}}(x_{\text{init}}, x)$ and $\texttt{cost-to-go}_{\mathcal{G}}(x)$ be $\min_{x_{\text{goal}} \in \mathcal{X}_{\text{goal}}} \{\texttt{cost}_{\mathcal{G}}(x, x_{\text{goal}})\}$. Namely for every node x, $\texttt{cost-to-come}_{\mathcal{G}}(x)$ is the minimal cost to reach x from x_{init} and $\texttt{cost-to-go}_{\mathcal{G}}(x)$ is the minimal cost to reach $\mathcal{X}_{\text{goal}}$ from x. Additionally, let $B_{\mathcal{G}}(x_{\text{init}}, r)$, $B_{\mathcal{G}}(\mathcal{X}_{\text{goal}}, r)$ be the set of all nodes whose cost-to-come (respectively, cost-to-go) value on $\mathcal{G}$ is smaller than r. Finally, we denote by $\texttt{Dijkstra}(G, x, c)$ an implementation of Dijkstra's algorithm running on the graph G from x until a maximal cost of c has been reached. The algorithm's implementation updates the cost to reach each node from x and outputs the set of nodes traversed.

Given a set of samples V, and a radius r, we denote by $G(V, r)$ the *disk graph*,[3] which is the graph whose set of vertices is V and two vertices $x, y \in V$ are connected by an edge iff the distance between x and y is less than r.

[2]Although we describe the algorithm for Euclidean spaces, by standard techniques (see, e.g. [2, Section 3.5, Section 7.1.2]) it can be applied to non-Euclidean spaces such as SE3. However, the AO proof of FMT*, presented in [1], and on which we rely on here, is shown only for Euclidean spaces.

[3]The disk graph is sometimes referred to as the *neighborhood graph*.

Algorithm 1 FMT* $(x_{\text{init}}, \mathcal{X}_{\text{goal}}, n)$

1: $V \leftarrow \{x_{\text{init}}\} \cup \texttt{sample\_free}(n)$; $E \leftarrow \emptyset$; $\mathcal{T} \leftarrow (V, E)$
2: PATH $\leftarrow \texttt{search}\ (\mathcal{T}, \mathcal{X}_{\text{goal}}, 0, \infty)$ // See Alg. 2
3: **return** PATH

B. Fast Marching Trees (FMT*)

FMT*, outlined in Alg. 1, performs a "lazy" dynamic programming recursion on a set of sampled configurations to grow a tree rooted at x_{init} [1]. The algorithm samples n collision-free nodes V (line 1). It searches for a path to $\mathcal{X}_{goal}$ by building a minimum-cost spanning tree $\mathcal{T}$ growing in cost-to-come space (line 2 and detailed in Alg. 2). As we explain in Section IV, the algorithm can benefit from using a heuristic function which estimates the cost-to-go of a node and a bound on the maximal length of the path that should be found. As these are not part of the original formulation of FMT*, we describe the search procedure of FMT* using a cost-to-go estimation of zero for each node and an unbounded maximal path length (marked in red in Alg. 1 and 2). This will allow us to use the same pseudo-code of Alg. 2 to explain the MPLB algorithm in Section IV.

The search-tree $\mathcal{T}$ is built by maintaining two sets of nodes H, W, such that H is the set of nodes added to $\mathcal{T}$ that may be expanded and W is the set of nodes that have not yet been added to $\mathcal{T}$ (Alg. 2, line 1). It then computes for each node of V the set of nearest neighbors of radius $r(n)$ (line 3). FMT* repeats the following process: the node z with the lowest cost-to-come value is chosen from H (line 4 and 16). For each neighbor x of z that is not already in H, the algorithm finds its neighbor $y \in H$ such that the cost-to-come of y added to the distance between y and x is minimal (lines 7-9). If the local path between y and x is free, x is added to H with y as its parent (lines 10-12). At the end of each iteration z is removed from H (line 13). If all nodes have been processed and no path to the goal has been found (line 15), FMT* terminates with FAILURE. Once the goal is reached (line 19) a path to $z \in \mathcal{X}_{\text{goal}}$ is returned.

To ensure AO, the radius $r(n)$ used by the algorithm is

$$r(n) = (1+\eta) \cdot 2 \left(\frac{1}{d}\right)^{\frac{1}{d}} \left(\frac{\mu(\mathcal{X}_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}} \left(\frac{\log n}{n}\right)^{\frac{1}{d}}, \quad (1)$$

where $\eta > 0$ is some small constant (see [1]), $\mu(\cdot)$ denotes the d-dimensional Lebesgue measure and ζ_d is the volume of the unit ball in the d-dimensional Euclidean space.

III. Anytime FMT* (aFMT*)

An algorithm is said to be *anytime* if it keeps improving the quality of the solution as time permits. We outline a straightforward enhancement to FMT* to make it anytime. As noted in previous work (see, e.g., [14]) one can turn a batch algorithm into an anytime one by the following general approach: choose an initial small number of samples $n = n_0$ and apply the algorithm. As long as time permits, double n and repeat the process. The total running time is less than twice that of the running time for the largest n. Note that as FMT* is AO, aFMT* is also AO.

Algorithm 2 search $(\mathcal{T}, \mathcal{X}_{\text{goal}}, \texttt{cost\_to\_go}, c_{\max})$

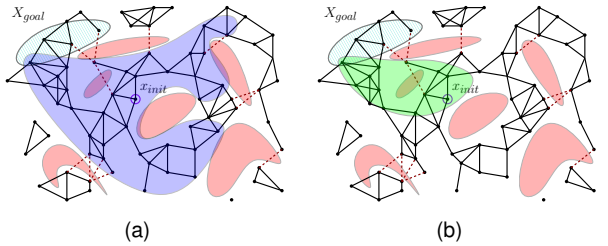
1: $H \leftarrow \{x_{\text{init}}\}$; $W \leftarrow V \setminus \{x_{\text{init}}\}$
2: **for all** $v \in V$ **do**
3: $N_v \leftarrow \texttt{nearest\_neighbors}(v, V \setminus \{v\}, r(n))$
4: $z \leftarrow x_{\text{init}}$
5: **while** $z \notin \mathcal{X}_{\text{goal}}$ **do**
6: $H_{\text{new}} \leftarrow \emptyset$; $X_{\text{near}} \leftarrow W \cap N_z$
7: **for** $x \in X_{\text{near}}$ **do**
8: $Y_{\text{near}} \leftarrow H \cap N_x$
9: $y_{\min} \leftarrow \arg\min_{y \in Y_{\text{near}}} \{\texttt{cost}_{\mathcal{T}}(y) + \texttt{dist}(y, x)\}$
10: **if** $\texttt{collision\_free}(y_{\min}, x)$ **then**
11: $\mathcal{T}.\texttt{parent}(x) \leftarrow y_{\min}$
12: $H_{\text{new}} \leftarrow H_{\text{new}} \cup \{x\}$; $W \leftarrow W \setminus \{x\}$
13: $H \leftarrow (H \cup H_{\text{new}}) \setminus \{z\}$
14: **if** $H = \emptyset$ **then**
15: **return** FAILURE
16: $z \leftarrow \arg\min_{y \in H} \{\texttt{cost}_{\mathcal{T}}(y) + \texttt{cost\_to\_go}(y)\}$
17: **if** $\texttt{cost}_{\mathcal{T}}(z) + \texttt{cost\_to\_go}(z) \geq c_{\max}$ **then**
18: **return** FAILURE
19: **return** $\texttt{path}(z, \mathcal{T})$

IV. Algorithmic Framework

We are now ready to present our approach to exploiting lower bounds on cost in order to speed up sampling-based motion-planning algorithms such as aFMT*.

Given a random infinite sequence of collision-free samples $S = s_1, s_2 \ldots$ denote by $V_i(S)$ the set of the first 2^i elements of S. Let $\mathcal{G}_i(S) = G(V_i(S), r(|V_i(S)|))$ (here $r(n)$ is the radius defined in Eq. 1) and let $\mathcal{H}_i(S) \subseteq \mathcal{G}_i(S)$ be the sub-graph containing collision-free edges only . For brevity, we omit S when referring to $V_i(S), \mathcal{G}_i(S)$ and $\mathcal{H}_i(S)$. Moreover, when we compare our new algorithm to the aFMT* algorithm, we do so for runs on the same random infinite sequence S. Clearly, for any two nodes $x, y \in V_i$, $\texttt{cost}_{\mathcal{G}_i}(x, y) \leq \texttt{cost}_{\mathcal{H}_i}(x, y)$. Thus for any node $x \in V_i$, $\texttt{cost-to-go}_{\mathcal{G}_i}(x) \leq \texttt{cost-to-go}_{\mathcal{H}_i}(x)$. Namely, the cost-to-go computed using the disk graph $\mathcal{G}_i$ is a lower bound on the cost-to-go that may be obtained using $\mathcal{H}_i$. We call this the *lower bound property*. For an illustration, see Fig. 1.

We present Motion Planning using Lower Bounds, or MPLB (outlined in Alg. 3). Similar to aFMT*, the algorithm runs in iterations and at the i'th iteration, uses V_i as its set of samples. Unlike aFMT*, each iteration of MPLB consists of a *preprocessing phase* (line 4) of computing a lower bound on the cost-to-go values and a *searching phase* (line 5) where a modified version of FMT* is used.



(a) (b)

Fig. 1: This figure demonstrates that the part of the tree expanded when searching in cost-to-come space (shaded blue region, Fig. (a)) is larger than the one expanded when searching in cost-to-come+cost-to-go space (shaded green region, Fig. (b)). Obstacles in the C-space are depicted in red, start location and goal region are depicted by a purple circle and a turquoise region, respectively. Edges of the disk graph $\mathcal{G}_i$ that are contained and not contained in $\mathcal{H}_i$ are depicted in black and dashed red, respectively. The figure is best viewed in color.

Let $c_i(\text{ALG})$ denote the cost of the solution obtained by an algorithm ALG using V_i as the set of samples (set $c_0(\text{ALG}) \leftarrow \infty$). We now show that only a subset of the nodes sampled in each iteration need to be considered. We then proceed to describe the two phases of MPLB.

A. Promising nodes

We exploit the lower-bound property to consider only a *subset* of V_i that will be used in the i'th iteration. Intuitively, we only wish to consider nodes that may produce a solution that is better than the solution obtained in previous iterations. This leads to the definition of promising nodes:

Definition 1: A node $x \in V_i$ is *promising* (at iteration i) if $\texttt{cost-to-come}_{\mathcal{H}_i}(x) + \texttt{cost-to-go}_{\mathcal{H}_i}(x) < c_{i-1}(\text{MPLB})$.

Note that if $c_{i-1}(\text{MPLB}) = \infty$ (namely, no solution was found in the previous iteration) then all nodes are promising. In the preprocessing phase, MPLB will traverse $\mathcal{G}_i$ (and not $\mathcal{H}_i$) to collect a set of nodes that contains all promising nodes (and possibly other nodes), compute a lower bound on their cost-to-go and use this set in the searching phase.

B. Preprocessing phase: Estimating the cost-to-go

Recall that in the preprocessing phase, outlined in Alg. 4 and visualized in Fig. 2, we compute a lower bound on the cost-to-go for (a subset of) nodes $x \in V_i$. This is done by collecting the set of nodes $V_{\text{preproc}} = B_{\mathcal{G}_i}\left(x_{\text{init}}, \frac{c_{i-1}(\text{MPLB})}{2}\right) \cup B_{\mathcal{G}_i}\left(\mathcal{X}_{\text{goal}}, \frac{c_{i-1}(\text{MPLB})}{2}\right)$. Namely, by performing one traversal from x_{init} (line 1) and one traversal from $\mathcal{X}_{\text{goal}}$ (line 2), all nodes such that $\texttt{cost-to-come}_{\mathcal{G}_i} \leq \frac{c_{i-1}(\text{MPLB})}{2}$ or $\texttt{cost-to-go}_{\mathcal{G}_i} \leq \frac{c_{i-1}(\text{MPLB})}{2}$ are found. Clearly, any node *not* in either set is not promising (lines 3-4).

After collecting all nodes in V_{preproc}, which includes the promising nodes and possibly some other nodes as well, the

Algorithm 3 MPLB $(x_{\text{init}}, \mathcal{X}_{\text{goal}}, n_0)$

1: $V_0 \leftarrow \{x_{\text{init}}\}$; $n \leftarrow n_0$; $c_{\text{prev}} \leftarrow \infty$; $i \leftarrow 1$
2: **while** $\texttt{time\_permits()}$ **do**
3: $V_i \leftarrow V_{i-1} \cup \texttt{sample\_free}(n)$; $E \leftarrow \emptyset$; $\mathcal{T} \leftarrow (V_i, E)$;
4: $\texttt{estimate\_cost\_to\_go}(V_i, x_{init}, \mathcal{X}_{goal}, c_{\text{prev}})$
5: PATH $\leftarrow \texttt{search}\ (\mathcal{T}, \mathcal{X}_{goal}, \texttt{cost\_to\_go}, c_{\text{prev}})$
6: $n \leftarrow 2n$; $i \leftarrow i + 1$
7: $c_{\text{prev}} = \min\{\texttt{cost}(\text{PATH}), c_{\text{prev}}\}$
8: **return** PATH

Algorithm 4 estimate_cost_to_go $(V, x_{\text{init}}, \mathcal{X}_{\text{goal}}, c)$

1: $V_{\text{preproc}} \leftarrow \texttt{Dijkstra}(G(V, r(|V|)), x_{\text{init}}, \frac{c}{2})$
2: $V_{\text{preproc}} \leftarrow V_{\text{preproc}} \cup \texttt{Dijkstra}(G(V, r(|V|)), \mathcal{X}_{\text{goal}}, \frac{c}{2})$
3: **for** $x \in V \setminus V_{\text{preproc}}$ **do**
4: cost_to_go$(x) \leftarrow \infty$
5: $\texttt{Dijkstra}(G(V_{\text{preproc}}, r(|V_{\text{preproc}}|)), \mathcal{X}_{\text{goal}}, c)$

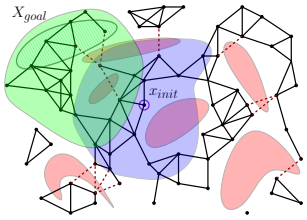


Fig. 2: Visualization of Alg. 4. Shaded blue and green regions represent the first and second traversal from the start location x_{init} and the goal region X_{goal} described in lines 1 and 2, respectively. V_{preproc} is the set of nodes in the union of these colored regions. All other nodes cannot be promising, thus their cost-to-go is set to ∞ (lines 3-4). The third traversal assigns cost-to-go estimates for all nodes in V_{preproc} (line 5). Figure best viewed in color.

distance of every such node from $\mathcal{X}_{\text{goal}}$ is computed (line 5). This is done by running a shortest path algorithm on the graph $\mathcal{G}_i$ restricted to the nodes in V_{preproc}. This distance is stored for each node and will be used as a lower bound on the cost-to-go. Notice that this phase only uses NN calls and does not perform any CD calls (as there are no LP calls).

C. Searching phase: Using cost-to-go estimations

The lower bounds computed in the preprocessing phase allow for two algorithmic enhancements to the searching phase when compared to aFMT*: (i) incorporating the cost-to-go estimation in the ordering scheme of the nodes and (ii) discarding nodes that are found to be not promising.

Node ordering: Recall that in aFMT*, H is the set of nodes added to the tree that may be expanded and that these nodes are ordered according to their cost-to-come value (Alg. 2, line 16). Instead, MPLB uses the cost-to-come added to the cost-to-go estimation to order the nodes in H. This follows exactly the formulation of A* [15], which performs a Dijkstra-like search on a set of nodes. The nodes that were encountered but not processed yet (H in our setting) are ordered according to a cost function $f(x) = g(x) + h(x)$ defined for a given node x. Here, $g(x)$ is the (computed) cost-to-come value of x ($\texttt{cost-to-come}_{\mathcal{H}_i}(x)$ in our case) and $h(x)$ is a lower bound on the cost-to-go of x to the goal ($\texttt{cost-to-go}_{\mathcal{G}_i}(x)$ in our case). aFMT* essentially uses the trivial heuristic $h = 0$.

Discarding nodes: In the preprocessing stage MPLB computes a set of nodes that *may* be promising, though for each such node, the cost-to-come value was not computed. In the searching phase, once a node is added to the tree, its cost-to-come value will not change in the current iteration. Thus, every node x added to the tree with $\texttt{cost-to-come}_{\mathcal{H}_i}(x) + \texttt{cost-to-go}_{\mathcal{G}_i}(x) \geq c_{i-1}(\text{MPLB})$ is discarded as it cannot be promising. This implies that MPLB will terminate an iteration when it is evident that the previous iteration's solution cannot be improved (see Alg. 2, lines 17-18).

In Section VI we demonstrate through various simulations that using lower bounds has a significant effect on the running time of the algorithm in practice. Ordering the nodes using a heuristic that tightly estimates the cost-to-go allows MPLB to expand a smaller portion of the nodes V_i while discarding nodes allows to focus the search only on nodes that may potentially improve the existing solution.

V. Comparative analysis and Discussion

We compare aFMT* and MPLB with respect to the size of the tree constructed and the primitive procedures calls, namely NN and LP. This is done by quantifying the number of NN and LP calls performed by both algorithms.

Let $\#_{\text{NN},i}(\text{ALG})$, $\#_{\text{LP},i}(\text{ALG})$ denote the number of NN and LP calls performed by an algorithm ALG at iteration i, for a fixed sequence of samples S. Recall that when comparing the two algorithms, it is done for the same sequence S.

A. Search-tree size

Let $V_i(\text{ALG}) \subseteq V_i$ denote the set of nodes in the tree in the i'th iteration of an algorithm ALG.

Lemma 1: At every iteration, the set of nodes traversed in MPLB's searching phase is not larger than that of aFMT*.

Proof: Every node x in the tree of aFMT* has cost-to-come not larger than $c_i(\text{aFMT}^*)$. Thus, the size of the search-tree of aFMT* is: $|V_i(\text{aFMT*})| = |\{x \in V_i \mid \texttt{cost-to-come}_{\mathcal{H}_i}(x) \leq c_i(\text{aFMT}^*)\}|$.

Similar to aFMT*, each node x traversed by MPLB in the searching phase has $\texttt{cost-to-come}_{\mathcal{H}_i}(x) + \texttt{cost-to-go}_{\mathcal{G}_i}(x) \leq c_i(\text{aFMT}^*)$. Additionally, due to node discarding (see Section IV), $\texttt{cost-to-come}_{\mathcal{H}_i}(x) + \texttt{cost-to-go}_{\mathcal{G}_i}(x) \leq c_{i-1}(\text{MPLB})$. Thus, the size of the search-tree of MPLB is: $|V_i(\text{MPLB})| = |\{x \in V_i \mid \texttt{cost-to-come}_{\mathcal{H}_i}(x) + \texttt{cost-to-go}_{\mathcal{G}_i}(x) \leq \min\{c_{i-1}(\text{MPLB}), c_i(\text{aFMT}^*)\}$. This, in turn, implies that $|V_i(\text{MPLB})| \leq |V_i(\text{aFMT*})|$. ■

B. Nearest neighbor calls (NN)

To quantify the number of NN queries performed by each algorithm we note the following observations (explained in detail in the extended version of this paper [16]):

Observation 1: The number of NN calls performed by aFMT* can be bounded from below as follows: $\#_{\text{NN},i}(\text{aFMT}^*) \geq |V_i(\text{aFMT*})|$.

Observation 2: The number of NN calls performed by MPLB is: $\#_{\text{NN},i}(\text{MPLB}) = \left|\left\{x | x \in B_{\mathcal{G}_i}\left(x_{\text{init}}, \frac{c_{i-1}(\text{MPLB})}{2}\right) \cup B_{\mathcal{G}_i}\left(\mathcal{X}_{\text{goal}}, \frac{c_{i-1}(\text{MPLB})}{2}\right)\right\}\right|$.

Thus, MPLB *may* perform more NN queries than FMT*.

C. Local planning calls (LP)

The LP will be called whenever either algorithm (aFMT* or MPLB) attempts to insert a node to the search-tree (line 10 in Alg. 2). Thus we can state the following lemma:

Lemma 2: If MPLB performs an LP call for the edge (x, y) in the i'th iteration then aFMT* will perform an LP call for the edge (x, y) as well.

Proof: The LP procedure will be called for every pair of nodes x, y such that: (i) x, y are neighbors in $\mathcal{G}_i$ (namely their distance is less than $r(|V_i|)$), (ii) $\texttt{cost-to-come}_{\mathcal{H}_i}(x) < \texttt{cost-to-come}_{\mathcal{H}_i}(y)$, and (iii) the edge (z, y) is not

collision-free for all other neighbors z of y in the tree that could potentially lead to smaller cost-to-come values of y.

If MPLB performs an LP call for the edge (x, y) then conditions (i),(ii) and (iii) hold for the samples x, y in MPLB. To prove the lemma we show that they hold for the samples x, y in aFMT*. Condition (i) holds trivially as it is a property of the samples. Note that the cost-to-come of any node z computed by both algorithms equals to $\texttt{cost-to-come}_{\mathcal{H}_i}(z)$. Using this and that $V_{\text{preproc}} \subseteq V_i$ (namely the set of nodes used by MPLB is a subset of the set used by aFMT*), conditions (ii) and (iii) hold as well. ■

D. Discussion

From the above analysis we conclude that MPLB will perform *no more* LP calls than aFMT*. It *may* perform *more* NN calls than aFMT*. As we demonstrate empirically in Section VI, the number of NN calls that MPLB performs may actually be smaller than that of aFMT*. Moreover, as the number of iterations increases, MPLB performs only a tiny fraction of the number of LP calls performed by aFMT*.

VI. Evaluation

We present simulations evaluating the performance of MPLB on 2, 3 and 6 dimensional C-spaces. All experiments were run on a 2.8GHz Intel Core i7 processor with 8GB of memory. The MPLB and aFMT* implementations are based on the FMT* implementation provided by Pavone's research group using the Open Motion Planning Library (OMPL 0.10.2) [17]. Each result is averaged over one hundred runs. Scenarios and additional material are available at `http://acg.cs.tau.ac.il/projects/MPLB`.

The AO proof of FMT* (and thus of aFMT* and MPLB) relies on the fact that the C-space is Euclidean. Thus, as a warm-up we start by studying the motion of robots translating in the plane and in space (Fig. 3a and 3b). Next, we continue to examine the behavior of the algorithms in SE3 (Fig. 3c and 3d). Here the radius provided for FMT* (Eq. 1) is irrelevant due to the differences in the rotational and translational components of the C-space. Hence, for both aFMT* and MPLB, we chose to connect each node to its k NN, where $k(n) = 9 \log n$: a variant of RRG was proposed [3] where each node is connected to its k_{RRG} NN for $k_{RRG}(n) \geq 2e \log n$. Although this variant was analyzed for Euclidean spaces only, applying it to non-Euclidean spaces works well in practice (see, e.g. [11]).

A. Fast convergence to high-quality solutions

We start by comparing the cost of a solution obtained by aFMT* and MPLB as a function of time (Fig 4a-4c). In all scenarios MPLB typically finds a solution of a given cost between two to three times faster than aFMT*. In the Corridors scenario (Fig. 3a) the convergence rate can be sped up by using an approximation factor (see suggestion for future work in Section VII). Interestingly, as we will show, the speedup obtained by MPLB is done while spending a smaller proportion of the time on LP compared to aFMT*. Next, we chose a slightly more difficult problem (Fig. 3d). Here, we measured the succes rate to find *any* solution.

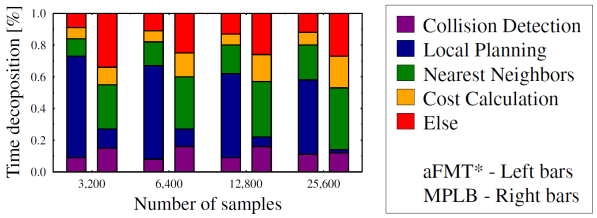


Fig. 5: Percentage of time spent for each of the main components in each iteration for both algorithms for the Grids Scenario. Each iteration is represented by the number of samples used. The left (right) bars of each iteration represent the result of aFMT* (MPLB, respectively). Note that the time of each iteration for each algorithm is different.

By that we test how MPLB behaves when all nodes are promising (i.e., when no solution has been obtained in the previous iterations). The results (Fig. 4d) show how the framework can also help in finding an initial solution much faster than aFMT*.

B. Nearest Neighbors and Local Planning calls

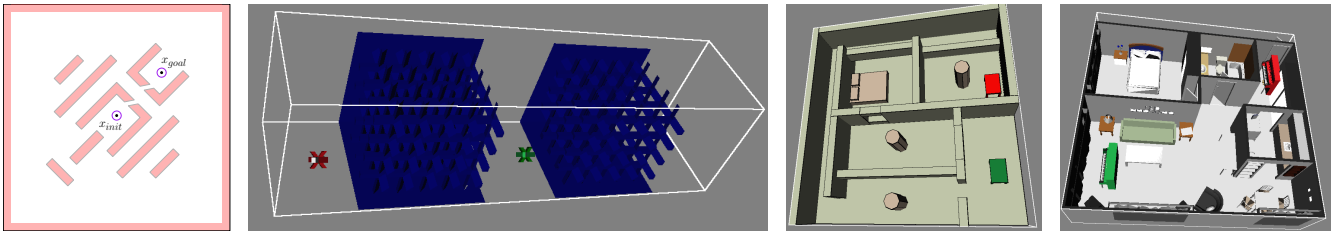
We profiled aFMT* and MPLB and collected the total time spent on CD for point sampling, LP for edges, NN calls and cost computations. Results for the Grids scenario are presented in Fig. 5 (similar behavior was observed for the other scenarios as well). Clearly, CD computation time (due to sampling, not LP) is negligible for both algorithms and cost calculation plays a larger (but still small) role for MPLB. CD calls due to LP calls are the main bottleneck for aFMT* (starting at around 65% and gradually decreasing to 45%). For MPLB such calls start as a main time consumer but as samples are added their percentage of the overall iteration time becomes quite small—around 2% for the last iteration. NN calls play an almost complementary role to the LP and for the last iteration take 40% of the total running time for the MPLB algorithm while taking less than 20% for aFMT*.

The table to the right reports on the ratio of the number of NN and LP calls performed by MPLB and aFMT* for the Grids scenario as a function of the number of samples n. For all values of n, the number of NN calls performed by MPLB is lower than those performed by aFMT*. As expected, MPLB performs less LP calls than aFMT*. Moreover, as n increases, the number of LP calls performed by MPLB decreases to a fraction of the number of calls performed by aFMT*.

n	the ratio $\frac{\#_{\text{NN}}(\text{MPLB})}{\#_{\text{NN}}(\text{aFMT*})}$	the ratio $\frac{\#_{\text{LP}}(\text{MPLB})}{\#_{\text{LP}}(\text{aFMT*})}$
1.6K	0.71	0.38
3.2K	0.53	0.31
6.4K	0.68	0.33
12.8K	0.68	0.19
25.6K	0.69	0.20
51.2K	0.99	0.05

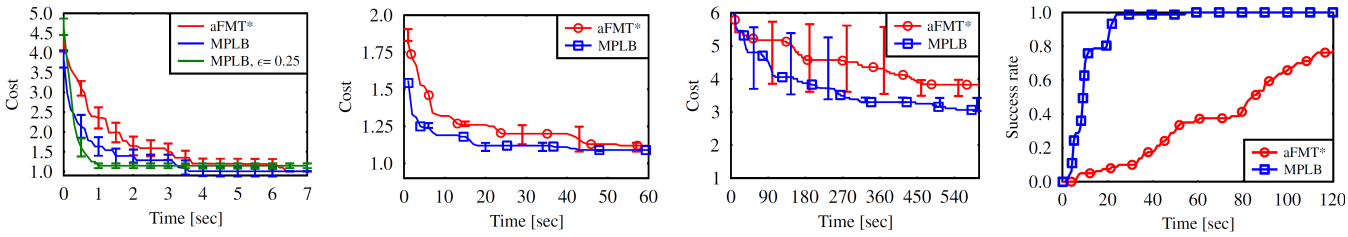
VII. Conclusion and Outlook

In this work we show that by using effective lower bounds and with no compromise on the cost of paths produced by the algorithm, the weight of CD (via LP calls) may become almost negligible when compared to the weight of NN. This follows the ideas presented by Bialkowski et al. [8] but uses more general methods. Looking into NN computation, one can notice that AO algorithms such as sPRM* [3], FMT* and



(a) Corridors (b) Grids (c) Home (d) Apartment

Fig. 3: Scenarios used for the evaluation. (a) Two dimensional setting for a point robot. A low-cost path is easy to find yet in order to find a high-quality path, the robot needs to pass through two narrow passages. (b) Three-dimensional C-space for a translating robot in space. To find the shortest path the robot needs to pass through a three-dimensional grid. (c) Six-dimensional C-space for a table-shaped robot translating and rotating in space. Finding a path is relatively easy yet much time is needed to converge to the optimal path. (d) Six-dimensional C-space for a piano-shaped robot. Start and target configurations for (b)-(d) are depicted by green and red robots, respectively, The Home and Apartment scenarios are provided by the OMPL [17] distribution.



(a) Corridors (b) Grids (c) Home (d) Apartment

Fig. 4: (a)-(c) Average cost vs. time. Cost values are normalized such that a cost of one represents the cost of an optimal path. Low and high error bars denote the twentieth and eightieth percentile, respectively. (d) Success rate to find a path for the Apartment scenario.

MPLB rely on a specific type of NN computation: given a set P of n points, either compute for each point p all its k nearest neighbors, or all neighbors within distance r from p. In both cases, P is known in advance and k (or r) are parameters that do not change throughout the algorithm or throughout a single iteration of the algorithm.

This calls for using application-specific NN algorithms and not general purpose ones. Indeed, in a different paper we show that using such a data structure allows to significantly speed up motion-planning algorithms [18].

A possible enhancement to MPLB is to relax the AO to ANO: AO motion-planning algorithms, from a certain stage of their execution, invest huge computational resources at only slightly improving the cost of the current best existing solution. Similar to [11] one can construct a variant where given an *approximation factor* ε, the cost of the solution obtained is within a factor of $1+\varepsilon$ from the solution that MPLB would obtain for the same set of samples. We expand on this idea in the extended version of our paper [16]. Preliminary results, presented in Fig. 4d show the potential benefit of this approach.

VIII. Acknowledgements

The authors are greatful to Marco Pavone and his co-workers for their advice and support regarding FMT*.

References

[1] L. Janson and M. Pavone, "Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions," *CoRR*, vol. abs/1306.3532, 2013.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, June 2005.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *I. J. Robotic Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[4] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Trans. Robot.*, vol. 12, no. 4, pp. 566–580, 1996.

[5] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[6] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *IROS*, 2011, pp. 2640–2645.

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IROS*, 2014, pp. 2997–3004.

[8] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning," in *WAFR*, 2012, pp. 365–380.

[9] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *I. J. Robotic Res.*, vol. 33, no. 1, pp. 18–47, 2014.

[10] Z. Littlefield, Y. Li, and K. E. Bekris, "Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics," in *IROS*, 2013, pp. 1779–1785.

[11] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality, motion planning," in *ICRA*, 2014, pp. 4680–4685.

[12] O. Salzman, D. Shaharabani, P. K. Agarwal, and D. Halperin, "Sparsification of motion-planning roadmaps by edge contraction," *I. J. Robotic Res.*, vol. 33, no. 14, pp. 1711–1725, 2014.

[13] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *ICRA*, 2013, pp. 2421–2428.

[14] W. Wang, D. Balkcom, and A. Chakrabarti, "A fast streaming spanner algorithm for incrementally constructing sparse roadmaps," *IROS*, pp. 1257–1263, 2013.

[15] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[16] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," *CoRR*, vol. abs/1403.7714, 2014.

[17] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robot. Automat. Mag.*, vol. 19, no. 4, pp. 72–82, 2012.

[18] M. Kleinbort, O. Salzman, and D. Halperin, "Efficient high-quality motion planning by fast all-pairs r-nearest-neighbors," in *ICRA*, 2015, these proceedings.