D-Lite: Navigation-Oriented Compression of 3D Scene Graphs for Multi-Robot Collaboration

Yun Chang, *Graduate Student Member, IEEE*, Luca Ballotta, *Member, IEEE*, and Luca Carlone, *Senior Member, IEEE*

***Abstract*—For a multi-robot team that collaboratively explores an unknown environment, it is of vital importance that the collected information is efficiently shared among robots in order to support exploration and navigation tasks. Practical constraints of wireless channels, such as limited bandwidth, urge robots to carefully select information to be transmitted. In this letter, we consider the case where environmental information is modeled using a *3D Scene Graph*, a hierarchical map representation that describes both geometric and semantic aspects of the environment. Then, we leverage graph-theoretic tools, namely *graph spanners*, to design greedy algorithms that efficiently compress 3D Scene Graphs with the aim of enabling communication between robots under bandwidth constraints. Our compression algorithms are *navigation-oriented* in that they are designed to approximately preserve shortest paths between locations of interest while meeting a user-specified communication budget constraint. The effectiveness of the proposed algorithms is demonstrated in robot navigation experiments in a realistic simulator.**

***Index Terms*—Communication constraints, semantic scene understanding, graph spanner, multi-robot SLAM, multi-robot systems.**

I. Introduction

IN THE near future, robot teams will perform cooperative tasks in a multitude of application scenarios, ranging from exploration of subterranean environments, to search-and-rescue missions in hazardous settings, to human assistance in houses, airports, factory floors, and malls, to mention a few.

A key requirement for cooperative exploration and navigation in an initially unknown environment is to build a map model of the environment as the robots explore it. Recent work has proposed *3D Scene Graphs* as an expressive hierarchical model of complex environments [1], [2], [3], [4], [5], [6]: a 3D Scene Graph organizes spatial and semantic information, including objects, structures (e.g., walls), places (i.e., free-space locations the robot can reach), rooms, and buildings into a graph with multiple layers corresponding to different levels of abstraction. 3D Scene Graphs provide a user-friendly model of the scene that can support the execution of high-level instructions by a human. Also, they capture traversability between places, rooms, and buildings that can be used for path planning.

To scale up from single- to multi-robot systems and to longer missions and larger environments, a key challenge is to share the map information among the robots to support cooperation. For instance, the robots may exchange partial maps such that a robot can navigate within a portion of the environment mapped by another robot. However, the potentially high volume of data to be transferred over a shared wireless channel easily saturates the available bandwidth, degrading team performance. This holds true especially when the wireless channel is also used to transmit other information in the field —such as images or place recognition information for localization and map reconstruction— which further limits the bandwidth available for transmitting map information in a timely manner [7], [8], [9], [10]. The challenge of information sharing is particularly relevant when the map is modeled as a 3D Scene Graph, since these are rich and potentially large models if all nodes and edges are retained. On the other hand, 3D Scene Graphs also provide opportunities for compression: for instance, the robots may exchange information about rooms in the environment rather than sharing fine-grained traversability information encoded by the place layer; similarly, for a large-scale scene, the robot may just specify a sequence

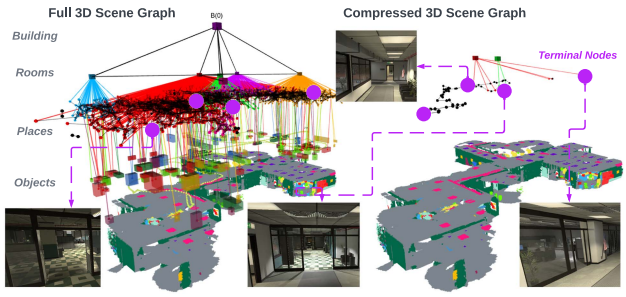


Fig. 1. 3D Scene Graph of an environment (left) and compressed version produced by D-Lite (right). The purple circles mark the *terminal nodes:* D-Lite approximately preserves shortest-path distances between those locations of interest.

Manuscript received 21 April 2023; accepted 8 September 2023. Date of publication 27 September 2023; date of current version 6 October 2023. This letter was recommended for publication by Associate Editor F. Arrichiello and Editor M. A. Hsieh upon evaluation of the reviewers' comments. This work was supported in part by ARL DCIST CRA under Grant W911NF-17-2-0181, in part by ONR RAIDER under Grant N00014-18-1-2828, and in part by Lincoln Laboratory's Resilient Perception in Degraded Environments program, the CARIPARO Foundation Visiting Programme "HiPeR", and the Italian Ministry of Education, University and Research (MIUR) through the PRIN Project under Grant 2017NS9FEY Realtime Control of 5G Wireless Networks. *(Yun Chang and Luca Ballotta contributed equally to this work.) (Corresponding author: Luca Ballotta.)*

Yun Chang and Luca Carlone are with the Laboratory for Information and Decision Systems (LIDS), Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: yunchang@mit.edu; lcarlone@mit.edu).

Luca Ballotta is with the Department of Information Engineering, University of Padova, 35131 Padova, Italy (e-mail: ballotta@dei.unipd.it).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3320011, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3320011

of buildings to be traversed, abstracting away geometric information at lower levels. This is similar to what humans do: when providing instructions to a person about how to reach a location in a building, we would specify a sequence of rooms and landmarks (e.g., objects or structures) rather than a detailed metric map or a precise path.

Therefore, the question we address in this letter is: *how can we compress a 3D Scene Graph to retain relevant information the robots can use for navigation while meeting a communication budget constraint, expressed as the maximum size of the map the robots can transmit?* Besides multi-robot communication, task-driven map compression can play a role in long-term autonomy under resource constraints, where the robots might suffer memory limitations and retain only key portions of a large map. Such a compression is also useful when it is desirable for the robots to share essential information under privacy considerations by sending only task-relevant data [11].

Related work: Graph compression is an active area of research in mathematics, computer science, and telecommunications, where it finds applications to, e.g., vehicle and packet routing [12], [13], [14], and 3D point cloud compression [15], [16], [17].

A prominent body of works simplifies a graph by carefully pruning it to retain relevant information. For example, references [13], [18] find efficient representations of huge web and communication networks by heuristically selecting a few key elements, while the work [19] prunes graphs while preserving connectivity among nodes. Within the discrete mathematics literature, graph compression has been studied with focus on ensuring low distortion (or *stretch*) of inter-node distances. For example, *spanning trees* and *Steiner trees* are the smallest subgraphs maintaining connectivity in undirected graphs [20], [21]. *Graph spanners* remove a subset of edges while allowing for a user-defined maximum distortion of shortest paths [22], [23], [24]. A special case are *distance preservers* [25] that prune graphs but keep unaltered the distances for specified node pairs. *Emulators* are tools that replace a large number of edges with a few strategic ones to ensure small stretch of distances [26].

Related work in robotics focuses on graph compression to speed up path planning and decision-making. Silver et al. [27] use Graph Neural Networks to detect key nodes by learning heuristic importance scores. Agia et al. [28] propose an algorithm that exploits the 3D Scene Graph hierarchy to prune nodes and edges not relevant to the robotic task. Targeting a related application domain, Tian et al. [29] study computation and communication efficiency of multi-robot loop closure, providing a strategy to share a limited number of visual features in multi-robot SLAM, while Denniston et al. [10] introduce a graph-based method to prune the multi-robot loop closures in order to save on processing time. Larsson et al. [30], [31], [32] propose algorithms to build hierarchical abstractions of tree-structured representations, for instance enabling fast planning on occupancy grid maps at progressively increasing resolution.

Novel contribution: In this letter, we tackle the challenging problem of efficiently sharing 3D Scene Graphs for navigation under hard communication constraints. We propose two greedy algorithms, BUD-Lite and TOD-Lite (collectively referred to as *D-Lite*), that leverage graph spanners to prune nodes and edges from a 3D Scene Graph while minimizing the distortion of the shortest paths between locations of interest (*terminal nodes*, see Fig. 1). Compared to the literature, our algorithms (i) are designed to retain navigation-relevant information, (ii) leverage the hierarchical structure of the 3D Scene Graph for compression, and (iii) enforce a user-specified size of the compressed 3D Scene Graph. Our algorithms are computationally efficient and apply to general 3D Scene Graphs. In contrast, related works are either restricted to trees or involve mixed-integer programming [30], [31]. Other pruning strategies do not directly target path planning tasks [28]. Finally, most works tailored to real-time compression do not allow for hard communication constraints [28], [30]. The effectiveness of our algorithms is validated through realistic simulated experiments. We show that the proposed method meets hard communication constraints without excessively impacting navigation performance. For example, navigation time on the compressed graph increases by at most 8% after compressing the 3D Scene Graph to 1.6% of its size.

II. Navigation-Oriented Scene Graph Compression

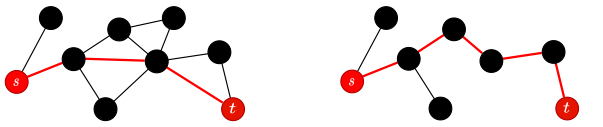
Motivating scenario: We consider a multi-robot team exploring an unknown environment. Each robot navigates to gather information and builds a 3D Scene Graph (DSG) $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ that describes the portion of the environment explored so far [1], [2], [3], [4]. As robots are scattered across a possibly large area, they exchange partial maps to collaboratively gather information about the environment. In particular, a robot r_1 may query another robot r_2 to get information about the area explored by r_2.

Navigation-oriented query: We assume that the querying robot r_1 needs to reach one or more *target locations* $\mathcal{T} \subset \mathcal{V}_\mathcal{G}$ within the DSG $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ built by robot r_2. Such locations, for instance, may be objects or points of interest (e.g., the building exits). Hence, r_2 shall transmit its local map (i.e., nodes and edges of its DSG) such that r_1 can reach locations in $\mathcal{T}$ from a set $\mathcal{S} \subset \mathcal{V}_\mathcal{G} \setminus \mathcal{T}$ of *source locations*. In practice, the latter may represent physical access points (e.g., doors) at the boundary of the area explored by r_2 that are near r_1, and may be estimated by r_2 based on the current location of r_1. In the following, we generically refer to sources and targets as *terminals* (or *terminal nodes*), which for the sake of this work are assumed to be *place* nodes in the DSG.

Communication constraints: Data sharing among robots occurs over a common wireless channel. Because of resource constraints of wireless communication, such as limited bandwidth, robot r_2 cannot transmit its entire DSG to robot r_1. Specifically, we assume that robots can send only a small portion of their DSG each time they receive a share request. Hence, queried robot r_2 needs to compress its DSG $\mathcal{G}$ into a subgraph $\mathcal{G}' = (\mathcal{V}_{\mathcal{G}'}, \mathcal{E}_{\mathcal{G}'})$, with $\mathcal{V}_{\mathcal{G}'} \subseteq \mathcal{V}$ and $\mathcal{E}_{\mathcal{G}'} \subseteq \mathcal{E}$, that contains at most B nodes (where the budget B reflects the available bandwidth) in order to comply with communication constraints, while at the same time retaining information useful for robot r_1 to navigate between the terminal nodes.

Pruning 3D Scene Graphs: Assuming navigation-oriented queries, the relevant information reduces to nodes and edges describing efficient paths robot r_1 can use to move across the map. Specifically, the collection of all shortest paths between a source $s \in \mathcal{S}$ and target $t \in \mathcal{T}$ is the least information ensuring that navigation by r_1 takes the shortest possible time, i.e., the time a robot with full knowledge of the map would take.

However, transmitting all nodes in the shortest paths may violate the communication constraint (see Fig. 6): this can happen with many terminals or if shortest paths have little overlap. Hence, heavier pruning of the DSG might be needed to make communication feasible. This means that information useful



(a) Full graph (length 3). (b) Pruned graph (length 5).

Fig. 2. Distortion of shortest path from s to t (thick red).

for path planning will be partially unavailable to the querying robot's planner. In other words, because the DSG $\mathcal{G}$ cannot be fully sent, the distance (length of a shortest path) between a pair of terminals in the transmitted graph $\mathcal{G}'$ will be larger than the distance between those same terminals in the original DSG. A schematic example is provided in Fig. 2, where the length of the shortest path between nodes s and t increases from 3 to 5 after node and edge removal. For example, a robot may prune place nodes within a room, or share only the room node as a coarse representation of places. This requires less communication, but the querying robot r_1, which receives a coarser map, will be forced to, e.g., take a longer detour across a room, instead of traversing the original shortest path along a set of place nodes. Mathematically, this means $d_{\mathcal{G}'}(s,t) \geq d_{\mathcal{G}}(s,t)$ for any $s \in \mathcal{S}$ and for any $t \in \mathcal{T}$, where $d_{\mathcal{G}}(u,v)$ is the distance from node u to node v in $\mathcal{G}$.

Problem formulation: For the querying robot r_1 to navigate efficiently, the distance $d_{\mathcal{G}'}(s,t)$ between $s \in \mathcal{S}$ and $t \in \mathcal{T}$ in the transmitted graph $\mathcal{G}'$ should not be much larger than the distance in the original graph $\mathcal{G}$. Hence, the queried robot r_2 shall prune $\mathcal{G}$ so as to minimize the *distortion*, or *stretch*, between shortest paths in the original and compressed graphs, while meeting the communication budget B. This can be cast into the following optimization problem:

$$\min_{\mathcal{G}' \subseteq \mathcal{G}} \quad \beta \tag{1a}$$

$$\text{s.t.} \quad d_{\mathcal{G}'}(s,t) \leq d_{\mathcal{G}}(s,t) + \beta W^{\mathcal{G}}_{\max}(s,t) \;\; \forall (s,t) \in \mathcal{P} \tag{1b}$$

$$|\mathcal{V}_{\mathcal{G}'}| \leq B, \tag{1c}$$

where $W^{\mathcal{G}}_{\max}(u,v)$ is the maximum edge weight on a shortest path from u to v in $\mathcal{G}$ and $\mathcal{P} \subseteq \mathcal{S} \times \mathcal{T}$ is the set of considered source-target pairs. Constraint (1c) ensures that the amount of transmitted information (number of nodes) meets the communication constraint, while constraint (1b) and cost (1a) encode minimization of the maximum distortion incurred by the shortest paths. The coefficient $W^{\mathcal{G}}_{\max}(s,t)$ in (1b) makes the distortion computation meaningful for weighted graphs.

Problem (1) can be solved by means of integer linear programming (ILP), see the technical report [33, Appendix A]. However, the runtime complexity of ILP solvers is subject to combinatorial explosion, making this approach impractical for online operation. Hence, we propose greedy algorithms that require lighter-weight computation, based on *graph spanners*. Background about these tools is given in [33, Section III].

III. 3D Scene Graph Compression Algorithms

We propose *D-Lite*, a compression method for DSGs to meet communication constraints with attention to navigation efficiency. We design two versions of D-Lite, which are initialized with a spanner of the full DSG (Section III-B) and tackle the compression problem from opposite perspectives.

The first algorithm, BUD-Lite (Section III-C), performs progressive bottom-up compression of the spanner computed during initialization, exploiting the DSG abstraction hierarchy. In contrast, the second algorithm, TOD-Lite (Section III-D), works top-down expanding nodes with the spanner as a target.

A. Intuition and the Role of the 3D Scene Graph Hierarchy

Assume we want to design a greedy procedure that removes nodes and edges in $\mathcal{G}$ while limiting the incurred path stretch. To this aim, we crucially exploit the *hierarchical structure* of the DSG. We refer to a node m that is adjacent to node n in the upper layer as a *child* of n to stress the hierarchical semantics of the DSG, and symmetrically call node n the *parent* of m. The children of n in graph $\mathcal{G}$ are denoted by $C_{\mathcal{G}}(n)$. Also, the set $\mathcal{E}_{\mathcal{G}}(n)$ gathers all edges incident to n in $\mathcal{G}$. The DSG hierarchy allows us to see a node as a "compressed", or "abstract", representation of its children. Hence, transmitting n rather than $C_{\mathcal{G}}(n)$ saves communication and conveys partial spatial information about nodes in $C_{\mathcal{G}}(n)$. For instance, let $C_{\mathcal{G}}(n)$ represent places inside a room and n the associated room node. A robot that needs to reach a location $t \in C_{\mathcal{G}}(n)$ (e.g., the door) in that room with no bandwidth constraints can be provided with a sequence of places to reach t. Alternatively, the robot can be given the room node n and it needs to explore the room to find the target t: this extra exploration takes longer, degrading navigation performance, but allows for compression to meet communication constraints. The navigation time for local exploration is encoded by the weights of edges connecting non-finest resolution nodes or nodes at different resolutions (layers). For our experiments, we derive such weights from the full DSG as detailed in [33, Appendix B]. However, we argue that a robot can estimate all weights on-the-fly (while building the DSG) based on the actual navigation time it experiences.

The discussion above suggests a simple way to compress the DSG: nodes in a layer can be progressively replaced by their parent nodes in the layer above. Every time we replace nodes with more "abstract" ones (rooms, buildings) the length of the paths passing through those nodes increases, indicating longer navigation. Hence, we can opportunistically select which nodes to "abstract away" so as to achieve a small stretch in the paths between terminals. In alternative, we can start with a coarse representation (including only the highest abstraction level) and expand it to reduce the stretch of the paths. We present these two greedy strategies below and initialize both procedures by computing a spanner of the given DSG, as explained next.

B. Building a DSG Spanner

The literature provides several algorithms to produce spanners of an input graph given a user-specified stretch on the distance between terminals. The spanner need not meet our budget constraint, hence we use it just as initialization for D-Lite. We adapt the algorithm in [24, Section 5] to build a spanner of the full DSG with additive path stretch. The procedure initializes the spanner with a random selection of edges: to exploit the DSG hierarchy, we modify the original algorithm by manually adding cross-layer edges during the initialization. Also, once the spanner is built, we retain only nodes and edges relevant for navigation by removing all nodes that are not traversed

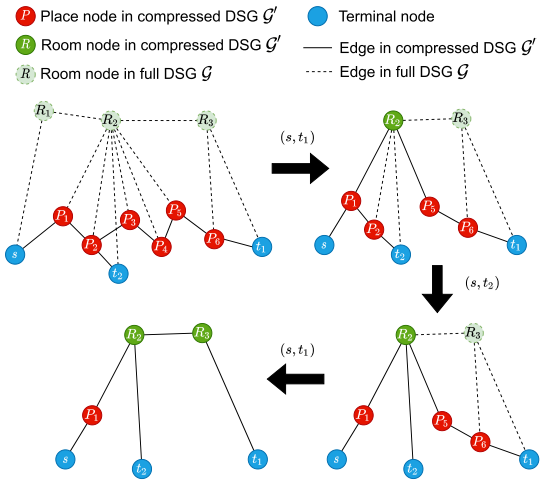


Fig. 3. Illustration of the BUD-Lite procedure with source s and targets t_1, t_2. At each iteration, place nodes in a shortest path between terminals are replaced by a room node.

by shortest paths between terminals in the spanner just built.[1] This greatly reduces the graph to be compressed, making our compression strategies based on hierarchal abstractions more efficient. We call this subroutine `build_spanner`. In the interest of space, we defer more details to [33, Section III-B].

C. BUD-Lite: A Bottom-Up Compression Algorithm

The idea behind our first algorithm (BUD-Lite, short for Bottom-Up D-Lite) is to iteratively *compress* the DSG spanner produced by `build_spanner`. The mechanism is simple: we progressively replace batches of nodes with their parents to reduce size, while attempting to keep the stretch incurred by the shortest paths between terminals low.

To gain intuition, consider Fig. 3 that illustrates three steps of BUD-Lite on a toy DSG.[2] Dashed edges and light-colored nodes are part of the full DSG $\mathcal{G}$ and can be added to the compressed DSG $\mathcal{G}'$. The latter is marked with solid lines and brighter colors. The first iteration of BUD-Lite parses the path from s to t_1 and abstracts away place nodes P_3 and P_4, which are replaced with room node R_2 that is a coarse representation of those places (top right). Room R_1 is skipped because it does not reduce budget as compared to keeping P_1. Place P_2 is not removed yet because it lies also on the path connecting pair (s, t_2), while P_5 is still needed to connect (s, t_1). Node P_2 is removed at the second round when the path from s to t_2 is parsed and shortcut through place node P_1 and room node R_2 (bottom right). The final step parses the last portion of the path connecting (s, t_1) and abstracts away the remaining places P_5 and P_6 under rooms R_2 and R_3 (bottom left). An example on an actual DSG build from simulated data is shown in Fig. 4, where the room node is used to abstract several places.

We formally introduce the compression procedure in Algorithm 1. The compressed graph is initialized as the DSG

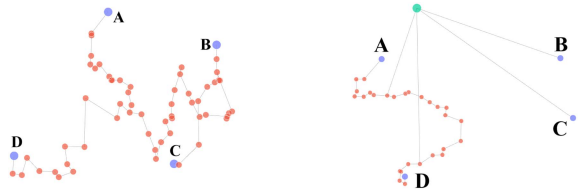


Fig. 4. Initial (left) and final DSG (right). Terminal nodes (**A**, **B**, **C**, and **D**) are in blue, place nodes in red, and the room node in green.

spanner $\mathcal{G}'$ output by `build_spanner` (Line 1). In the following, the symbol $\mathcal{L}_i^{\mathcal{G}}$ refers to the nodes within the ith layer of $\mathcal{G}$. For example, $\mathcal{L}_0^{\mathcal{G}}$ collects all place nodes of the full DSG $\mathcal{G}$. Also, $\mathcal{G}st$ denotes the shortest path between s and t in $\mathcal{G}$. The external loop at Line 4 parses each layer $\mathcal{L}_i^{\mathcal{G}'}$ of $\mathcal{G}'$, starting from the bottom ($i = 0$) and moving to the upper layer $\mathcal{L}_{i+1}^{\mathcal{G}'}$ after $\mathcal{L}_i^{\mathcal{G}'}$ has been compressed (Line 5). At each iteration of the inner loop at Line 6, the algorithm checks if the shortest path connecting terminals s and t in $\mathcal{G}'$ is traversed by nodes in layer $\mathcal{L}_i^{\mathcal{G}'}$ with the same parent node $n \in \mathcal{L}_{i+1}^{\mathcal{G}}$ (Line 7): if this is the case, such nodes with common parent are removed from $\mathcal{G}'st$ and replaced (*compressed*) with their parent node n (Line 8).[3] Such a *compression* in the graph causes a corresponding *stretch* of the actual path followed by the robot, in light of the discussion in Section III-A. The nested structure of Algorithm 1 looping over layers externally (Line 4) and over paths internally (Line 6) stretches distances in a balanced fashion: after the inner loop parses all paths once, each path traversed by nodes in the finest layer is compressed by one coarse node (e.g., a room node replacing place nodes), and such coarse nodes are in the same layer for all compressed paths (e.g., during the first iteration of the outer loop, places can be compressed to rooms but not to buildings). This means that the additional stretch is the same for all paths, on average: differences arise if the paths traverse unbalanced locations, e.g., a path passes through a large room (which yields high distortion) while one through a small room (low distortion). To ensure paths are always feasible, nodes are removed when they are unused (Lines 11 to 14). For BUD-Lite to terminate, the budget B has to accommodate at least minimal-cardinality paths between terminals, which we assume holds true.

Performance bound: We now provide an analytical bound on the worst-case stretch that is incurred by every shortest path after running BUD-Lite. First, we provide two definitions that are instrumental to the understanding of the bound.

Definition 1 (Ancestor): The *(ith) ancestor* $a_{\mathcal{G}}^i(n)$ of node $n \in \mathcal{L}_{i_0}^{\mathcal{G}}$ is the unique node in layer $\mathcal{L}_i^{\mathcal{G}}$, $i > i_0$, that is connected to n by a path composed of only cross-layer edges.

In words, the ancestors of node n are coarse representations of n in upper layers. For example, the first two ancestors of a place node are its room and building nodes, respectively.

Definition 2 (Diameter): For any node $n \in \mathcal{V}_{\mathcal{G}}$, its *diameter* $\text{diam}_{\mathcal{G}}(n)$ is the maximum cardinality of all shortest paths connecting two children nodes of n in $\mathcal{G}$, that is,

$$\text{diam}_{\mathcal{G}}(n) \doteq \max \{ |\mathcal{G}c_1 c_2| : c_1, c_2 \in C_{\mathcal{G}}(n) \}, \qquad (2)$$

[1] We assume that the robot can compute shortest paths between terminals in a reasonable time as compared to the overall compression procedure.

[2] While Fig. 3 considers only place and room layers for the sake of visualization, our algorithm applies to DSGs with any number of layers.

[3] For consistency of navigation, we do not compress terminal nodes in our implementation, but this can be changed to accommodate the budget constraint.

Algorithm 1: BUD-Lite.

Input: DSG $\mathcal{G}$, terminal pairs $\mathcal{P}$, node budget B.
Output: Compressed DSG $\mathcal{G}'$.

1 $\mathcal{G}' \leftarrow$ `build_spanner`$(\mathcal{G}, \mathcal{P})$; // initialization
2 **foreach** $n \in \mathcal{V}_{\mathcal{G}'}$ **do** // track node usage
3 $\quad \mathcal{D}[n] \leftarrow \{(s,t) \in \mathcal{P} : n \in P_{\mathcal{G}'}(s,t)\}$;
4 **for** $i = 0, \dots, L-1$ **do** // parse layers bottom-up
5 $\quad$ **while** $\mathcal{L}_i^{\mathcal{G}'} \neq \emptyset$ **do** // parse layer till empty
6 $\quad\quad$ **foreach** $(s,t) \in \mathcal{P}$ **do** // parse path from s to t
7 $\quad\quad\quad$ **if** $\exists n \in \mathcal{L}_{i+1}^{\mathcal{G}} : P_{\mathcal{G}'}(s,t) \cap C_{\mathcal{G}'}(n) \neq \emptyset$ **then**
8 $\quad\quad\quad\quad$ replace $P_{\mathcal{G}'}(s,t) \cap C_{\mathcal{G}'}(n)$ with n in $P_{\mathcal{G}'}(s,t)$;
9 $\quad\quad\quad\quad$ $\mathcal{D}[n] \leftarrow \mathcal{D}[n] \cup (s,t)$;
10 $\quad\quad\quad\quad$ **foreach** $m \in P_{\mathcal{G}'}(s,t) \cap C_{\mathcal{G}'}(n)$ **do**
11 $\quad\quad\quad\quad\quad$ $\mathcal{D}[m] \leftarrow \mathcal{D}[m] \setminus (s,t)$;
12 $\quad\quad\quad\quad\quad$ **if** $\mathcal{D}[m] = \emptyset$ **then** // prune DSG
13 $\quad\quad\quad\quad\quad\quad$ $\mathcal{V}_{\mathcal{G}'} \leftarrow \mathcal{V}_{\mathcal{G}'} \setminus \{m\}$;
14 $\quad\quad\quad\quad\quad\quad$ $\mathcal{E}_{\mathcal{G}'} \leftarrow \mathcal{E}_{\mathcal{G}'} \setminus \mathcal{E}_{\mathcal{G}'}(m)$;
15 $\quad\quad\quad\quad$ **if** $|\mathcal{V}_{\mathcal{G}'}| \leq B$ **then** // budget is met
16 $\quad\quad\quad\quad\quad$ **return** $\mathcal{G}'$.

where $|\mathcal{G}_{c_1 c_2}|$ denotes the number of nodes in $\mathcal{G}_{c_1 c_2}$.

In words, the diameter of a node describes how "large" the node is when expanded into its children in the layer below.

We now assume the following bounds on quantities associated with the original DSG $\mathcal{G}$. Recall that any edge $(m,n) \in \mathcal{E}_{\mathcal{G}}$ with $m, n \in \mathcal{V}_{\mathcal{G}}$ is assigned a weight $W^{\mathcal{G}}(m,n)$.

Assumption 1 (DSG bounds): For any layer $i \in \{1, \dots, L\}$,

$$\begin{aligned} W_{\max}^{i} &\doteq \max\left\{W^{\mathcal{G}}(m,n) : m,n \in \mathcal{L}_i^{\mathcal{G}}\right\}, \\ W_{\max}^{i-1,i} &\doteq \max\left\{W^{\mathcal{G}}(m,n) : m \in \mathcal{L}_{i-1}^{\mathcal{G}}, n \in \mathcal{L}_i^{\mathcal{G}}\right\}, \\ u_{\min}^{i} &\doteq \min\left\{\left|\mathcal{G}a_{\mathcal{G}}^{i}(s) a_{\mathcal{G}}^{i}(t)\right| : (s,t) \in \mathcal{P}\right\}, \\ \mathrm{diam}_{\min}^{i} &\doteq \min\left\{\mathrm{diam}_{\mathcal{G}}(n) : n \in \mathcal{L}_i^{\mathcal{G}}\right\}. \end{aligned} \tag{3}$$

- $W_{\max}^{i}$ is the maximum weight of edges in layer $\mathcal{L}_i^{\mathcal{G}}$;
- $W_{\max}^{i-1,i}$ is the maximum weight of cross-layer edges between layers $\mathcal{L}_{i-1}^{\mathcal{G}}$ and $\mathcal{L}_i^{\mathcal{G}}$;
- $u_{\min}^{i}$ is the minimum cardinality of a shortest path between the ith ancestors of every two connected terminals;
- $\mathrm{diam}_{\min}^{i}$ is the minimum diameter of nodes in layer $\mathcal{L}_i^{\mathcal{G}}$.

Equipped with the definition above, we can bound the distortion on the compressed DSG $\mathcal{G}'$ provided by BUD-Lite.

Proposition 1 (Worst-case BUD-Lite stretch): After k total iterations of the innermost loop in Algorithm 1, the distance between any two terminals in the compressed graph $\mathcal{G}'$ is

$$\begin{aligned} d_{\mathcal{G}'}(s,t) \;\leq\; & 2\sum_{i=1}^{\ell_{\max}} W_{\max}^{i-1,i} + \left(u_{\min}^{\ell_{\max}-1} - \alpha_k \mathrm{diam}_{\min}^{\ell_{\max}}\right) W_{\max}^{\ell_{\max}-1} \\ & + \alpha_k W_{\max}^{\ell_{\max}}, \quad \forall (s,t) \in \mathcal{P}, \end{aligned} \tag{4}$$

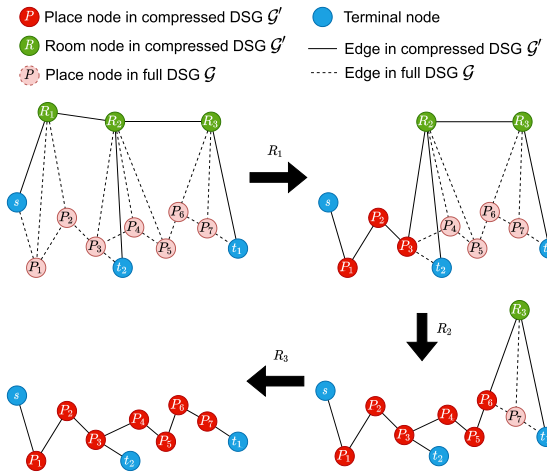


Fig. 5. Illustration of the TOD-Lite expansion procedure with one source s and two targets t_1 and t_2. At each iteration, a room node is expanded and replaced with its children place nodes. Adjacent nodes may be added to ensure connectivity (e.g., P_3 at first iteration).

where

$$\alpha_k \doteq \left\lceil \frac{k}{|\mathcal{P}|} - \sum_{i=\ell_0}^{\ell_{\max}-1} u_{\min}^{i} \right\rceil, \tag{5}$$

$$\ell_{\max} \doteq \max\left\{\ell : k > |\mathcal{P}| \sum_{i=\ell_0}^{\ell-1} u_{\min}^{i}\right\}, \tag{6}$$

$$\begin{aligned} \ell_0 \doteq \max\Bigg\{\ell : & \min_{(s,t)\in\mathcal{P}} \left(d_{\mathcal{G}}(s,t) + \beta W_{\max}^{\mathcal{G}}(s,t)\right) \\ & \geq 2\sum_{i=1}^{\ell} W_{\min}^{i-1,i} + u_{\min}^{\ell} W_{\min}^{\ell}\Bigg\}. \end{aligned} \tag{7}$$

Proof: See the technical report [33, Appendix D]. ■

In words, ℓ_0 is the index of the bottom layer in the initial spanner $\mathcal{G}'$ in Line 1 (excluding terminals); $\ell_{\max}$ is the index of the top (coarsest) layer reached by BUD-Lite after k total iterations; α_k is the number of nodes in the latter layer that have been added to the compressed graph after k iterations. In the bound (4), the first term is the stretch due to cross-layer edges connecting the terminals to nodes in the upper layers, while the other terms are the stretch due to the shortest path passing partially across the coarsest layer $\mathcal{L}_{\ell_{\max}}^{\mathcal{G}}$ (third term) and partially across layer $\mathcal{L}_{\ell_{\max}-1}^{\mathcal{G}}$ (second term).

D. TOD-Lite: A Top-Down Expansion Algorithm

This section presents our second greedy algorithm. Symmetrically to the bottom-up approach of Algorithm 1, the idea behind TOD-Lite (short for TOp-down D-Lite) is to exploit the DSG hierarchy by expanding node children to iteratively increase spatial granularity of the compressed graph (Fig. 5).

The idea of TOD-Lite is depicted with a toy example in Fig. 5, where room nodes R_1, R_2, and R_3 are progressively replaced with their respective children (place nodes). The initial condition (top left) contains the minimum set of nodes that guarantee connectivity between terminals, and it features coarse spatial abstractions through the retained nodes.

We formally describe TOD-Lite in Algorithm 2. During initialization, Algorithm 2 builds a spanner $\mathcal{G}'_{\text{target}}$ of the DSG with `build_spanner`, which is used as target for the final compressed graph $\mathcal{G}'$ (Line 1). Then, it populates a "hierarchical spanner" $\mathcal{H}$ (Line 2): this is simply a graph obtained from the original DSG $\mathcal{G}$ by keeping the spanner $\mathcal{G}'_{\text{target}}$ plus nodes and edges encountered starting from $\mathcal{G}'_{\text{target}}$ and going up the DSG hierarchy all the way to the top layer. Elements unrelated to the ancestors of $\mathcal{G}'_{\text{target}}$ are removed. For example, if $\mathcal{G}'_{\text{target}}$ is made of place nodes, $\mathcal{H}$ includes $\mathcal{G}'_{\text{target}}$, the room nodes associated with those places (with cross-layer edges), and possibly nodes above in the hierarchy, e.g., the buildings containing those rooms. Graph $\mathcal{H}$ is used to expand nodes from coarser to finer layers, as explained next. The expansion priority is given by the number of paths that traverse a node (Lines 3 to 7): this way, expanding a node with high priority restores the spatial resolution of many paths (i.e., those traversing that node).

The main phase is an iterative top-down expansion through the hierarchical spanner $\mathcal{H}$. The output graph $\mathcal{G}'$ is initialized with terminal nodes and paths that connect them under minimal communication budget: these are obtained by connecting each source-target pair to their common ancestor at minimal distance through cross-layer edges (Lines 8 to 12). Then, starting from the top layer $\mathcal{L}_L^{\mathcal{G}'}$ and going down one layer at a time (Line 13), each node in $\mathcal{G}'$ is *expanded* (Lines 15 to 17) till such operation is infeasible (Line 18). In particular, if a node $n \in \mathcal{G}'$ has a set of children $C_{\mathcal{H}}(n)$ in the hierarchical spanner $\mathcal{H}$, then Line 16 removes n from $\mathcal{G}'$ and Line 17 adds to $\mathcal{G}'$ nodes in $C_{\mathcal{H}}(n)$ and their incident edges $\mathcal{E}_{\mathcal{H}}(C_{\mathcal{H}}(n))$.

Expanding nodes gradually restores the geometric granularity of the DSG spanner, because a spatially coarse representation (e.g., room node) is replaced by a group of nodes with fine resolution (e.g., place nodes). This expansion comes at the price of heavier communication burden. Nonetheless, using the hierarchical spanner allows us to narrow the expansion procedure to a small set of navigation-relevant nodes, both saving runtime and helping meet communication constraints.

Note that, with enough communication resources, TOD-Lite would exactly output the target spanner $\mathcal{G}'_{\text{target}}$. Under limited budget, some nodes in $\mathcal{G}'_{\text{target}}$ cannot be expanded, e.g., a room may be used as a coarse representation of its places.

E. Discussion: *BUD-Lite* vs. *TOD-Lite*

BUD-Lite compresses the DSG in a more granular fashion compared to TOD-Lite: that is, it adds distortion to paths more slowly, because it compresses a limited portion of one path at a time. On the other hand, the expansion strategy of TOD-Lite restores all children of a parent node at once. This difference makes BUD-Lite generally slower but able to reach a final graph size closer to the budget, whereas TOD-Lite is typically faster but retains fewer nodes and leads to more distorted paths.

Those differences make the two strategies suited to different scenarios. For instance, a map that includes both large and small rooms may cause TOD-Lite to get stuck after expanding the nodes with the largest number of children, while the path-wise compression of BUD-Lite is less sensitive to heterogeneous maps. On the other hand, to compress a large but homogeneous map with many relevant locations, one may use TOD-Lite to favor compression speed against a slightly worse result.

Algorithm 2: TOD-Lite.

Input: DSG $\mathcal{G}$, terminal pairs $\mathcal{P}$, node budget B.
Output: Compressed DSG $\mathcal{G}'$.

1 $\mathcal{G}'_{\text{target}} \leftarrow$ `build_spanner`$(\mathcal{G}, \mathcal{P})$;
2 $\mathcal{H} \leftarrow$ hierarchical spanner from $\mathcal{G}'_{\text{target}}$;
3 **foreach** $n \in \mathcal{V}_{\mathcal{G}'_{target}}$ **do** // for expansion priority
4 $\quad \mathcal{D}[n] \leftarrow \left|\{(s,t) \in \mathcal{P} : n \in P_{\mathcal{G}'_{\text{target}}}(s,t)\}\right|$;
5 **for** $i = 1, \ldots, L$ **do**
6 $\quad$ **foreach** $n \in \mathcal{L}_i^{\mathcal{H}}$ **do**
7 $\quad\quad \mathcal{D}[n] \leftarrow \sum_{n' \in \mathcal{V}_{\mathcal{H}}(n)} \mathcal{D}[n']$;
8 $\mathcal{V}_{\mathcal{G}'} \leftarrow \mathcal{P}$; // add terminals
9 **foreach** $(s,t) \in \mathcal{P}$ **do** // add cheapest path from s to t
10 $\quad a \leftarrow$ lowest common ancestor of s and t in $\mathcal{H}$;
11 $\quad \mathcal{V}_{\mathcal{G}'} \leftarrow \mathcal{V}_{\mathcal{G}'} \cup \{a\}$;
12 $\quad \mathcal{E}_{\mathcal{G}'} \leftarrow$ edges connecting s and t with a in $\mathcal{G}'$;
13 **for** $i = L, \ldots, 1$ **do** // parse layers top-down
14 $\quad$ **foreach** $n \in \mathcal{L}_i^{\mathcal{G}'}$ **do** // sorted by $\mathcal{D}[n]$
15 $\quad\quad$ **if** *can expand* n *without exceeding* B **then**
16 $\quad\quad\quad \mathcal{V}_{\mathcal{G}'} \leftarrow \mathcal{V}_{\mathcal{G}'} \setminus \{n\}$; $\mathcal{E}_{\mathcal{G}'} \leftarrow \mathcal{E}_{\mathcal{G}'} \setminus \mathcal{E}_{\mathcal{G}'}(n)$;
17 $\quad\quad\quad \mathcal{V}_{\mathcal{G}'} \leftarrow \mathcal{V}_{\mathcal{G}'} \cup C_{\mathcal{H}}(n)$;
$\quad\quad\quad\quad \mathcal{E}_{\mathcal{G}'} \leftarrow \mathcal{E}_{\mathcal{G}'} \cup \mathcal{E}_{\mathcal{H}}(C_{\mathcal{H}}(n))$;
18 $\quad$ **if** *no node in* $\mathcal{L}_i^{\mathcal{G}'}$ *has been expanded* **then**
19 $\quad\quad$ **return** $\mathcal{G}'$.

IV. Experiments

This section shows that our method retains information for efficient navigation while meeting the communication budget constraint. We also show that the algorithms run in real time.

A. Experimental Setup

Besides benchmarking D-Lite against the solution to (1) (label: "Optimum") found via integer linear programming (ILP), we also adapt and compare the compression strategy introduced in [30] (label: "IB"), as discussed below.

Q-Tree search adaptation: The compression approach in [30] builds on the Information Bottleneck (IB) [34]. This approach aims to find a compact representation T of a random variable X by solving a relaxed version of the IB problem,

$$\min_{p(T|X)} I(T;X) - \beta I(T;Y), \tag{8}$$

where $I(T;X)$ is the mutual information between T and X and $I(T;Y)$ represents the information that T retains about a third variable Y that encodes relevant information about X. Parameter β can be seen as a knob to trade amount of relevant information retained in T for compression rate.

To adapt this approach to navigation-oriented DSG compression (since the Q-tree does not encode connectivity within a layer of the scene graph), we define a uniform distribution $p(x)$ over the place nodes. Next, we associate Y with shortest paths between terminals: if place x_i is on the shortest path y_j, then $p(y_j|x_i) = 1$. From the place layer, we build X by propagating $p(x)$ and $p(y|x)$ to upper layers by a weighted sum (cf. [30]). We manually add the terminals if they are not automatically added,

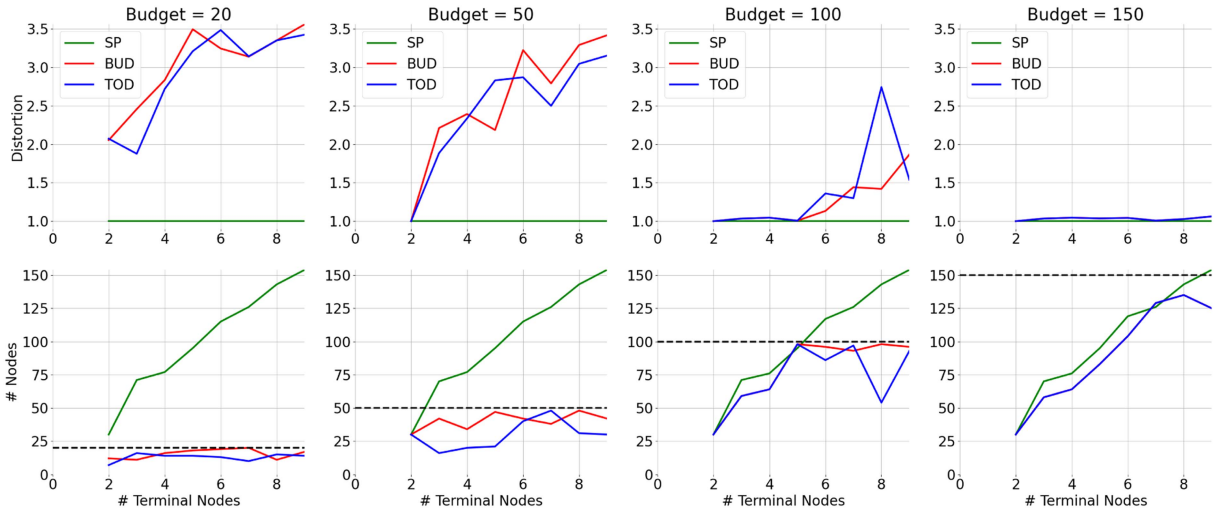


Fig. 6. Comparison on distortion (top row) and number of nodes after compression (bottom row) for BUD-Lite and TOD-Lite against computing the shortest paths (SP) and pruning all nodes that are not on them. The dotted lines mark the communication budget.

TABLE I
SUMMARY OF RESULTS

		Full	Optimum	IB	BUD-Lite	TOD-Lite
short	Comp[s] ↓	0 ± 0	247 ± 0	**1 ± 0**	**3 ± 0**	**3 ± 0**
	Nom[s] ↓	11 ± 0	**11 ± 0**	43 ± 0	**11 ± 0**	**11 ± 0**
	Mis[s] ↓	64 ± 8	56 ± 5	115 ± 16	**62 ± 2**	**59 ± 3**
	Size(#) (≤60)	3814 ± 0	51 ± 0	60 ± 0	49 ± 0	49 ± 0
medium	Comp[s] ↓	0 ± 0	294 ± 1	**1 ± 0**	**3 ± 0**	**3 ± 0**
	Nom[s] ↓	18 ± 0	**18 ± 0**	42 ± 0	**18 ± 0**	29 ± 0
	Mis[s] ↓	87 ± 7	**77 ± 2**	92 ± 22	**85 ± 8**	144 ± 20
	Size(#) (≤60)	3814 ± 0	56 ± 0	60 ± 0	48 ± 0	58 ± 0
long1	Comp[s] ↓	0 ± 0	-	**2 ± 0**	**3 ± 0**	**3 ± 0**
	Nom[s] ↓	27 ± 0	-	∞	**31 ± 0**	39 ± 0
	Mis[s] ↓	134 ± 5	-	∞	**167 ± 18**	273 ± 22
	Size(#) (≤60)	3814 ± 0	-	60 ± 0	58 ± 0	20 ± 0
long2	Comp[s] ↓	0 ± 0	-	**2 ± 0**	**3 ± 0**	**3 ± 0**
	Nom[s] ↓	32 ± 0	-	36 ± 0	**33 ± 0**	34 ± 0
	Mis[s] ↓	150 ± 6	-	218 ± 20	**164 ± 30**	291 ± 39
	Size(#) (≤60)	3814 ± 0	-	60 ± 0	60 ± 0	9 ± 0

Arrows indicate that lower is better. The table reports mean and standard deviation across three runs.

and in view of (1) we use the number of nodes as a stopping condition besides the one in [30].

Simulator: We showcase the online operation of D-Lite in the Office environment of the uHumans2 simulator (Fig. 1) [35], with 4 scenarios featuring different distances between navigation goal and starting position of the robot.

The queried robot r_2 sending the compressed DSG is given potential locations the querying robot r_1 may come from. The places closest to these source locations along with the place closest to the navigation goal are the terminals of D-Lite. In the short and medium sequences, r_1 gets two putative source locations, hence three total terminals. In the two long sequences, r_1 gets three putative source locations, hence four total terminals. For all sequences, we choose 60 nodes as communication budget, which is 1.6% of the original DSG.

Upon receiving the compressed DSG, robot r_1 finds the place node s closest to its location and computes the shortest path on the compressed DSG from s to the place node t that represents the goal. Robot r_1 treats the nodes along the shortest path as navigation waypoints. We combine waypoint following with the ROS navigation stack for local obstacle avoidance: the latter is needed to allow navigation in the areas where low-level geometric information (i.e., places nodes) has been abstracted away during the DSG compression.

B. Results and Discussion

Comparison with baselines: The results on the four scenarios are documented in Table I. We show the compression time (label: "Comp"), the nominal (label: "Nom", computed from the compressed DSG) and simulated (label: "Mis", computed as the actual time r_1 takes to reach its destination in the simulator) mission times, and the size of the compressed DSG (upper bounded by the budget B), all averaged across three runs.[4] The two best results for each row are in bold.

The combinatorial nature of problem (1) makes the ILP solver impractical in robotic applications: for the long runs, the calculation of Optimum did not finish within an hour.

In all scenarios, robot r_1 reaches the goal using the compressed DSG output by D-Lite. The simulated mission time is at times faster on the compressed graph compared to the full DSG because the former has fewer waypoints: a sparser list of waypoints in a less cluttered space can actually yield faster navigation. The different performance of BUD-Lite and TOD-Lite is due to the different abstraction mechanisms, whereby the path-wise node compression in the former yields finer granularity and usually better performance. Discrepancies between nominal and simulated mission times are due to local navigation, whose exploration time is difficult to estimate *a posteriori* from the full DSG. D-Lite always outperforms IB in terms of both

[4]The nominal mission time is computed by projecting the waypoints found by r_1 in the compressed DSG onto the full DSG, calculating the total path length of traversing through those on the full DSG, and dividing by the maximum velocity of the agent. In other words, it is the theoretical navigation time on the original DSG and measures quality of compression. Note that we do not directly use the compressed DSG to estimate the nominal time because the cross-layer edge weights would be different and likely smaller in value compared to those calculated on the full DSG, see [33, Appendix B].

nominal and simulated mission times. Specifically, the navigation planned on the compressed DSG produced by BUD-Lite is only a minute longer than the optimal path on the original DSG. IB is also unable to find a compressed DSG that preserves the necessary connectivity for the long1 case.

Ablation study: We compare distortion and number of nodes on the shortest paths between terminal nodes, for increasing number of terminal nodes and increasing budget constraints in Fig. 6. The shortest paths are optimal in terms of navigation performance (no distortion, top row), but easily violate the communication constraints (exceeding the budget, bottom row). On the other hand, BUD-Lite and TOD-Lite trade-off the path lengths between the terminal nodes to meet the budget constraint, and as we relax the latter, the distortion decreases. For the case with a budget of 150 nodes (last column), BUD-Lite and TOD-Lite obtain the same results, since the initial spanner already satisfies the budget constraint.

A study on the runtime of D-Lite and a comparison of BUD-Lite against the bound (4) are provided in [33, Section V-B].

V. Conclusion

Motivated by collaborative multi-robot exploration, we proposed a method to compress 3D Scene Graphs under communication constraints. Our algorithms can accommodate a sharp node budget while retaining navigation performance. Realistic simulations validate the effectiveness of our approach.

References

[1] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, "3D dynamic scene graphs: Actionable spatial perception with places, objects, and humans," in *Proc. Robot.: Sci. Syst.*, 2020. [Online]. Available: https://www.roboticsproceedings.org/

[2] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A real-time spatial perception engine for 3D scene graph construction and optimization," in *Proc. Robot.: Sci. Syst.*, 2022. [Online]. Available: https://www.roboticsproceedings.org/

[3] I. Armeni et al., "3D scene graph: A structure for unified semantics, 3D space, and camera," in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 5664–5673.

[4] S. Wu, J. Wald, K. Tateno, N. Navab, and F. Tombari, "SceneGraphFusion: Incremental 3D scene graph prediction from RGB-D sequences," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 7511–7521.

[5] U.-H. Kim, J.-M. Park, T.-j. Song, and J.-H. Kim, "3-D scene graph: A sparse and semantic representation of physical environments for intelligent agents," *IEEE Trans. Cybern.*, vol. 50, no. 12, pp. 4921–4933, Dec. 2020.

[6] R. Talak, S. Hu, L. Peng, and L. Carlone, "Neural trees for learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 26395–26408.

[7] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual SLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 2466–2473.

[8] Y. Tian, K. Khosoussi, M. Giamou, J. P. How, and J. Kelly, "Near-optimal budgeted data exchange for distributed loop closure detection," in *Proc. Robot.: Sci. Syst.*, 2018. [Online]. Available: https://www.roboticsproceedings.org/

[9] Y. Chang, Y. Tian, J. How, and L. Carlone, "Kimera-multi: A system for distributed multi-robot metric-semantic simultaneous localization and mapping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 11210–11218.

[10] C. Denniston et al., "Loop closure prioritization for efficient and scalable multi-robot SLAM," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 9651–9658, Oct. 2022.

[11] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models," *Intl. J. Robot. Res.*, vol. 36, no. 12, pp. 1286–1311, 2017.

[12] A. Becker, P. N. Klein, and D. Saulpic, "A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs," in *Proc. Annu. Eur. Symp. Algorithms*, vol. 87, 2017, pp. 12:1–12:15.

[13] A. C. Gilbert, T. Labs-Research, P. Avenue, F. Park, and K. Levchenko, "Compressing network graphs," in *Proc. LinkKDD Workshop ACM Conf. KDD*, 2004, vol. 124.

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

[15] R. L. d. Queiroz and P. A. Chou, "Compression of 3D point clouds using a region-adaptive hierarchical transform," *IEEE Trans. Image Process.*, vol. 25, no. 8, pp. 3947–3956, Aug. 2016.

[16] X. Sun, H. Ma, Y. Sun, and M. Liu, "A novel point cloud compression algorithm based on clustering," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2132–2139, Apr. 2019.

[17] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3D point cloud sequences," *IEEE Trans. Image Process.*, vol. 25, no. 4, pp. 1765–1778, Apr. 2016.

[18] S. Raghavan and H. Garcia-Molina, "Representing web graphs," in *Proc. Int. Conf. Data Eng.*, 2003, pp. 405–416.

[19] C. Chekuri, T. Rukkanchanunt, and C. Xu, "On element-connectivity preserving graph simplification," in *Proc. Algorithms - ESA*, Berlin, Heidelberg, 2015, pp. 313–324.

[20] P. Harish, P. J. Narayanan, V. Vineet, and S. Patidar, "Fast minimum spanning tree computation," in *GPU Computing Gems Jade Edition*. Amsterdam, The Netherlands: Elsevier, Jan. 2012, ch. 7, pp. 77–88.

[21] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs," *Inf. Process. Lett.*, vol. 27, no. 3, pp. 125–128, Mar. 1988.

[22] R. Ahmed et al., "Graph spanners: A tutorial review," *Comput. Sci. Rev.*, vol. 37, Aug. 2020, Art. no. 100253.

[23] Y. Kobayashi, "An FPT algorithm for minimum additive spanner problem," in *Proc. Int. Symp. Theor. Aspects Comp. Sci.*, 2020, vol. 154, pp. 11:1–11:16.

[24] M. Elkin, Y. Gitlitz, and O. Neiman, "Improved weighted additive spanners," *Distrib. Comput.*, vol. 36, no. 3, pp. 385–394, 2022.

[25] G. Bodwin, "On the structure of unique shortest paths in graphs," in *Proc. Annu. ACM-SIAM Symp. Discrete Algorithms*, 2019, pp. 2071–2089.

[26] M. Elkin and O. Neiman, "Efficient algorithms for constructing very sparse spanners and emulators," *ACM Trans. Algorithms*, vol. 15, no. 1, pp. 1–29, Nov. 2018.

[27] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, May 2021, vol. 35, no. 13, pp. 11962–11971.

[28] C. Agia et al., "Taskography: Evaluating robot task planning over large 3D scene graphs," in *Proc. 5th Conf. Robot Learn.*, 2022, pp. 46–58.

[29] Y. Tian, K. Khosoussi, and J. P. How, "A resource-aware approach to collaborative loop-closure detection with provable performance guarantees," *Int. J. Robot. Res.*, vol. 40, no. 10-11, pp. 1212–1233, Sep. 2021.

[30] D. T. Larsson, D. Maity, and P. Tsiotras, "Q-tree search: An information-theoretic approach toward hierarchical abstractions for agents with computational limitations," *IEEE Trans. Robot.*, vol. 36, no. 6, pp. 1669–1685, Dec. 2020.

[31] D. T. Larsson, D. Maity, and P. Tsiotras, "Information-theoretic abstractions for resource-constrained agents via mixed-integer linear programming," in *Proc. Workshop Comput.-Aware Algorithmic Des. Cyber- Phys. Syst.*, 2021, pp. 1–6.

[32] D. T. Larsson, D. Maity, and P. Tsiotras, "Information-theoretic abstractions for planning in agents with computational constraints," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7651–7658, Oct. 2021.

[33] Y. Chang, L. Ballotta, and L. Carlone, "D-lite: Navigation-oriented compression of 3d scene graphs for multi-robot collaboration," 2022, *arXiv:2209.06111*.

[34] N. Tishby, F. Pereira, and W. Bialek, "The information bottleneck method," in *Proc. Annu. Allerton Conf. Commun. Control Comput.*, 2001, vol. 49, pp. 368–377.

[35] A. Rosinol et al., "Kimera: From SLAM to spatial perception with 3D dynamic scene graphs," *Int. J. Robot. Res.*, vol. 40, no. 12–14, pp. 1510–1546, 2021.