Fast Planning Over Roadmaps via Selective Densification

Brad Saund[1] and Dmitry Berenson[1]

***Abstract*—We propose the Selective Densification method for fast motion planning through configuration space. We create a sequence of roadmaps by iteratively adding configurations. We organize these roadmaps into layers and add edges between identical configurations between layers. We find a path using best-first search, guided by our proposed estimate of remaining planning time. This estimate prefers to expand nodes closer to the goal and nodes on sparser layers.**

We present proofs of the path quality and maximum depth of nodes expanded using our proposed graph and heuristic. We also present experiments comparing Selective Densification to bidirectional RRT-connect, as well as many graph search approaches. In difficult environments that require exploration on the dense layers we find Selective Densification finds solutions faster than all other approaches.

***Index Terms*—Motion and Path Planning**

I. Introduction

We examine the motion planning problem, the task of finding a valid path through continuous configuration space from a start to a goal. Probabilistically complete [1] and asymptotically optimal [2] algorithms are known, but to be practical algorithms must be fast. The challenge for a search algorithm is to explore regions that will likely lead to a path, while not performing excessive checking of configurations that do not lead to the goal.

A common planning approach is to precompute a (probabilistic) roadmap (PRM), which is a graph where vertices represent configurations and edges represent motions. Planning can then be reduced to connecting the query start and goal to the graph and finding a path through the graph. In many robotics problems the obstacles are not known a priori, thus the validity of edges must be checked online. The computation time of this approach is dominated by node expansions of a graph-search algorithm, and collision checks for edges. Lazy edge evaluation has proved effective in robotics, with algorithms like LazyPRM [3] and the generalization Lazy Shorted Path (LazySP) [4] that minimize the number of collision checks.

A challenge when precomputing a roadmap is choosing the appropriate density of nodes in configuration space. Too few nodes and edges can result in a roadmap with no solutions.

Manuscript received: September, 9, 2019; Revised December, 4, 2019; Accepted January, 19, 2020.

This paper was recommended for publication by Editor Nancy Amato upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF under grant IIS-1750489 and by Toyota Research Institute (TRI). This article solely reflects the opinions of its authors and not TRI or any other Toyota entity.

[1]Authors are with the Robotics Department, University of Michigan, Ann Arbor, MI, USA. {bsaund, dmitryb}@umich.edu

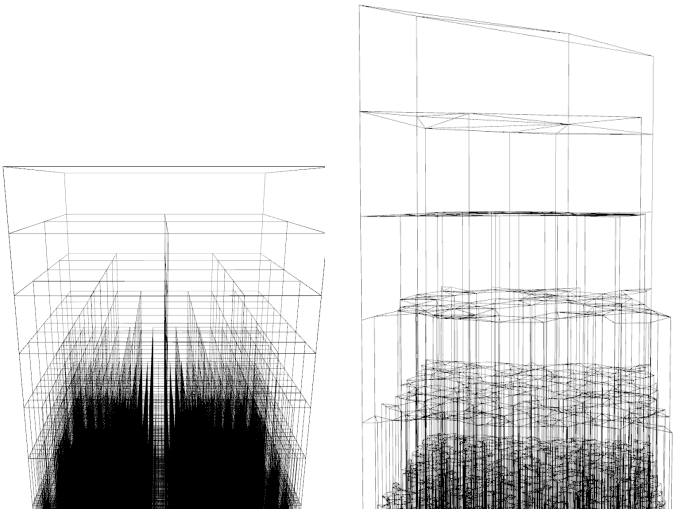

Fig. 1: Layered Graphs in a 2D C-space using a grid structure (left) and halton sequence (right)

Increasing the number of nodes in the roadmap increases the computation cost by increasing both the number of node expansions and number of collision-checks during a search. For many problems dense roadmaps are needed only in some regions while sparse roadmaps perform better in other regions with more free space. We desire a search method that is able to shift between graphs of different densities, achieving fast performance by searching sparser graphs in most regions and only densifying when required.

Our key contribution is the Selective Densification method for guiding the density of nodes and edges explored using a heuristic of remaining planning time. Specifically, this paper contributes:

- The Layered Graph: A sequence of roadmaps of increasing density combined into a single graph (Fig. 1)
- The Selective Densification Heuristic: An estimate of both remaining execution cost and planning cost used to guide best-first search
- Bidirectional search in LazySP with search direction during an iteration chosen by minimum planning time

We present proofs of the solution quality and maximum depth searched. To evaluate our method in practice, we perform motion planning experiments on a robotic manipulator arm in simulation and benchmark against existing methods. We find that in environments where paths exist on low-density graphs Selective Densification performs similarly to common

methods. In the most challenging environment tested, where paths exist only on high-density graphs, we find Selective Densification performs 4x faster than the next best approach.

II. Related Work

A. Sampling-based C-space planning

In many robotics domains a graph is constructed to perform search over a continuous space. Sampling-based motion planning consists of building a graph with an embedding into C-space and then searching the graph for a valid (collision-free) path from a start to a goal.

While approaches such as RRT [1] build a tree online, other approaches such as PRM [5] can precompute a roadmap and perform online collision checking of edges. Both RRT and PRM seek feasible solutions in continuous space without explicit regard to path quality. Early asymptotically-optimal planners such as RRT* [2] find minimum cost solutions in the limit as $t \to \infty$, with improvements such as BIT* [6] seeking faster convergence. Interestingly, in practice, repeated RRT with shortcut smoothing tends to outperform RRT* and variants both in time and path quality [7] [8]. Given these results, our approach focuses on finding solutions quickly and relies on the shortcut smoothing post-processing to reduce cost.

A* [9] can be used to search roadmaps, however the edge collision-check is typically the most expensive operation in robotics applications, thus algorithms such as LazyPRM [3] and the later generalization LazySP [4] that minimize the number of edge evaluations typically run faster. In fact, LazySP is optimal w.r.t. the number of edge checks [10] and later work balances time spent on edge checking vs. expansion [11]. Using a precomputed roadmap offers the advantage that in static scenes edge validity is constant, so at most one collision check is required across multiple queries. However even in a single query setting using a precomputed roadmap offers two distinct advantages over a RRT: determinism and the ability to precompute environment-independent properties of edges. We employ both forms by generating a graph in the robot configuration space and precomputing the swept volumes occupied by the robot for some edges. This is done without knowledge of the specific environment and prior to query thus we consider this precomputation not time sensitive.

B. Densification Strategies

A challenge when constructing roadmaps is determining the number of vertices and edges needed. Selecting too few may yield a roadmap without a feasible solution or only a costly solution. Selecting too many yields a roadmap that is computationally expensive to store and search. Approaches such as SPARS [12], bridge sampling [13], and others therefore build explicit graphs online using specific vertex sampling and edge-connection methods that are aware of obstacles to limit the graph size. We pre-compute a much larger graph than these methods generate, but bias search towards the sparser portions.

Other approaches use multi-resolution roadmaps, connecting a low resolution roadmap to high resolution roadmaps in certain regions specified by heuristics [14] [15]. Planning with Adaptive Dimensionality uses a user-specified projection into a lower dimensional space, and reverts to the full dimensional space where the projection is inadequate [16] [17] [18]. For example, a projection may map a car to a 2D point or a robotic arm to the end effector location. This approach will only construct two graphs of different densities and relies on a user-defined projection that must obey certain properties.

Yet another approach is batching, where a fixed roadmap is searched completely before densifying [19], [20]. This has been used to first find solutions quickly on sparse layers then find better [21], and asymptotically optimal [22] solutions by searching denser layers. A drawback is the expense of the search of an entire layer before densifying, which can be especially large if no path exists. As suggested in the future work of LEMUR [23] it would be desirable to further densify promising regions before exploring the entire batch. Our work pursues this approach of selectively densifying certain regions by balancing the expected path length and computational cost at different batch levels.

C. Heuristic Graph Search

Finding a path in a roadmap requires searching a graph. While A* finds optimal paths using an admissible heuristic [9], weighted A* inflates the heuristic [24] achieving bounded sub-optimality (both with [25] and without [26] using a closed list). Methods such a ARA* [27], ANA* [28] and numerous others observe that an inflated heuristic and/or a greedy search tend to find solutions faster, trading path cost for planning time, although there is no such guarantee and in some cases this can slow down search [29]. A^*_ϵ [24], BUGSY [30] and LEMUR [23] make this tradeoff explicitly by including various estimates of "planning-cost-to-go" in the search heuristic. We extend this idea to our Layered Graph, on each layer increasing the estimated future planning cost, and therefore inflation factor.

III. Problem Definition and Notation

Let $\mathcal{C}$ be a configuration space with free space $\mathcal{C}_{free}$ and C-space obstacles $\mathcal{C}_{obs} = \mathcal{C} \setminus \mathcal{C}_{free}$. $\mathcal{C}_{obs}$ are represented implicitly via a collision-check function that given a $q \in \mathcal{C}$ tests whether $q \in \mathcal{C}_{free}$. A path $\xi_{s\to g}$ from q_s to q_g is feasible if $\xi_{s\to g} \subset \mathcal{C}_{free}$ In practice we relax feasibility by assuming it is sufficient to collision-check a densely discretized $\xi_{s\to g}$.

The execution cost of the straight-line motion between configurations is given by $c_x : \mathcal{C} \times \mathcal{C} \to \mathbb{R}_{\geq 0}$, which for this work we assume is given by a distance: $c_x(q_1, q_2) = ||q_1 - q_2||$. The execution cost of a feasible discretized path $\xi_{s\to g} = [q_1, ..., q_n]$ is given by

$$c_x(\xi_{s\to g}) = \sum_{i=1}^{n-1} c_x(q_i, q_{i+1}) \tag{1}$$

Furthermore, a planner incurs a cost c_p in computing a feasible path ξ. We consider the case where c_p is the time spent to return ξ once q_s, q_g and $\mathcal{C}_{obs}$ are provided. As the primary objective we seek planners that find a feasible path as quickly as possible.

Define a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E}$. A roadmap is an embedding of $\mathcal{G}$ into $\mathcal{C}$ such that vertices map to configurations $q \in \mathcal{C}$ and edges map to C-space paths $\xi_e \subset \mathcal{C}$ connecting vertices. Let an r-disk graph be a graph with edges connecting exactly those vertices v_1, v_2 for which the corresponding configuration q_1, q_2 satisfy $||q_1 - q_2|| < r$.

An edge e is collision-free if and only if the corresponding path $\xi_e \subset \mathcal{C}_{free}$. As such, a feasible path through a graph from v_s to v_g consists of collision-free edges and induces a feasible C-space path ξ from q_s to q_g. Each edge has an associated cost, and the cost of a full path is the sum of the edge costs.

For use in lazy edge evaluation, let $\hat{\mathcal{G}}$ denote the graph $\mathcal{G}$ initialized with the collision state of all edges labeled as "unknown".

IV. Approach: Selective Densification

Our approach consists of finding a path on a graph with an embedding into C-space. The core innovation is the combination of specific graph structure with a heuristically guided search that balances execution cost and expected remaining planning cost. The graph has connected layers of different densities allowing the search to traverse to denser layers to navigate through tight spaces with precision and then return to sparser layers that are searched more quickly. The graph is searched using A* with an inadmissible heuristic estimating both path cost and planning cost. As the true remaining planning cost is unknown, it is estimated via a heuristic that grows as the layer depth increases.

A. Graph Structure

Consider a sequence of unique configurations $Q = (q_1, q_2, \dots)$, and a strictly increasing sequence of positive integers $(n_1 < n_2 < ...n_D)$. A layer L_i is an r-disk graph constructed from the first n_i configurations of Q and connection radius r_i. Denote a vertex of L_i with v_j^i where $j \leq n_i$. For each pair of adjacent layers (L_i, L_{i+1}) define the inter-layer edges as $\mathcal{E}_{i \leftrightarrow i+1} = \{e(v_j^i \leftrightarrow v_j^{i+1}) \ \forall j \leq n_i\}$. Note that v_j^i and v_j^{i+1} represent the same configuration q_j, thus the inter-layer edges are zero-cost edges that change layers without changing configurations. The Layered Graph is defined as $\mathcal{G} = (\cup_i L_i) \cup (\cup_i \mathcal{E}_{i \leftrightarrow i+1})$.

We define $v.q$ as the configuration associated with vertex v and $v.d$ as the layer number i where $v \in L_i$

Figure 1 visualizes two Layered Graphs in a 2D C-space, with the vertical dimension representing layer depth. Vertical edges are therefore the zero-cost edges connecting vertices in adjacent layers representing the same configuration.

Although this graph can be precomputed, a query may consist of a q_s and q_g that are not in $\mathcal{G}$, thus during a query vertices for q_s and q_g are added to each layer, edges within each L_i are added determined by the connection radius r_i, and inter-layer edges are added for the vertices corresponding to q_s and q_g.

B. Utility Guided Graph Search

To answer a query (q_s, q_g) first the corresponding graph nodes (v_s, v_g) are found (or added), a best-first (A*) search with lazy edge evaluation is performed over the graph yielding a path through $\mathcal{G}$, and the induced C-space path is returned. Lazy edge evaluation attempts to reduce the number of edge collision-checks, but may require many calls to A*. This approach has been called LazyPRM [3] or LazySP [4].

Algorithm 1 Selective Densification Search

```
1: while True do
2:     ξ ← A*(Ĝ, v_s, v_g)
3:     if CheckEdges(ξ) then
4:         return ξ
```

Algorithm 2 Bidirectional Selective Densification Search

```
1: t_forward ← 0
2: t_backward ← 0
3: while True do
4:     if t_forward ≤ t_backward then
5:         ξ ← A*(Ĝ, v_s, v_g)
6:         t_forward ← t_forward + timeOf(Line 5)
7:     else
8:         ξ ← A*(Ĝ, v_g, v_s).reverse()
9:         t_backward ← t_backward + timeOf(Line 8)
10:    if CheckEdges(ξ) then
11:        return ξ
```

Algorithms {1 or 2} and 3 show the outer loops that perform the lazy edge evaluation. Line 2 of Algorithm 1 performs a best-first search over $\hat{\mathcal{G}}$, which is graph $\mathcal{G}$ with the optimistic assumption that unevaluated edges are collision free. Edges along this optimistic path are collision-checked in order by Algorithm 3. The result of a collision check is stored so that future iterations of A* may not traverse invalid edges.

Algorithm 2 implements bidirectional LazySP as bidirectional search tends to find solutions faster than unidirectional variants. Bidirectional LazySP alternates a unidirectional A* search at each iteration reusing the results of collision checks and does not require the algorithmic machinery of bidirectional variants of A* to maintain guarantees. A common implementation of bidirectional search alternates search direction at each iterations. We introduce this variant that balances the total time spent searching in each direction. If the search time per iteration is independent of direction, our version is equivalent to alternating direction after each iteration. However in practice we observe the A* time is not negligible and can have a huge dependence on direction. In such cases we empirically observe significantly better performance when balancing total time instead of iterations.

Algorithm 3 CheckEdges(ξ)

```
1: for e in ξ do
2:     if e is not valid then                ▷ collision check
3:         mark e as invalid
4:         return False
5:     mark edge as valid
6: return True
```

Algorithm 4 A* $(\hat{\mathcal{G}}, v_s, v_g)$

1: $open \leftarrow \{v_s\}, closed \leftarrow \{\}, \hat{g}(v) \leftarrow \infty, \hat{g}(v_s) = 0$
2: **while** $open$ is not empty **do**
3: $v \leftarrow open$.pop_lowest_$f$value()
4: insert v into $closed$
5: **if** v is v_g **then**
6: **return** ReconstructPath(v_s, v_g)
7: **for** edge e and successor v' of v **do**
8: **if** e is $invalid$ **then** skip
9: **if** $\hat{g}(v') > \hat{g}(v) + c_x(e)$ **then**
10: $\hat{g}(v') = \hat{g}(v) + c_x(e)$
11: **if** $v' \notin closed$ **then**
12: $\hat{f}(v') = \hat{g}(v') + \hat{h}_{SD}(v')$
13: insert v' into $open$
14: **return** No Path Exists

Algorithm 4 is A* with invalid edges removed, and using the inadmissible heuristic $\hat{h}_{SD}$. The functions $g^*(v), h^*(v)$, and $f^*(v)$ for a node are the optimal cost-to-come, cost-to-go, and cost from the start to goal through v respectively. $\hat{g}, \hat{h}_{SD}$, and $\hat{f}$ are estimates of these quantities.

C. Heuristics

The heuristic $\hat{h}_{SD}$ contains both a heuristic estimate of the execution cost-to-go ($\hat{h}_x$) and the planning time-to-go ($\hat{h}_p$). We choose $\hat{h}_x$ as the euclidean distance to the goal. $\hat{h}_x$ is consistent (and therefore admissible) and the cost is only achieved if there is a straight line path from v to the goal. $\hat{h}_p$ is proportional to distance $||v.q - q_g||$, the number of nodes in the layer $n_{v.d}$, and a tunable constant w_t.

$$\hat{h}_x(v) = ||v.q - q_g|| \tag{2}$$
$$\hat{h}_p(v) = w_t n_{v.d} ||v.q - q_g|| \tag{3}$$
$$\hat{h}_{SD}(v) = \hat{h}_x(v) + \hat{h}_p(v) \tag{4}$$
$$= \hat{h}_x(v)(1 + w_t n_{v.d}) \tag{5}$$

We now discuss our choice of $\hat{h}_p$, with formal analysis provided in the next section. First consider that $\hat{h}_p$ is proportional to $||v.q - q_g||$. Assuming a maximum edge length, doubling the distance to the goal will double the lower bound on the number of node expansions required (approximately, due to the discretization of nodes). Mathematically, proportionality to $||v.q - q_g||$ causes $\hat{h}_p$ to appear as an inflation of $\hat{h}_x$, which has empirically shown to reduce planning times [24] [29].

Next, we discuss $\hat{h}_p(v) \propto n_{v.d}$. Consider search over a traditional roadmap composed of a single layer. We expect search time to increase for a denser roadmap If an oracle guides a search to only expand nodes on the optimal path, we might expect search time to be inversely proportional to the maximal edge. However, consider the case of misleading cul-de-sacs, as shown in Fig. 3, where a best-first search must expand nodes in the entire volume of the cul-de-sac, thus the total time is proportional to the total number of nodes. We intentionally tailor $\hat{h}_p$ to these environments.

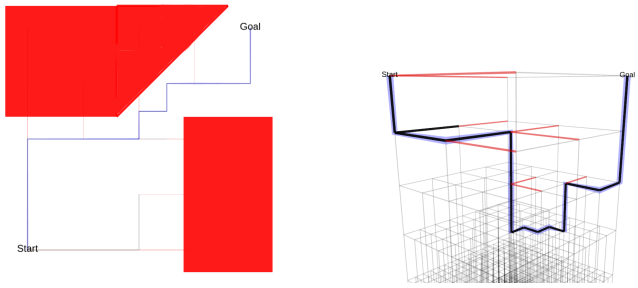

Fig. 2: A 2D search problem solved using Selective Densification with evaluated edges shown in black (valid) and red (invalid) and the final path shown in blue. Left: 2D view with red obstacles. Right: View of Layered Graph with unevaluated edges shown in light grey

Finally, consider the constant w_t. Theoretically w_t can appear as a weighting between a user's preference between planning time and execution time. Clearly if a user is agnostic about planning time w_t should be set to 0, as then $\hat{h}_{SD}$ reduces to $\hat{h}_x$, and thus A* will yield the optimal path on $\hat{\mathcal{G}}$. On the other hand if the goal is simply to find a feasible path as quickly as possible, setting w_t very high yields a greedy search over each layer, which empirically finds solutions quickly [28], [29]. However, the optimal w_t might also depend on factors such as the expected path length, past planning experience, expected size of cul-de-sacs, and expected percentage of freespace in $\mathcal{C}$. In practice we observe that regardless of the value of w_t, shortcut smoothing tends to yield paths with similar cost.

Fig. 2 illustrates (unidirectional) Selective Densification search applied to a toy example. Because $\hat{h}_p \propto n_{v.d}$, A* tends to explore nodes in sparser layers first. The search first explores the sparsest edges in the top layer but neither is valid. The search progresses to nodes on denser layers closest to the goal, and then pops back up after navigating the narrow region. Although the graph extends deep, these edges are never explored, due to the higher planning heuristic of nodes on denser layers.

As an aside, suppose Selective Densification finds a path well before a robot can begin execution. An anytime method that converges to the optimal path could be created through the following modifications. Store the cost of the best path found so far $c_x(\xi^{Best})$ after Algorithm 2. Add the check $\hat{g}(v') + \hat{h}_x(v') < c_x(\xi^{Best})$ to Algorithm 4 Line 11 to prevent expansion of nodes that could not possibly lead to better paths. Repeat Algorithm 2 until the user requests a result, or until no nodes are on the open list. One could imagine a variety of schemes to decrease w_t, but choosing a particular scheme is beyond the scope of this work.

V. Analysis

In this section we prove bounds on the solution cost and search depth of Selective Densification. Because $\mathcal{G}$ contains nodes with different estimated planning times connected by zero-cost (vertical) edges, $\hat{h}_{SD}$ is not simply an inflated

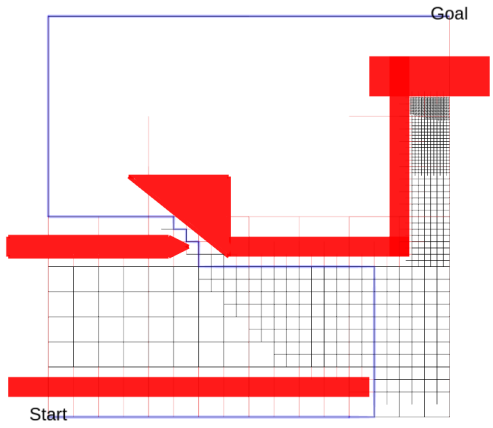

Fig. 3: All edges explored using Algorithm 4 (unidirectional) search with a misleading trap.

consistent heuristic, however we build up similar guarantees relying on consistency across each layer of $\mathcal{G}$.

For this section define g_i^* as g^* over the single-Layered Graph L_i. References to A* use the proposed heuristic Eq. (5), which is written as a per-layer inflation factor $\hat{h}_{SD}(v) = \epsilon_i \hat{h}_x(v)$ with $\epsilon_i = (1 + w_t n_i)$.

First, consider the A* search over a single layer ($\mathcal{G} = L_i$).

Theorem V.1. *Any v expanded by A* (L_i, v_s, v_g) satisfies $\hat{g}(v) \leq \epsilon_i g^*(v)$.*

Proof. On a single layer A* reduces to weighted-A* with $\hat{h}_{SD}$ equivalent to the consistent heuristic $\hat{h}_x$ inflated by ϵ_i, thus we can directly apply the proof presented in [25], Theorem 10. □

Next we show this per-layer bound holds on the full graph.

Theorem V.2. *Consider an optimal path $\xi_i = (v_s^i = v_1, ..., v_k = v_g^i)$ on L_i. For every $v_j \in \xi_i$ expanded by A* over $\hat{\mathcal{G}}$ it holds $\hat{g}(v_j) \leq \epsilon_i g_i^*(v_j)$*

Proof. Each iteration of A* expands the node with the lowest $\hat{f}$, thus it is sufficient to show that if a node $v_z \in \xi_i$ is on the open list with $\hat{g}(v_z) > \epsilon_i g_i^*(v_z)$ then there must be another node on the open list with lower $\hat{f}$-value. We show this by induction, by showing that if at some iteration Theorem V.2 holds for all $v \in \xi_i$ expanded so far implies it holds for the next iteration. The base case trivially holds, as initially no nodes have been expanded.

Consider some iteration of A* and assume Theorem V.2 holds for all $v \in \xi_i$ expanded so far. Consider some node $v_z \in \xi_i$ with $\hat{g}(v_z) > \epsilon_i g_i^*(v_z)$, thus with $\hat{f}(v_z) > \epsilon_i(g_i^*(v_z) + \hat{h}_x(v_z))$. Note that if no such v_z exists, Theorem V.2 trivially holds for the next iteration. We show that v_z will not be chosen for expansion.

Case 1: A node before v_z along ξ_i has been expanded. There then must be a $v_{m-1} \in \xi_i$ before v_z along ξ_i with successor $v_m \in \xi_i$ on the open list. By assumption $\hat{g}(v_{m-1}) \leq \epsilon_i g_i^*(v_{m-1})$. It follows that $\hat{g}(v_m) \leq \epsilon_i g_i^*(v_m)$.

$$\begin{aligned} \hat{g}(v_m) &\leq \hat{g}(v_{m-1}) + c_x(v_{m-1}, v_m) && (6)\\ &\leq \epsilon_i g_i^*(v_{m-1}) + c_x(v_{m-1}, v_m) && (7)\\ &\leq \epsilon_i g_i^*(v_m) && (8)\end{aligned}$$

We show v_m has a lower $\hat{f}$-value than v_z so the assumption will also hold in the next iteration.

For every node along ξ_i it holds that

$$\begin{aligned} g_i^*(v_j) + \hat{h}_x(v_j) &\leq g_i^*(v_j) + \hat{h}_x(v_{j+1}) + c_x(v_j, v_{j+1}) && (9)\\ &= g_i^*(v_{j+1}) + \hat{h}_x(v_{j+1}) && (10)\end{aligned}$$

Eq. (9) is obtained using the consistency of $\hat{h}_x$ over L_i. Eq. (10) uses $g_i^*(v_m) + c_x(v_m, v_{m+1}) = g_i^*(v_{m+1})$ along the optimal path.

This is then used to related the $\hat{f}$-values of v_m and v_z:

$$\begin{aligned} \hat{f}(v_m) &= \hat{g}(v_m) + \epsilon_i \hat{h}_x(v_m) && (11)\\ &\leq \epsilon_i(g_i^*(v_m) + \hat{h}_x(v_m)) && (12)\\ &\leq \epsilon_i(g_i^*(v_z) + \hat{h}_x(v_z)) && (13)\\ &< \hat{f}(v_z) && (14)\end{aligned}$$

Thus v_z will not be expanded in this iteration.

Case 2: No node before v_z along ξ_i has been expanded yet. Thus there must be some $v_s^{i'}$ with $i' \leq i$ on the open list. That is, the open list will contain either the start node, or a node on a denser layer representing the same configuration as the start node. It is straightforward to see $\hat{f}(v_s^{i'}) < \hat{f}(v_s^i) \forall i' \leq i$. Applying the same analysis from Case 1: $\hat{f}(v_s^{i'}) < \hat{f}(v_z)$ □

Corollary 1. *Any $v^i \in L_i$ expanded by A* on $\mathcal{G}$ satisfies $\hat{g}(v) \leq \epsilon_i g_i^*(v)$.*

Proof. Theorem V.2 directly proves this by setting v^i as the end of ξ_i. If there is no feasible ξ_i then $g_i^*(v) = \infty$ and this trivially holds. □

As this bound on $\hat{g}$ holds for any node, it must hold for the goal node v_g. This bound on $\hat{g}$ therefore also applies to the path returned by A*.

Corollary 2. *Consider an optimal path ξ_i^* on a L_i. Selective Densification over $\mathcal{G}$ returns a path ξ_{SD} with execution cost $c_x(\xi_{SD}) \leq \epsilon_i \cdot c_x(\xi_i^*) \forall i$ with $\epsilon_i = 1 + w_t n_i$.*

Proof. Selective Densification consists of a LazySP-style search (Algorithm 2) using A*. A* is performed over $\hat{\mathcal{G}}$, the copy of $\mathcal{G}$ that is optimistic about unevaluated edges. Due to this optimism, ξ_i^* will always be valid on $\hat{\mathcal{G}}$.

A* returns $\hat{\xi}$, a path over $\hat{\mathcal{G}}$ with edges that may not have been collision checked yet. On the final iteration Selective Densification validates all edges in $\hat{\mathcal{G}}$, returning ξ_{SD}.

$$c_x(\xi_{SD}) = c_x(\hat{\xi}) = \hat{g}(v_g) \leq \epsilon_i g_i^*(v_g) \leq \epsilon_i c_x(\xi_i^*) \quad (15)$$

□

This demonstrates that adding additional layers to $\mathcal{G}$ can only improve the bound of the execution cost of the path

returned by A*. In particular the larger inflation factor of the dense layers does not worsen the bound from the lower inflation factor of a sparse layer.

Corollary 3. *With $w_t = 0$ Selective Densification returns the optimal path.*

Proof. $w_t = 0 \implies \epsilon = 1 \implies c_x(\xi) = c_x(\xi^*)$ □

Next we show that when a path exists the $\hat{h}_{SD}$ heuristic bounds the deepest layer of a search. We do this by showing that for a sufficiently deep layer $\hat{h}_{SD}$ will be larger than any $\hat{f}$ on the open list. To make this claim, we first define clearance. A path ξ has clearance δ if and only if for every point p on ξ the configuration-space ball of radius δ centered at p contains only valid configurations. We assume there exists a path with clearance δ.

Theorem V.3. *Consider a Layered Graph $\hat{\mathcal{G}}$ that contains a feasible solution ξ_i with clearance δ on L_i. The deepest node expanded by A* is at most on L_j for a j with $\epsilon_j \geq \epsilon_i \cdot c_x(\xi_i)/\delta$.*

Proof. From Theorem V.2

$$\epsilon_i \cdot c_x(\xi_i) \geq \epsilon_i f_i^*(v) \geq \hat{f}(v) \forall v \in \xi_i \tag{16}$$

Define v_k as the first node expanded on layer L_{j+1}.

Consider two cases:

If $||v_k - v_g|| \geq \delta$ then

$$\hat{f}(v_k) \geq \hat{h}_{SD}(v_k) \tag{17}$$
$$\geq \epsilon_{j+1}||v_k - v_g|| \tag{18}$$
$$\geq \epsilon_{j+1}\delta \tag{19}$$
$$> \epsilon_j \delta \tag{20}$$

Consider a layer j sufficiently deeper than layer i such that $\epsilon_j \geq \epsilon_i \cdot c_x(\xi_i)/\delta$. Then Eq. (20) yields $\hat{f}(v_k) \geq \epsilon_i \cdot c_x(\xi_i)$. Thus by Eq. (16) all nodes in ξ_i would be expanded before any node on layer L_{j+1}.

Otherwise consider $||v_k - v_g|| < \delta$. Since v_k is the first node expanded on layer L_{j+1} the predecessor to v_k therefore must represent the same configuration on layer L_j. However, with $\delta < r_j$ then v_g is on the open list and will be expanded before v_k. Since v_g has clearance δ the edge would be valid. Selective Densification would then terminate without ever expanding v_k. □

Theorem V.3 provides insight into potential traps for Selective Densification, as although the number of layers searched is bounded, this bound may be large. In practice this can occur when nodes are expanded in a cul-de-sac close to the goal, causing the heuristic to be a small fraction of $\hat{f}$, and therefore misleading Selective Densification to explore on a dense layer. In such a scenario $\hat{h}_p$ vastly underestimates the remaining planning time of dense layers. Fig. 3 illustrates the behavior in such an environment (using unidirectional search).

VI. Experiments

Experiments were performed comparing Selective Densification with many methods of searching the same graph, and also with numerous algorithms from OMPL [31] on three scenarios for a 7-DOF robotics arm. Each strategy was tested 20 times in each scenario with the same start and goal configurations using different random seeds creating different graphs. We considered the cost of a path to be its length in C-space.

We implemented Selective Densification in C++. Robot-environment collision checking used GPU-Voxels [32] with a voxel grid of 200x200x200 and a voxel size of 2cm^3. Collision checks were performed by intersecting the voxel grid representing the robot with the voxel grid representing the environment obstacles. A NVidia 1080Ti GPU and an Intel i7-7700K CPU was used for all experiments, and a single collision check took approximately 1ms.

Each Layered Graph was constructed explicitly, using Q as a 7D halton sequence with a pseudo-random offset. Layer i contained $n_i = 2^i$ nodes, with connection radius r_i set so the expected number of edges per node was 30, consistent with previous halton roadmap experiments [4]. We used 18 layers as empirically this fit comfortably in memory and was able to solve our scenarios. Each graph was therefore constructed with 524287 nodes and ∼7 million edges, taking ∼2 minutes. Collision checking was performed by discretely checking states along an edge at most 0.02 radians apart.

We examine Selective Densification with and without precomputation, where the swept volume of relevant edges has been precomputed (SD-pre). Note that unlike the more common reuse of PRMs by storing edge validity for particular obstacles, this method is agnostic to changes in the environment, and only requires that the robot remain the same. During online planning a collision check can be performed quickly via a lookup of the swept volume followed by an intersection with the environment. For the results shown we precomputed all edges that were checked, except from v_s and v_g, as these were not added until query time. Checking a precomputed edge took ∼1ms, while without precomputation checking took between ∼1ms (the first configuration checked was invalid) and ∼150ms. Note that it is infeasible to store the swept volumes for all 7 million $\mathcal{E} \in \mathcal{G}$. In these experiments we stored all edges checked by all previous planning runs thus this provides an optimistic assessment of the practical improvement that precomputed swept volumes can offer.

For each scenario we compared Selective Densification to many common methods of graph search used in robotics. In all cases we use lazy collision checking, as direct collision checking takes excessively long We attempted search using A* with the admissible heuristic of C-space distance from Equation (2), as well as the inflated, and greedy heuristic. We compare against Iterative Deepening (ID), or batching, where a search continues on a single layer of $\mathcal{G}$ until that layer is shown to have no solution. We also compare against bidirectional RRT-connect, PRM, SPARS, STRIDE, BIT*, and LBKPIECE from OMPL, although all SPARS attempts exceeded our 5 minute limit so results are not presented. These approaches have many variants able to improve solutions over time (e.g. ANA*, RRT*, etc.), but we present results using shortcut smoothing as we found it vastly outperformed other methods on these problems. Since shortcut smoothing no longer constrains the path to $\mathcal{G}$, methods can outperform A*.

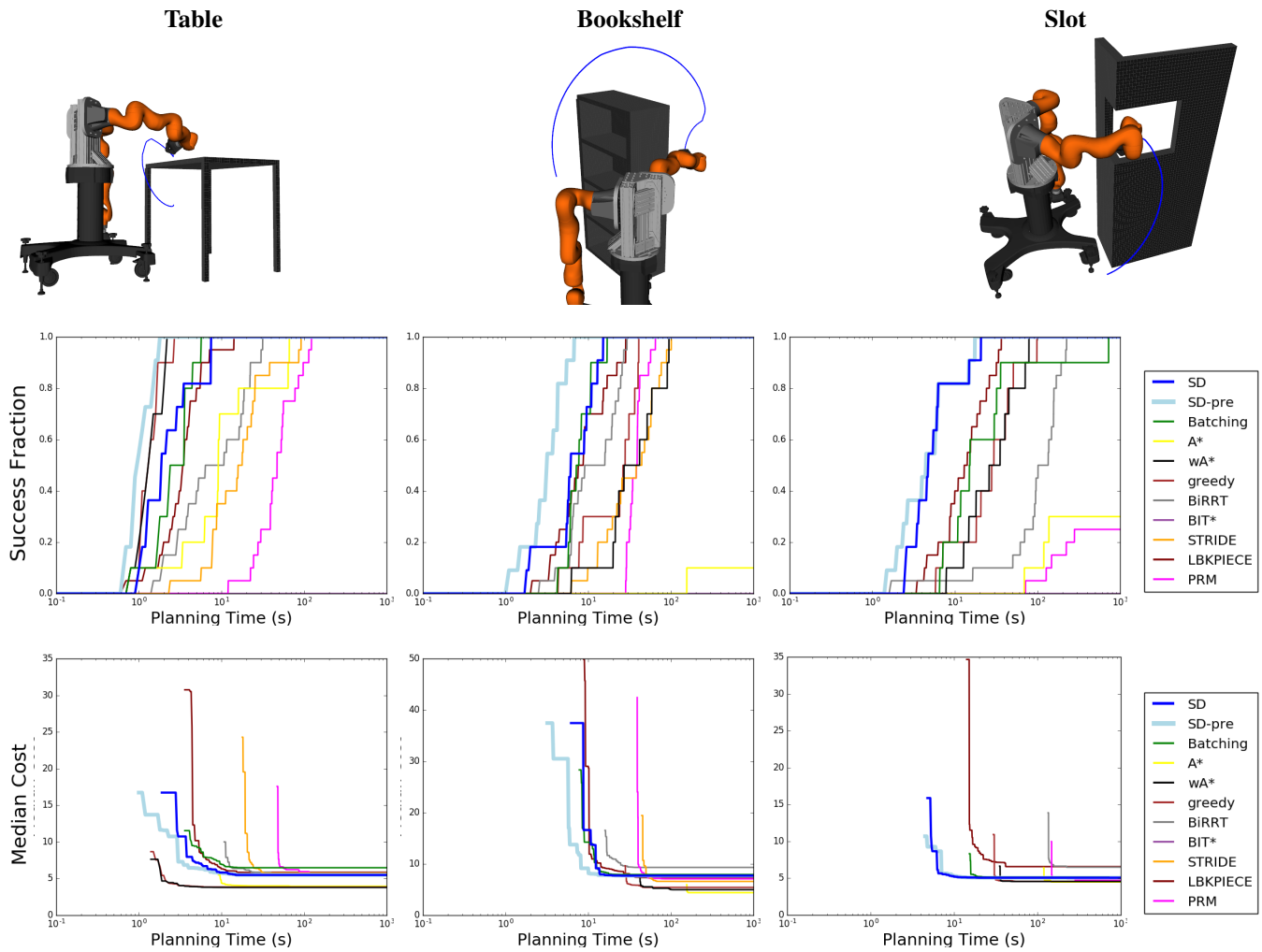

Fig. 4: Planning Comparisons. Top: The robot at the goal with a blue line tracing the end effector path from the start. Middle: Fraction of trials that have achieved any solution, as a function of time. Bottom: Median length of path found.

Results are shown in Fig. 4. The deepest layer on which a collision check was performed using SD was 8 (Table), 9 (Bookshelf), and 15 (Slot).

We find our proposed method (SD and SD-pre) finds a feasible solution to our scenarios in a time comparable to (Table) or faster than (Slot) all other algorithms tested. The path length of the initial solution found by SD (and other algorithms) was substantially suboptimal, however a few seconds of shortcut smoothing dramatically reduced the path length in all trials for all methods. This suggests that in practice although SD finds paths far better than the bound provided in Theorem V.2, the solution found is far from optimal and will be sensitive to the quality of the smoother.

In Fig. 4 we set w_t (which governs the inflation of each layer of SD) to 1.0, causing a near-greedy search on deeper layers. We performed a sweep over w_t for a single $\mathcal{G}$, with results for the Table Scenario shown in Fig. 5. As expected, we found that smaller w_t result in slower search with shorter paths after the graph search.

We find that the largest component of planning time is the collision checking of edges, as expected. However, we find repeated A* search time is significant. As this LazySP approach repeatedly searches similar graphs, we attempted to reuse information from previous A* iterations using generalized LPA* [33] but found this approach slower.

Overall we observed precomputing the swept volume of edges significantly reduces collision checking time of valid edges, however only a modest decrease is seen when checking invalid edges, as we observe in our Scenarios that edges are invalidated typically after checking only a few configurations. Overall we observed precomputation yields a modest improvement in overall performance.

Finally, we compare a baseline bidirectional LazySP search to our proposal of balancing the time searching each direction (Algorithm 2). In the Table and Bookshelf Scenarios both approaches perform similarly. In the Slot Scenario the forward search expands far more nodes and edges than the reverse search. By balancing search time our proposed approach performs fewer forward search iterations before finding a solution using the reverse search (Table I).

VII. Conclusions and Future Work

We proposed and implemented Selective Densification, a motion planning method that searches a graph composed of

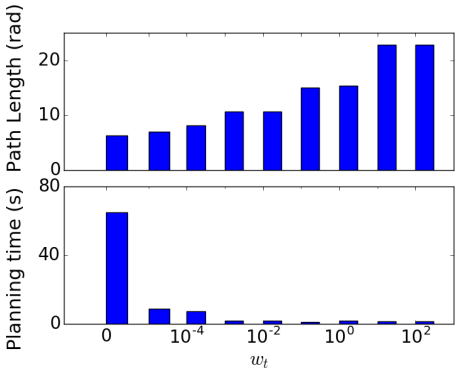

Fig. 5: Comparing Path Length (before smoothing) and planning time on the Table Scenario using Selective Densification with varying w_t.

	Total	Edge Check	Forward	Reverse
Proposed	6.5 ± 5.7	3.1 ± 2.1	1.8 ± 2.0	1.4 ± 1.5
Baseline	73.4 ± 79.3	9.3 ± 4.4	63.0 ± 75.1	0.9 ± 0.8

TABLE I: Planning times in seconds for bidirectional search comparing the Proposed (Algorithm 2) and Baseline (alternating each iteration) using SD on the Slot Scenario

layers of different densities of nodes. SD prioritizes search that is close to the goal and on sparse layers through the planning cost-to-go heuristic $\hat{h}_{SD}$. We presented proofs of path quality and limited search depth and performed planning experiments for a robotic arm demonstrating a speed up when planning in environments requiring dense graphs.

Unexplored in this work is the best method to set w_t, the weighting parameter for $\hat{h}_{SD}$. We showed a larger w_t tended to lower planning times but increase execution cost of the path found by the graph search. However, post-processing via shortcut smoothing drastically reduced the path cost suggesting only a marginal benefit might be gained from setting a low w_t.

REFERENCES

[1] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *ICRA*, 2000.

[2] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, 2011.

[3] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *ICRA*, 2000.

[4] C. Dellin and S. Srinivasa, "A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors," *ICAPS*, 2016.

[5] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 1996.

[6] J. Gammell, S. Srinivasa, and T. Barfoot, "Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," *ICRA*, 2015.

[7] J. Luo and K. Hauser, "An empirical study of optimal motion planning," in *IROS*, 2014.

[8] J. Meijer, Qujiang Lei, and M. Wisse, "An empirical study of single-query motion planning for grasp execution," in *IEEE AIM*, 2017.

[9] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[10] N. Haghtalab, S. Mackenzie, A. Procaccia, O. Salzman, and S. Srinivasa, "The provable virtue of laziness in motion planning," *ICAPS*, 2018.

[11] A. Mandalika, O. Salzman, and S. Srinivasa, "Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges," *ICAPS*, 2018.

[12] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," in *IJRR*, 2014.

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

[14] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *IJRR*, 2009.

[15] J. Petereit, T. Emter, and C. Frey, "Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality." IFAC, 2013.

[16] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path Planning with Adaptive Dimensionality," *SoCS*, 2011.

[17] B. Cohen, G. Subramanian, S. Chitta, and M. Likhachev, "Planning for Manipulation with Adaptive Motion Primitives," *ICRA*, 2011.

[18] K. Gochev, A. Safonova, and M. Likhachev, "Incremental Planning with Adaptive Dimensionality," *ICAPS*, 2013.

[19] S. Choudhury, O. Salzman, S. Choudhury, C. Dellin, and S. Srinivasa, "Anytime motion planning on large dense roadmaps with expensive edge evaluations," *CoRR*, 2017.

[20] J. Starek, J. V. Gómez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, "An asymptotically-optimal sampling-based algorithm for bi-directional motion planning," *IROS*, 2015.

[21] S. Choudhury, O. Salzman, S. Choudhury, and S. Srinivasa, "Densification strategies for anytime motion planning over large dense roadmaps," in *ICRA*, 2017.

[22] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *IJRR*, 2018.

[23] C. Dellin, "Completing Manipulation Tasks Efficiently in Complex Environments," *PhD Thesis*, 2016.

[24] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial Intelligence*, 1970.

[25] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Formal analysis," *Technical Report CMU-CS-03-148*, 2003.

[26] J. T. Thayer and W. Ruml, "Faster than Weighted A*: An Optimistic Approach to Bounded Suboptimal Search," in *ICAPS*, 2008.

[27] M. Likhachev, G. Gordon, and S. Thrun, "ARA *: Anytime A* with Provable Bounds on Optimality," *NIPS*, 2004.

[28] J. v. d. Berg, R. Shah, A. Huang, and K. Goldberg, "ANA*: Anytime Nonparametric A*," in *AAAI Conference on Artificial Intelligence*, 2011.

[29] C. Wilt and W. Ruml, "When does Weighted A* Fail?" *SoCS*, 2012.

[30] E. Burns, W. Ruml, and M. B. Do, "Heuristic search when time matters," *JAIR*, 2013.

[31] I. Şucan, M. Moll, and L. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012.

[32] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified GPU voxel collision detection for mobile manipulation planning," *IROS*, 2014.

[33] M. Likhachev and S. Koenig, "A generalized framework for lifelong planning A* search," in *ICAPS*, 2005.