Informed Steiner Trees: Sampling and Pruning for Multi-Goal Path Finding in High Dimensions

Nikhil Chandak, Kenny Chour, Sivakumar Rathinam, Senior Member, IEEE, and R. Ravi

Abstract—We interleave sampling based motion planning methods with pruning ideas from minimum spanning tree algorithms to develop a new approach for solving a Multi-Goal Path Finding (MGPF) problem in high dimensional spaces. The approach alternates between sampling points from selected regions in the search space and de-emphasizing regions that may not lead to good solutions for MGPF. Our approach provides an asymptotic, 2-approximation guarantee for MGPF. We also present extensive numerical results to illustrate the advantages of our proposed approach over uniform sampling in terms of the quality of the solutions found and computation speed.

Note to Practitioners—MGPF is concerned with finding a collision-free, near-optimal path for a robot visiting a set of target configurations. This problem arises in applications that use robotic manipulators such as advanced manufacturing, surface inspection, package sorting, and in other logistical applications where the cost of the traveling between any two configurations of a robot cannot be readily determined a-priori. As robots are expected to perform a large number of tasks, the sequencing of these tasks become important specifically when the travel costs are challenging to estimate. This paper provides an approach to handle this problem in higher dimensions with theoretical guarantees as well as provides simulation results on a broad class of environments to corroborate its performance with respect to the state of the art.

Index Terms—Shortest path problem, Steiner trees, approximation algorithms, motion planning.

I. INTRODUCTION

MULTI-GOAL Path Finding (MGPF) problems aim to find a least-cost path for a robot to travel from an origin (s) to a destination (d) such that the path visits each node in a given set of goals (T) at least once. In the process of finding a least-cost path, MGPF algorithms also find an optimal sequence in which the goals must be visited. When the search space is discrete (i.e., a finite graph), the cost of traveling between any two nodes can be computed using an all-pairs shortest paths algorithm. In this case, the MGPF encodes a variant of the Steiner Traveling Salesman Problem (TSP) and is NP-Hard [28]. In the general case, the search space is continuous and the least cost to travel between any two nodes is not known a-priori. This least-cost path computation between any two nodes in the presence of obstacles, in itself, is one of the most widely studied problems in robot motion planning [27], [30]. We address the general case of MGPF as it naturally arises in active perception [1], [36], surface inspection [7] and logistical applications [24], [35], [40]. Specifically, industrial applications such as spot welding, spray-painting or drilling [42], [46] use robot manipulators with end-effectors to travel through a set of target configurations to perform their respective tasks. While the target configurations are known, the sequence in which these configurations must be visited are not known. In addition, the cost of traveling between any two configurations is not easy to pre-compute and is independent of the sequence in which the target configurations are visited.

MGPF is notoriously hard as it combines the challenges in Steiner TSP and the least-cost path computations in the presence of obstacles; hence, we are interested in finding approximate solutions for MGPF. Irrespective of whether the search space is discrete or continuous, Steiner trees spanning the origin, goals and the destination play a critical role in the development of approximation algorithms for MGPF. In the discrete case, doubling the edges in a suitable Steiner tree, and finding a feasible path in the resulting Eulerian graph leads to 2-approximation algorithms for MGPF [3], [28], [38]. This approach doesn’t readily extend to the continuous case because we do not a-priori know the travel cost between any two nodes in \( T := \{s, t\} \cup \bar{T} \). One can appeal to the well-known sampling-based methods [18], [19], [26], [27] to estimate the costs between the nodes, but the following key questions remain: 1) How to sample the space so that the costs of the edges joining the nodes in \( T \) can be estimated quickly so that we can get a desired Steiner tree? 2) Should we estimate the cost of all the edges or can we ignore some edges and focus our effort on edges we think will likely end up in the Steiner tree?

We call our approach Informed Steiner Tree (IST). IST iteratively alternates between sampling points in the search space and pruning edges. Throughout its execution, a Steiner tree is maintained which is initially empty but eventually spans the nodes in \( T \), possibly including a subset of sampled points. IST relies on two key ideas. First, finding a Steiner tree spanning \( T \) commonly involves finding a Minimum Spanning

\({}^1\)Any node that is not required to be visited is referred to as a Steiner node. A path may choose to visit a Steiner node if it helps in either finding feasible solutions or reducing the cost of travel.
The Steiner tree (MST) in the metric completion of the nodes in $T$. For any two distinct terminals $u, v$, we maintain a lower bound and an upper bound on the cost of the edge $(u, v)$. Using these bounds and cycle properties of an MST, we identify edges which will never be part of the MST. This allows us to sample regions corresponding to only those edges that can be part of the MST. We further bias our sampling by assigning a suitable probability distribution over the search space based on the bounds on the cost of the edges. For each new sample added, we may facilitate a lower-cost feasible path between terminals requiring us to frequently update the Steiner tree. To address this efficiently, we develop an incremental version of the Steiner tree algorithm while maintaining its properties. Since this incremental approach correctly finds a Steiner tree, the number of sampled points tends to infinity, IST* provides an asymptotic 2-approximation guarantee for MGPF.

We use the sampling procedure developed in Informed RRT* [17] to choose points from selected regions in our approach. Informed sampling in synergy with pruning enables faster convergence to the optimal solution than uniform sampling. After describing IST* with its theoretical properties, we provide extensive computational results on high-dimensional problem instances. We release IST* as an open-source package.  

### II. RELATED WORK

The MGPF problem belongs to a general class of integrated task and motion planning problems which have been studied in the literature [16], [20], [33], [41]. Here, we provide an extensive review of the work relevant to the MGPF in continuous domains.

2The metric completion here is a complete weighted graph on all the nodes in $T$ where the cost of an edge between a pair of nodes in $T$ is the minimum cost of a path between them.

3Upper bound is the cost of a feasible path from $u$ to $v$.

4https://github.com/nikhilchandak/InformedSteinerTrees

In [5], a meta-heuristic similar to simulated annealing is combined with multiple Rapidly-exploring Random Tree (RRT) expansions to solve MGPF. In [8], LazyTSP was introduced for efficient path planning that combines RRT-based planning with TSP [25]. It starts with TSP to compute an initial tour and then verifies the connections using RRT-based planning, iteratively refining the tour until a valid trajectory is found. In [48], a method called Space Filling Forests (SFF) is proposed to solve MGPF. Multiple trees are grown from the nodes in $T$ and unlike the approach in [5] where any two close enough (or neighboring) trees are merged into a single tree, in SFF, multiple virtual connections are allowed between two neighboring trees leading to multiple paths between any two nodes in $T$. Recently, in [24], a generalization of SFF (called SFF*) has been proposed to also include rewiring of the edges in the trees similar to RRT*. Computational results show that SFF* performed the best among several previously known solvers for the simulation environments considered in [24]. We evaluated all the above methods with our proposed approach and report our findings in the results section.

A generalization of MGPF was considered in [42] where all the goals were partitioned into groups and the aim is to also visit one goal from each partition. They work under the assumption that a path between two nodes is computed at most once which may, in reality, be far from the optimal path. Their planner stops the instant it has found a feasible solution path unlike our approach which keeps refining the paths.

Un-supervised learning approaches [9], [11], [15] using Self Organizing Maps (SOMs) have been used to solve MGPF. SOM is a two-layer neural network and is an iterative procedure that provides a nonlinear mapping between a higher dimensional space (set of goals) to a lower dimensional one (goal ordering). SOM also relies on an underlying distance metric,  

---

Since the code for these learning approaches were not available, we could not directly test them on the problem instances considered in this paper.

---

Fig. 1. The Steiner tree (thick-blue lines) computed in three consecutive iterations of IST*, showing the advantage of pruning. The environment has 2 U-shaped obstacles with tiny openings. The numbered green circles denote the terminals in $T$ that must be connected. A yellow-shade ellipse denotes an edge’s informed set while its color intensity is the probability of that edge being sampled. Sampled points in free space are part of the roadmap (shown in grey). Left: The thin solid red lines are the actual paths corresponding to active non-MST edges under consideration (their edge costs act as upper bounds on the corresponding optimal costs of the edges). Until any pruning, these edges are also sampled in addition to the edges in the MST. Middle: Dashed red lines show the edges that have been discarded (pruned) from further consideration as they will not contribute to the optimal MST. Consequently, their regions will be not be sampled any further. Right: After pruning, strategic densification of the roadmap happens (for example, the region around the edge between nodes 2 and 5) based on the probabilities at the end of the previous iteration. We observe that the roadmap is further densified around active edges leading to an optimal MST faster.
which can be Euclidean [15], visibility graph-based [11], [14],
or sampling-based graph approximation [9], [47]. In [9],
SOM is combined with a rapidly exploring random graph
algorithm to find feasible solutions for the MGPF. However,
in comparison to our asymptotic 2-approximation guarantee,
there is no guarantee on the quality of solutions obtained using
SOMs.

In [11] and [14], a point robot moves in an environment
represented as a polygonal domain, and a convex polygon
partition is used for workspace decomposition. Such assump-
tions may not be straightforward to generalize, for example,
such a decomposition is not possible even in common robotic
environments like SE(3); thus, making them incomparable to
IST in the environments considered.

Multi-goal motion planning has also been explored in appli-
cation to data collection. Reference [10] consider this problem
in planar environment with neighborhood and prize on the
goals. The vehicle is required to visit only the neighborhood
of each goal and all the goal (regions) are not given the same
priority. This makes their problem significantly different from
the one we have addressed. Similarly, [37] have considered
rewards on goals but also assume the fact that both goals and
rewards are unknown to the operating vehicle in the beginning
and become known only when a goal comes under its sensing
radius.

Reference [12] consider the generalized traveling salesman
problem with neighborhoods (GTSPN) specifically for 3D
environments, where an individual neighborhood may consist
of multiple regions (thus, forming a neighborhood set), and
the problem is to determine a shortest multi-goal path to
visit at least one region of each neighborhood. They propose
two heuristics for solving the GTSPN with neighborhoods
defined as polyhedra and ellipsoids to quickly find a feasible
solution. The heuristics exploit properties of the 3D instances
with convex regions making their approach inapplicable for
comparison with ours. Further, they mention that an extension
of their proposed approach for high-dimensional problems
is a subject of their future work, which precisely forms the
motivation for our work.

Other papers have addressed variations of multi-goal motion
planning taking into account additional constraints like Dubins
vehicle [13], energy-aware planning with recharging sta-
tions [49], availability of physics-based game engine to model
dynamics [6] which are outside our scope.

The approaches in [6] and [7] are based on a hybrid search
that uses generalized traveling salesman paths defined over a
probabilistic road map to guide the sampling-based expansion
of a motion tree. The method in [6] works effectively in
two dimensions using a triangulation-based decomposition
approach for partitioning spaces. However, triangulations are
inherently two-dimensional structures and do not straightforwardly extend to higher dimensions. Further, decomposing
obstacles and their interaction with the environment using
triangulations would require constructing higher-dimensional
simplices or polytopes, which are computationally expensive
and difficult to represent. Thus, the approach in [6] seems to
incur challenges when applied to higher dimensions, which is
the prime focus of our work. The method called DROMOS

in [7] deals with the generalized (or one-in-a-set) TSP and
includes motion constraints from vehicle dynamics. The basic
problem we consider only deals with a TSP and our approach
doesn’t explicitly deal with motion constraints. In addition,
DROMOS deals with a Hamiltonian path while ours deals with
a TSP tour; while on paper these are minor modifications,
the approximation guarantees can be quite different for both
variants [23].

There is also an extensive literature on efficiently construct-
ing minimal-cost Euclidean Steiner Tree spanning a set of
vertices on a plane avoiding obstacles, having applications in
VLSI designs and ascent assembly engineering [4], [31], [50].
However, all such works assume the underlying roadmap graph
(whose nodes act as Steiner points) to be given up-front as
input rather than constructing and refining it over time to
reduce the path cost.

Finally, we would like to emphasize that the major differ-
ce in our proposed approach with respect to prior work is to
adapt informed sampling (with a pruning method based on the
current bounds on the edge lengths) to the setting of multiple
goals enabling faster convergence to an asymptotically 2-
approximate solution.

III. BACKGROUND AND PRELIMINARIES

Let $X \subseteq \mathbb{R}^n$ be the n-dimensional configuration space
of a robot, and let $X_f \subseteq X$ be the set of obstacle-free
configurations. Let $\sigma : [0, 1] \rightarrow X_f$ be a collision free,
feasible path such that the path starts at the origin ($\sigma(0) = s$),
visits each of the goals in $T$ at least once and ends at the
destination ($\sigma(1) = d$). Let $c(\sigma) \in \mathbb{R}_{\geq 0}$ denote the cost
of the path $\sigma$. The objective of MGPF is to find a collision-free
feasible path $\sigma$ such that $c(\sigma)$ is minimized.

Let $S \subseteq X_f$ be a set of sampled points (also referred to as
nodes) from $X_f$. Let $V := S \cup T$. We refer to all the nodes in
$T$ as terminals.\footnote{This is a commonly used term in the optimization literature.} Nodes which are within a radius of $\rho$ of each
other are treated as neighbors. Further, let $N(\rho)(u)$ denote the set
of nodes which are neighbors to the node $u$. Edges are formed
between neighbors if a feasible path is found between them
using a local planner (for example, straight-line connection).
Let $G = (V, E)$ denote the undirected graph thus formed
where $E$ denotes all the edges joining any pair of neighbors
in $V$. Henceforth, we refer to $G$ as the roadmap. Given a graph
$G$, we use $E(G)$ to refer to all the edges present in $G$.

Let $e := (u, v)$ be an edge joining two distinct nodes $u, v$ in
$G$. Edge $e$ represents a feasible path between $u$ and $v$ found by
the local planner. The cost\footnote{We assume the costs are symmetric, i.e., cost of traveling a given path
from node $u$ to node $v$ is the same as the cost of traveling the path from $v$ to $u$.} of a feasible path joining vertices
$u$ and $v$ is denoted as $dist(e)$. Let $h(e)$ be a lower bound on
the length of any feasible path between $u$ and $v$; typically, we
set $h(e)$ to be the Euclidean distance between nodes $u$ and $v$.

One approach for solving the MGPF is to sample as many
points uniformly as possible from the obstacle-free space
and form a roadmap spanning the terminals and the sampled
points. At the end of the search process, the recently developed
Algorithm 1 IST*

1. **Input:**
2. $X_f \subset X$ // Collision free space
3. $T \subset X_f$ // Discrete set of terminals
4. $h(e)$ for any $e = (u, v), u, v \in X_f$ // Heuristic cost between terminals $u$ and $v$
5. $n_s$ // Samples per batch
6. $n_b$ // Number of batches
7. **Output:**
8. $S_T$ // Steiner tree spanning the terminals in $T$
9. **Initialization:**
10. $V \leftarrow T$ // Add terminals to the roadmap
11. $\mathcal{N}_T(u) := \{v : \text{dist}(v, u) \leq \rho, v \neq u, v \in V\}$ for any $u \in V$ // Nearest neighbors of $u$ in $V$
12. $E \leftarrow \{(u, v) : v \in \mathcal{N}_T(u), u \in V\}$
13. $\text{cost}(e) = \text{dist}(u, v)$ for any $e = (u, v) \in E$
14. $G \leftarrow (V, E)$
15. $\mathcal{A} = \{(u, v) : u, v \in T, u \neq v\}$ // Set of active edges joining terminals
16. $\text{cost}_T(u, v) = \infty$ for any $(u, v) \in \mathcal{A}$
17. $\text{Prob}(e) = \frac{h(u)}{\sum_{e \in \mathcal{A}} h(e)}$ for all $e \in \mathcal{A}$ // Initial probabilities using lower bounds
18. $S_T \leftarrow \emptyset$
19. // Note: $G, \tilde{G}, \text{cost}, \text{cost}_T$ are global variables.
20. **Main loop:**
21. for iteration $= 1, \ldots, n_b$ do
22. \hspace{1em} $S' \leftarrow \text{AddSamples}(\text{Prob}, \mathcal{A}, \text{cost}_T, n_s)$
23. \hspace{1em} $\text{Ripple}(S')$
24. \hspace{1em} $S_T \leftarrow \text{UpdateTree}(S_T, \mathcal{A})$
25. \hspace{1em} $\mathcal{A} \leftarrow \text{PruneEdges}(S_T, \mathcal{A})$
26. \hspace{1em} $\text{Prob} \leftarrow \text{UpdateProbability}(\text{Prob}, \mathcal{A}, S_T, h)$
27. return $S_T$

Algorithm 2 AddSamples(Prob, $\mathcal{A}$, cost$_T$, $n_s$)

1. $S' \leftarrow \emptyset$
2. for $1, \ldots, n_s$ do
3. \hspace{1em} Choose $(u, v) \in \mathcal{A}$ using $\text{Prob}$
4. \hspace{1em} $x_{\text{rand}} \leftarrow \text{Sample}(u, \text{cost}_T(u, v))$
5. \hspace{1em} $S' \leftarrow S' \cup \{x_{\text{rand}}\}$
6. return $S'$

A* based approach [3] for discrete version of MGPF can be used to find a suitable Steiner tree. In this article, we consider this approach as the baseline against which IST* is compared.

IV. INFORMED STEINER TREES

An overview of IST* is presented in Algorithm 1. At any iteration, IST* maintains three graphs to enable the search process:

1) **Roadmap graph:** $G = (V, E)$ includes all the terminals and the sampled points, and the edges between its neighboring nodes. Two nodes $u, v$ are neighboring if $\text{dist}(u, v) \leq \rho$, where $\rho$ is the connection radius. We provide a discussion on the role (and implementation) of $\rho$ in the Appendix. Initially, $G$ includes only the terminals and the edges between any pair of neighboring terminals.

2) **Shortest-path graph:** This graph is denoted as $\tilde{G} := (V, \tilde{E})$ and is a subgraph of the roadmap. It is a forest which contains a tree rooted at each terminal and maintains all the shortest paths from a terminal to each sampled point in its component. The shortest-path graph is used to find feasible paths between terminals. Initially, $G$ is an empty forest (i.e., contains no edges).

3) **Terminal graph:** This graph is denoted as $G_T = (T, \mathcal{A})$ and consists of only the terminals and any edges connecting them. Any edge in $G_T$ corresponds to a path between two terminals in the roadmap. Therefore, any spanning tree $S_T$ in the terminal graph corresponds to a feasible Steiner tree in $G$. For example, the leftmost subfigure in Fig. 1 shows the terminal graph. It is used in determining which regions to sample and in finding feasible Steiner trees for MGPF. $\mathcal{A}$ contains the set of all the edges that can possibly be part of $S_T$, and hence are currently actively explored when sampling. Initially, $\mathcal{A}$ consists of all the edges that join any pair of terminals in $T$. As the algorithm progresses, edges in $\mathcal{A}$ may be pruned. Also, the cost of an edge connecting any pair of distinct terminals $u, v$ in $G_T$ is denoted as $\text{cost}_T(u, v)$. Initially, $\text{cost}_T(u, v)$ is set to $\infty$ for all $u, v \in T$ and $S_T$ is empty.

In each iteration of IST*, the algorithm (i) samples new points in $X_f$ (line 21 of Algorithm 1), (ii) updates $S_T$ based on a new incremental Steiner tree algorithm called Ripple (lines 22–23 of Algorithm 1), (iii) prunes edges from $\mathcal{A}$ (line 24 of Algorithm 1), and (iv) updates the probability distribution (line 25 of Algorithm 1). Each of these key steps are discussed in the following subsections.

A. Sampling

The sampling procedure (Algorithm 2) uses the routine $\text{Sample}$ developed in Informed RRT* [18, Algorithm 2]. Informally speaking, that subroutine samples within a prolate hypersphere with the focal points being the start and goal configurations and the diameters defined by the cost. Any configurations out of this hypersphere cannot improve the current-best solution. In Fig. 1, this corresponds to sampling within the yellow ellipsoids as shown through the progressions.

In our sampling procedure, a fixed number of samples $n_s$ is added to $S'$ from $X_f$ in each iteration. First, an edge $e := (u, v)$ is drawn from the probability distribution over $\mathcal{A}$ (line 3 in Algorithm 2). Initially, until a feasible $S_T$ is obtained, the probability distribution is computed using the heuristic lower bounds (line 17 in Algorithm 1). Once a random sample $x_{\text{rand}}$ is obtained for the edge $(u, v)$ using $\text{Sample}$, it is added to $S'$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Algorithm 3 Ripple(S′)

// For any u ∈ V, let r_u denote the closest terminal to u in G. If u is a terminal, then r_u := u. The parent of any terminal is itself
1 while S′ ≠ ∅ do
2 s ← S′.pop()
3 V ← V ∪ {s}
4 E ← E ∪ {(s, u) : u ∈ N_V(s)}
5 parent(s), r_s ← NULL
6 n∗ ← arg min{(g(n) + cost(s, n)) : n ∈ N_V(s), r_n ≠ NULL}
7 if n∗ ≠ NULL then
8 E ← E ∪ {(s, n∗)}
9 g(s) ← g(n∗) + c(s, n∗)
10 r_s ← r_n
11 parent(s) ← n∗
12 Q.insert(s, g(s)) // Priority Queue
13 while Q ≠ ∅ do
14 u∗ ← Q.extractMin()
15 foreach n ∈ N_V(u∗) do
16 if g(u∗) + cost(u*, n) < g(n) then
17 Q.insert(n, g(n))
18 r_n ← r_u∗
19 E ← E \ {(n, parent(n)) \ (n, u∗)}
20 parent(n) ← u∗
21 else if r_n ≠ r_u∗ then
22 d ← g(n) + cost(n, u∗) + g(u∗) if
23 d < cost(y, r_u∗) then
24 cost(r_u∗, r_u) ← d
25 return

Algorithm 4 UpdateTree(S_T, A)

// Notation: Ω((u, v), S_T) is the cycle induced by adding edge (u, v) to S_T
1 if S_T is empty then
2 S_T = Kruskal(cost_T)
3 return S_T
4 for (u, v) ∈ A : (u, v) ≠ S_T do
5 P := {(u′, v′) ∈ Ω((u, v), S_T) \ (u, v)}
6 pathCost ← Σ{cost_T(u′, v′) : (u′, v′) ∈ P}
7 (u∗, v∗) = arg max{cost_T(u′, v′) : (u′, v′) ∈ P}
8 if cost_T(u∗, v∗) > cost_T(u, v) then
9 S_T ← S_T \ {(u∗, v∗)} ∪ {(u, v)}
10 else if pathCost < cost_T(u, v) then
11 cost_T(u, v) ← pathCost // Update non-MST edge with a cheaper path via MST
12 return S_T

Algorithm 5 PruneEdges(S_T, A)

1 for (u, v) ∈ A : (u, v) ≠ S_T do
2 (u∗, v∗) = arg max{cost_T(u′, v′) : (u′, v′) ∈ Ω((u, v), S_T) \ (u, v)}
3 if h(u, v) > cost_T(u∗, v∗) then
4 A ← A \ {(u, v)}
5 return A

B. Incremental Steiner Tree Algorithm

This procedure is accomplished by first finding lower-cost, feasible paths between the terminals through Ripple (Algorithm 3), and then updating the spanning tree S_T. In Ripple, each sample is processed individually until S′ becomes empty. Adding a new point s to the roadmap (consequently to G) can facilitate new paths between terminals. If s has a neighbor that is connected to a terminal (or is itself a terminal), then s and each of its neighbors are expanded in a Dijkstra-like fashion until the priority queue Q is empty (lines 13–21 of Algorithm 3). The variable g(u) keeps track of the shortest path from u to its closest terminal which is stored in r_u. The key part of the Ripple algorithm lies in finding new feasible paths between the terminals. This happens (line 22 of Algorithm 3) during the expansion of a node u* to its neighbor n when g(u∗) + cost(u∗, n) ≥ g(n) and r_u∗ ≠ r_u, i.e., n is closer to its root r_u than to r_u∗. In this case, a feasible path from r_u∗ to r_u has been discovered through the nodes u∗ and n. If the cost of this feasible path is lower than cost_T(r_u∗, r_u), then cost_T(r_u∗, r_u) is updated accordingly.

After cost_T(u, v) for any pair of terminals (u, v) is updated, it is relatively straightforward to check if any edge not in S_T should become part of S_T. First, we check if S_T is empty or not. If S_T is empty, then we just use Kruskal’s algorithm [29] to find a spanning tree (if it exists) for T (line 2 of Algorithm 4). If S_T is not empty, we appeal to the following well-known cycle property of minimum spanning trees to see if we should include (u, v) in S_T.

Cycle Property 1: Suppose that S_T is a MST in G_T and when a new edge (u, v) ≠ S_T is added to G_T, it forms a cycle C = Ω((u, v), S_T). Consider an edge (u∗, v∗) ∈ C other than (u, v) that has the maximum of all the edge costs in C. If cost_T(u∗, v∗) > cost_T(u, v), then S_T \ {(u∗, v∗)} ∪ {(u, v)} is an MST of the updated G_T.

Thus, in our algorithm, if cost_T(u∗, v∗) > cost_T(u, v), then edge (u∗, v∗) is deleted from S_T and edge (u, v) is added to S_T (lines 9 of Algorithm 4).

C. Pruning Edges From A

The pruning procedure (Algorithm 5) is similar to what we discussed in the previous tree update procedure except that we now appeal to the bounds on the cost of the edges. Consider an edge (u∗, v∗) ∈ Ω((u, v), S_T) other than (u, v) that has the maximum of all the edge costs in C. If cost_T(u∗, v∗) < h(u, v), and since h(u, v) ≤ cost_T(u, v), we have cost_T(u∗, v∗) < cost_T(u, v). Then, again applying the cycle property, edge (u, v) is pruned from A and is never considered by the algorithm henceforth (line 4 of Algorithm 5). This can be seen in Fig. 1 where every non-MST edge gets pruned because its lower bound (straight line distance in this case) is more than the current cost of each of the edges it encloses in its MST cycle.
D. Update Probability Distributions

Once a feasible $S_T$ is found, the probability distribution is updated (Algorithm 6) in each iteration to reflect the changes in the costs and $A$. We want to sample the region around an edge more if the uncertainty about the cost of the edge is higher. This is based on our assumption that the more we learn about the cost of an edge, the better we can decide on its inclusion in $S_T$. If an edge $(u, v)$ already belongs to $S_T$, we assign a sampling probability for $(u, v)$ proportional to the difference between the edge cost and its lower bound. On the other hand, if an edge $(u, v)$ does not belong to $S_T$, we first aim to include it in $S_T$ and later hope to drive its cost to its lower bound. Thus, we assign a sampling probability for $(u, v)$ proportional to the difference between the edge cost and the cost of its next largest cost edge in $\Omega((u, v), S_T)$. An illustration of this can be seen in Fig. 1: In the middle subfigure, the MST edge 2−5 has highest deviation from its lower bound (compared to other MST edges). Thus, we would like to sample this edge’s informed set more to reduce the uncertainty (deviation). This is reflected by the sampling probability which can be seen to be darker compared to rest. Consequently, in the next iteration, we significantly densify in its corresponding ellipsoid making the deviation from its lower bound approach 0.

E. Theoretical Properties

At any iteration of the algorithm, the algorithm maintains a Steiner tree $S_T$ which is formed by expanding the paths of a MST on the terminal graph $G_T$. For any given state of the graph $G$ that contains the terminals $T$ along with the sampled points so far, we argue that the tree returned by IST* is such an MST.

Theorem 1: IST* will return an MST over the shortest paths between terminals in $G$.

We show this theorem by arguing that the edge pruning from the active set $A$ of inter-terminal edges is correct and that the sampling procedure with updated probabilities coupled with the Ripple update will eventually find all relevant shortest paths. The former claim follows from the cycle property as discussed in the pruning step. To prove the latter claim, we use the following key lemma, that is proved in the Appendix.

Lemma 2: Suppose a sample point $s \in S'$ is processed by Ripple so that $r_s$ is a terminal $t$, then in $G$, $t$ is (one of) the closest terminal to $s$ among all terminals $T$.

In this way, Ripple maintains every sample point in the correct so-called “Voronoi” partition of the terminals (i.e. assigning it to its closest terminal). From this lemma, we see that the edges between terminals in $G_T$ whose costs are updated by Ripple (line 24, Algorithm 3) are precisely those where an edge between a node $u^*$ and its neighbor $n$ such that $r_n \neq r_n$ discovers a new cheaper path between $r_n$ and $r_n$. Thus, Ripple is able to find all shortest paths between pairs of terminals that are witnessed by a pair of neighboring boundary nodes. We can now use a result of [38] (Lemma in Section 2.2) that shows that this subgraph $G_T$ of the subset of shortest paths between terminals in $T$ that are witnessed by adjacent boundary nodes is sufficient to reconstruct an MST of $G$. This shows that $S_T$ constructed as a MST of $G_T$ is indeed an MST of the metric completion of $T$ using all the edges in $G$ as claimed.

Since the UpdateTree method in IST* always maintains a MST of the sampled graph $G$, in the limit with more sampling, the actual shortest paths corresponding to the true final MST will be updated to their correct lengths with diminishing error, and this MST will be output by IST*. Since the MST is a 2-approximation to the optimal Steiner tree in any graph and consequently, the MGPF problem [28], we get the following result.

Corollary 3: IST* outputs a 2-approximation to the MGPF problem asymptotically.

V. RESULTS

We compared the performance of IST* with the existing approaches, namely SFF* [24], Multi-T-RRT [5] and LazyTSP [8]. However, we found that these approaches timed out (took significantly greater time to find an initial solution) even in three-dimensional environments. Thus, for a more holistic evaluation of IST*, we came up with a baseline, as described in the Background and Preliminaries section. Specifically for the baseline, we densified a roadmap given by PRM* [26] for a fixed amount of time via uniform random sampling after which $S^*$ (specifically, $S^*$-BS variant) was run on the resulting roadmap to obtain an MST-based Steiner tree [3]. It was ensured that the combined time spent in growing the roadmap and running $S^*$ exhausted the input time limit.

The planners (IST* and the Baseline) were implemented in Python 3.7 using OMPL [45] v1.5.2 on a desktop computer running Ubuntu 20.04, with 32 GB of RAM and an Intel i7-8700k processor. Further implementation details are available in the Appendix. Our implementation of IST* and all the code for experiments is publicly available for reproducibility.\[8\] A node $u$ is said to be a boundary node if not all its neighbors have root $r_u$.\[9\] https://github.com/nikhilchandak/InformedSteinerTrees
Due to the absence of a standard suite of instances for benchmarking planners for MGPF, we use the environments commonly considered in motion planning and extend them to our setting by randomly generating valid goal configurations. Specifically, we test our planners in rigid body motion planning instances available in the OMPL app GUI and the environments considered by the authors of SFF* [9]. A problem instance is uniquely identified by an environment (with its specific robot model), number of terminals, and the computational time given for planning. For each environment, a varying number of terminals \( |T| = 10, 30, 50 \) was generated. The choice for the size of the problem instances was motivated by unmanned vehicle applications [39]. We now describe the testing environments and the corresponding results next.

A. Environments

As the environment models in OMPL App are designed for single source and destination motion planning (with no goals), many of them have disconnected regions making them unsuitable for benchmarking multi-goal path planning under random generation of goal points. Thus, we used the Home, Abstract and Twisty Cool environment which admit one connected SE(3) configuration space.

Home (Fig. 2a) and Abstract (Fig. 2b) are related to the classic Piano Mover’s problem [44], admitting several narrow passages offering several homotopy classes of solution paths. Meanwhile, Twisty Cool (Fig. 2c) represents a cubicle divided into two regions by a wall in between which has a narrow window connecting the two regions.

Apart from OMPL environments, we also considered 3D environments (6D configuration space) introduced by the authors of SFF*: specifically, Dense (Fig. 2d) and Triangles (Fig. 2e). Triangles has multiple triangular prisms of same size symmetrically placed as obstacles while Dense represents a city-like environment with various free-form building-like structures, placed irregularly, acting as obstacles.

The spatial size (volume) of each of the environments is shown in Table I.

B. Comparison With the State of the Art

Given the stochastic nature of sampling-based path planning methods, all the approaches considered for evaluation were ran 30+ times to obtain a distribution of the cost of the Steiner tree found by them. First, we note that IST* and the Baseline find a (initial) solution for all the problem instances considered within 5 minutes. Meanwhile, it was found that LazyTSP took more than 45 minutes across all the problem instances so we explicitly exclude it from further discussion. Next, Multi-R RRT, unlike SFF*, IST* and the Baseline, stops the moment it finds a solution and doesn’t improve further. Thus, we tried to find a rough average of the time taken by Multi-T-RRT to solve a problem instance. In the cases where it finished under an acceptable threshold (30 minutes\(^{10}\)), we recorded its duration and provided the same time to SFF*, IST* and the Baseline for fair comparison. Otherwise, they were given (the same) time based on the number of terminals and difficulty of planning in that environment. The exact time for each instance finally used is shown in Table I.

The results are shown as bar plots for each planner across environments in Fig. 3. To make them more insightful, we also calculate a lower bound on the cost of the MST (described in Appendix) in each problem instance and show it as a dashed red bar in the figures. It represents the best possible cost achievable for that instance.

We find that Multi-T-RRT was only able to find solutions in the Triangles environment which has relatively less volume compared to other environments. This suggests that Multi-T-RRT is only suitable for low-dimensional environments as it timed out even for 10 terminals in other environments. Furthermore, the solutions it found were 4 times worse than other planners which is not surprising given its nature of stopping at the first solution found.

SFF* is more promising that Multi-T-RRT, finding solutions across all environments for 10 terminals, across more than half of the environments for 30 terminals but only in Triangles.

---

\(^{10}\)Given both IST* and the Baseline finds a solution within 5 minutes, a significantly larger time period (30 minutes) was chosen to be the upper limit.
Fig. 3. Performance of the different approaches on the environments considered in Table I. The plots represent the mean (across 30 runs) with error bars (standard deviation). The number on top of a bar represents the percentage of failure while its absence indicates no failures. A cross ✗ represents 100% failure (timeout in all runs). Note that the y-axis is in log scale.

Fig. 4. A 2-dimensional illustration of the problems in real-vector spaces. The configuration space for both is a unit hypercube, i.e., each state space was bounded to the interval [0, 1], illustrated by the black bounding box. Solid regions represent invalid states. In Fig. 4(b), the hypercube obstacles are axis-aligned (specifically, 10 on each axis) and uniformly spread with $w = 0.075, d = 0.025$.

Fig. 4. A 2-dimensional illustration of the problems in real-vector spaces. The configuration space for both is a unit hypercube, i.e., each state space was bounded to the interval [0, 1], illustrated by the black bounding box. Solid regions represent invalid states. In Fig. 4(b), the hypercube obstacles are axis-aligned (specifically, 10 on each axis) and uniformly spread with $w = 0.075, d = 0.025$.

when given 50 terminals. Further, it excels in 10 terminals case, reaching the optimal solution in 4 out of the 5 environments consequently outperforming IST*. However, when considering more number of terminals, it either finds a poor solution, worse than our Baseline (30 terminals) or times out (50 terminals). SFF further has more variance and higher rate of failure. Meanwhile, our Baseline consistently finds good solutions across all instances and performs only marginally worse to IST* making it a competitive alternative for future evaluations.

Overall, we find SFF* is best suited for the cases when the number of terminals is low. In all other cases, IST* is superior to it, consistent, asymptotically optimal, and also performs good with low number of terminals. A final point of distinction we would like to emphasize is that the available implementation of Multi-T-RRT and SFF* only supports SE(3) (and its subspace), and thus, cannot be evaluated on environments like $\mathbb{R}^4$ and $\mathbb{R}^8$ presently. Further, SFF* was primarily designed for SE(3) and lower-dimensional environments given its dense space filling nature while Multi-T-RRT only tries to find a solution and never improves on it. IST* overcomes all these limitations and is scalable to higher dimensions as we show below.

C. Extensive Comparison With Baseline

Given the similar final solution cost of the Baseline with IST*, we now try to investigate in detail the difference in their performance, especially in higher dimensions. We let both the planners output the cost of the Steiner tree found over time. We ran each planner 50 times to obtain a distribution of solution costs. The mean solution cost with a 99% confidence interval is calculated from 50 trials and shown in Table II.

D. Real-Vector Space Problems

IST* and the Baseline were tested on two simulated problems in a unit hypercube with distinct obstacle configurations in $\mathbb{R}^4$ and $\mathbb{R}^8$ (Fig. 4). The collision detection resolution was set to $10^{-4}$ to make evaluating edge costs computationally expensive. The time given to each planner is available in Table V (in Appendix).

Center Obstacle (CO): This problem consisted of a big hypercube at the center (with volume $(0.9)^n$ in $\mathbb{R}^n$) (Fig. 4a).

https://github.com/ctu-mrs/space_filling_forest_star
TABLE II
COMPARISON OF IST* AGAINST THE BASELINE ACROSS 6 DIFFERENT ENVIRONMENTS – REAL VECTOR SPACES OVER UNIFORM HYPER RECTANGLES (UH) AND CENTER OBSTACLES (CO), AND SE(3) ENVIRONMENTS HOME AND ABSTRACT – OVER 10/30/50 TERMINALS. THE Y-AXIS REPRESENTS THE SOLUTION COST (I.E., STEINER TREE COST) WHILE THE X-AXIS IS THE TIME HORIZON (IN SECONDS). THE THIN SOLID LINE REPRESENTS THE MEAN SOLUTION COST WITH THE SHADED REGION BEING THE 99% CONFIDENCE INTERVAL ABOUT THIS MEAN.
In this environment, IST* started on par with the Baseline but significantly improved while the Baseline remained stuck in a sub-optimal solution.

Uniform Hypercubes (UH): This domain was filled with a regular pattern of axis-aligned hypercubes (identically distributed with uniform gaps) (Fig. 4b). For a real vector space of dimension \( n \), this environment contains \( 10^n \) obstacles with a total volume of 0.75\(^n\). In this environment, we see a dominant performance of IST* over the Baseline with up to 30% better solution cost.

**E. SE(3) Problems**

Among the OMPL environments, Home offers the highest scope for optimization among all the environments considered, with the final solution often being 50% better than the initial solution. In Home, Baseline performs competitively to IST*, and sometimes also finds better solutions initially. However, IST* converges to a better solution asymptotically in all the problem instances of Home. In Abstract, both Baseline and IST* find a good solution quickly, and as a result, the scope for optimization is not much. Still, IST* is able to improve very quickly, dominates throughout, and reaches the final solution cost of Baseline \( \approx 6 \) times faster.

**F. Impact of Ripple**

In each iteration of IST*, a batch of samples\(^{12}\) is added to the roadmap \( G \). As Ripple processes each sample incrementally and keeps rewiring a rooted shortest-path tree until convergence, it may seem better to simply add all the samples at once and then run S* on the updated roadmap to get \( S_T \). However, the roadmap grows significantly over time, making S* computationally expensive for repeated executions. Whereas, Ripple only has to process a small fraction of the roadmap for a fixed number of samples per batch. Note that Ripple uses the same amount of memory as S* given that the underlying graph is same for both. However, the time taken by each may vary, significantly.

We benchmarked Ripple with S* to investigate its effect on the performance of IST*. A pseudorandom generator was used for generating the same sample points for both. The problem instances considered for this comparison were same as before (Table V). However, due to space constraints, we only show the results (in Table III) for just 2 environments depicting extreme scenarios.\(^{13}\) In UH \( \mathbb{R}^6 \), both S* and Ripple perform similarly, with Ripple being marginally better in the instance with 50 terminals. In Home, Ripple not only finds a better final solution, but is also 2 – 3x faster than S* for the same solution cost consistently. In all other problem instances as well, these 2 scenarios were observed throughout: either Ripple’s performance was very similar to S* or it was significantly faster while converging to a better solution. In conclusion, while both S* and Ripple can lead to the optimal MST over \( G_T \), Ripple dominates S* across all instances considered.

**G. Comparing Path Costs**

While IST*’s primary output is a Steiner tree, we also evaluated the cost of the feasible path obtainable from IST* for MGPF (details provided in Appendix). Overall, it was seen to perform better than the Baseline in path computations too, an example of which is shown on two challenging problem instances in Fig. 5.

\(^{12}\) Note that the batch size is a hyperparameter and remains fixed throughout the execution.

\(^{13}\) The plot on rest of the problem instances with further discussion is available in the Appendix.
A critical phase in IST* and the Baseline, it adaptively limits the connections in $G$ as the number of samples increase by decreasing the radius $\rho$ (generally done to make the minimal number of connections required to ensure asymptotic optimality), as proposed in [26]

$$\rho(q) := \eta \left( 2 \left( 1 + \frac{1}{d} \right) \left( \frac{\lambda(X_f)}{\zeta_d} \right) \left( \frac{\log(q)}{q} \right) \right)^{\frac{1}{2}}$$

where $q$ is the number of points sampled in $X_f$ (ie., $|V|$), $d$ is the dimension of $X$, $\eta > 1$ is a tuning parameter, and $\lambda(X_f)$ is Lebesgue measure of the obstacle-free space and $\zeta_d$ is the volume of the unit ball in the $d$-dimensional Euclidean space.

C. Pruning

1) Lower Bound on Edges: A critical phase in IST* is pruning edges of $G_T$ which cannot be part of the MST in future. Success of this phase relies heavily on how close are the lower bounds of these edges to their optimal cost. Past research has been focused mostly on finding lower bounds for special instances of the general motion planning problem [2], [34]. A separate line of research recently has been focused on calculating the lower bound on the cost obtainable from the current set of samples [43]. However, these are not lower bound on the optimal path cost but only representative of the lower bound derived from the discrete approximation of the state space. Thus, due to lack of effective worst-case lower bound on the optimal path cost for the general motion planning

---

**TABLE IV**

| Environment | Terminals | 10         | 30         | 50         |
|-------------|-----------|------------|------------|------------|
| CO $\mathbb{R}^4$ | 91% | 98% | 98% |
| CO $\mathbb{R}^8$ | 64% | 84% | 90% |
| UI $\mathbb{R}^4$ | 92% | 97% | 98% |
| UI $\mathbb{R}^8$ | 37% | 79% | 81% |
| HOME        | 87% | 91% | 98% |
| ABSTRACT    | 97% | 98% | 99% |

**VI. CONCLUSION**

Inspired by the recent advancements in sampling-based methods for shortest path problems for single source and destination in continuous space and multi-goal path finding in discrete space, we provide a unifying framework to promote a novel line of research in MGPF. For the first time, to the best of the authors’ knowledge, the ideas of informed sampling have been fruitfully extended to the setting of planning for multiple goals. Our approach is decoupled in the sense that any further advancements in PRM* or informed sampling can be directly used to update the respective components of our proposed framework.

In the future, we plan to explore obtaining effective lower bounds on the shortest path cost between two terminals instead of using the Euclidean distance. Including heuristics in Ripple also seems to be a promising line of research.

**APPENDIX**

**A. Key Lemma on Ripple**

We will prove Lemma 3 here by induction. At the beginning of the algorithm, $G$ consists only of the terminals $t' \in T$ for which $r_{t'} = t'$ so the claim is trivially true. When a new sample point $s$ is processed, it is assigned to its neighbor $n$ with the smallest value of $g(n) + \text{cost}(s,n)$ (line 6 of Algorithm 3) via which it traces a shortest path to its closest terminal (by the definition of the $g$-values of its neighbors).

---

**TABLE V**

TIME (IN SECONDS) GIVEN TO IST* AND THE BASELINE AS INPUT IN DIFFERENT PROBLEM INSTANCES

| Environment | Terminals | 10   | 30   | 50   |
|-------------|-----------|------|------|------|
| CO $\mathbb{R}^4$ | 450s | 900s | 1350s |
| CO $\mathbb{R}^8$ | 900s | 1800s | 2700s |
| UI $\mathbb{R}^4$ | 600s | 1200s | 1800s |
| UI $\mathbb{R}^8$ | 1200s | 2400s | 3600s |
| HOME        | 600s | 1200s | 1800s |
| ABSTRACT    | 600s | 600s  | 600s  |

---

Fig. 5. Comparisons of the costs of the feasible MGPF solution found by the planners in the high-dimensional real environments with 50 terminals (over 50 runs).
problem, we use the Euclidean distance between two points in $X_f$ as the heuristic estimate.

2) Numerical Results: In Table IV, we show the impact of the pruning condition we have developed. Even though we have a very simple heuristic estimate as the lower bound, we are able to prune a significant number of edges. With more terminals, it is likely that edges in the MST of $G_T$ will be a straight-segments in $X_f$ (hence, the optimal path is quite close to Euclidean distance). This is turn should lead to more pruning which we observe as a trend in Table IV.

While the pruning statistics may look appealing, it is possible to generate instances where no edges would be pruned.  

\textit{a) Worst case:} Consider a problem instance with a star-shaped obstacle in the center with terminals at the concave openings between two sharp ends of the star obstacle. Further, the placement of terminals and the concave openings should be such that all terminals are near to each other while the optimal path is long and convoluted. For example, in Figure 6, lower bound between every edge is less than the optimal cost between any two distinct terminals which means no edge will be pruned in $G_T$.

\textbf{D. Lower Bound Computation}

A lower bound in the results can provide a more comprehensive and informative view when comparing the performance...
of different methods. Recall that a trivial lower bound for any edge between two nodes is the Euclidean distance between them. Thus, we can create a lower bound graph, where each edge between two nodes is the Euclidean distance and calculate the Minimum Spanning Tree on this graph to obtain a valid lower bound on the original problem. While this may seem to be a loose bound, many planners actually come fairly close to it (Fig. 3) establishing that lower bound was not far from the optimal solution.

E. Implementation Details

While we mention that a fixed number of samples is added to the roadmap in the pseudocode of IST', the implementation could not exactly follow this as such an operation is not allowed in the available module for PRM*. However, OMPL does support growing the roadmap for a fixed amount of time. Hence, we had a time-based implementation where in each iteration of IST', the roadmap was grown for the same amount of time. Consequently, the number of samples added per iteration was not fixed and varied depending upon the time spent in collision-checking for each new sample point. We let IST' and the Baseline run for all of the provided time.

Both the planners used the implementation of PRM* from OMPL (with default parameters). We also tried LazyPRM* [21] from OMPL but it performed worse than PRM*, contrary to our expectations. However, our approach (subsequently, the codebase of IST') is modular such that LazyPRM* or any other implementation of the underlying roadmap can be used to interface with our planner.

1) Path Cost Computations: To evaluate the cost of the feasible path obtainable from IST' for MGPF (results shown in Fig. 3), Baseline and IST' were given the same time as earlier (mentioned in Table III) to compute the Steiner Tree. This was followed by computing shortest path in the roadmap graph G for every pair of goal nodes to obtain a distance matrix which was passed to the Lin-Kernighan heuristic (LKH) [22], [32]. LKH was run for 100 iterations for both the Baseline and IST' to get the final path cost for MGPF.

F. Ripple: More Results and Discussion

We show the comparison of Ripple with S* on more environments in Table VI. In Abstract with 10 terminals, we see Ripple performing better throughout while in all other instances its performance is almost same as Baseline. In real-vector space instances, it was observed that size of the roadmap in the end was much smaller compared to instances like Home or Abstract. As the size of roadmap was tiny, so we couldn’t witness the benefits of Ripple. We believe this was a consequence of our implementation in Python.

Real environments like CO and HR (Fig. 2) were custom defined in Python so when the roadmap was grown for both IST' and the Baseline, OMPL’s planners in C++ had to interface with the collision checker of these environments in Python for each point sampled in the configuration space. This made the growth of PRM in these instances extremely slow due to the constant back and forth call between Python and C++. Thus, the size of the roadmap G in these environments was pretty small compared to Home or Abstract which are defined in OMPL App itself.

REFERENCES

[1] G. Best, J. Faigl, and R. Fitch, “Multi-robot path planning for budgeted active perception with self-organising maps,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Oct. 2016, pp. 3164–3171.
[2] J. Canny and J. Reif, “New lower bound techniques for robot motion planning problems,” in Proc. 28th Annu. Symp. Found. Comput. Sci. (SFCS), Oct. 1987, pp. 49–60.
[3] K. Chour, S. Rathinam, and R. Ravi, “S*: A heuristic information-based approximation framework for multi-goal path finding,” in Proc. Int. Conf. Automated Planning Scheduling, vol. 31, 2021, pp. 85–93.
[4] T. Decroos, P. De Causmaecker, and B. Demoen, “Solving Euclidean Steiner tree problems with multi swarm optimization,” in Proc. Companion Publication Annu. Conf. Genetic Evol. Comput., Jul. 2015, pp. 1379–1380.
[5] D. Devaurs, T. Siméon, and J. Cortés, “A multi-tree extension of the transition-based RRT: Application to ordering-and-pathfinding problems in continuous cost spaces,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Sep. 2014, pp. 3891–3896.
[6] S. Edelkamp and E. Plaku, “Multi-goal motion planning with physics-based game engines,” in Proc. IEEE Conf. Comput. Intell. Games, Aug. 2014, pp. 1–8.
[7] S. Edelkamp, B. C. Secim, and E. Plaku, “Surface inspection via hitting sets and multi-goal motion planning,” in Proc. Annu. Conf. Towards Auto. Robotic Syst. Cham, Switzerland: Springer, 2017, pp. 134–149.
[8] R. Englot and F. S. Hover, “Three-dimensional coverage planning for an underwater inspection robot,” Int. J. Robot. Res., vol. 32, nos. 9–10, pp. 1048–1073, Aug. 2013.
[9] J. Faigl, “On self-organizing map and rapidly-exploring random graph in multi-goal planning,” in Advances in Self-Organizing Maps and Learning Vector Quantization. Berlin, Germany: Springer, 2016, pp. 143–153.
[10] J. Faigl and G. A. Hollinger, “Unifying multi-goal path planning for autonomous data collection,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Sep. 2014, pp. 2937–2942.
[11] J. Faigl, M. Kulich, V. Vonásek, and L. Přeučil, “An application of the self-organizing map in the non-Euclidean traveling salesman problem,” Neurocomputing, vol. 74, no. 5, pp. 671–679, Feb. 2011.
[12] J. Faigl, P. Vána, and J. Deckeřová, “Fast heuristics for the 3-D multi-goal path planning based on the generalized traveling salesman problem with neighborhoods,” IEEE Robot. Autom. Lett., vol. 4, no. 3, pp. 2439–2446, Jul. 2019.
Annu. Rev. J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Informed sampling...

J. C. Fort, “Solving a combinatorial problem via self-organizing process: An application of the Kohonen algorithm to the traveling salesman problem,” Biol. Cybern., vol. 59, no. 1, pp. 33–40, Jun. 1988.

C. Friedrich, A. Csiszar, A. Lechler, and A. Verl, “Efficient task and path planning for maintenance automation using a robot system,” IEEE Trans. Autom. Sci. Eng., vol. 15, no. 3, pp. 1205–1215, Jul. 2018.

J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Informed sampling for asymptotically optimal path planning,” IEEE Trans. Robot., vol. 34, no. 4, pp. 966–984, Aug. 2018.

J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Sep. 2014, pp. 2997–3004.

J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Batch informed trees (BIT): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2015, pp. 3067–3074.

C. R. Garrett et al., “Integrated task and motion planning,” Annu. Rev. Control Robot. Auton. Syst., vol. 4, pp. 263–293, May 2021.

K. Hauser, “Lazy collision checking in asymptotically-optimal motion planning,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2015, pp. 2951–2957.

K. Helsgaun, “An effective implementation of the Lin–Kernighan traveling salesman heuristic,” Eur. J. Oper. Res., vol. 126, no. 1, pp. 106–130, Oct. 2000.

J. A. Hoogeveen, “Analysis of Christofides’ heuristic: Some paths are more difficult than cycles,” Oper. Res. Lett., vol. 10, no. 5, pp. 291–295, Jul. 1991.

J. Janoš, V. Vonásek, and R. Penicka, “Multi-goal path planning using multiple random trees,” IEEE Robot. Autom. Lett., vol. 6, no. 2, pp. 4201–4208, Apr. 2021.

M. Jünger, G. Reinelt, and G. Rinaldi, “The traveling salesman problem,” in Handbooks in Operations Research and Management Science, vol. 7, Amsterdam, The Netherlands: Elsevier, 1995, pp. 225–330.

S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” Int. J. Robot. Res., vol. 30, no. 7, pp. 846–894, Jun. 2011.

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmap planners for high-dimensional configuration spaces,” IEEE Trans. Robot. Autom., vol. 12, no. 4, pp. 566–580, Aug. 1996.

L. Kou, G. Markowsky, and L. Berman, “A fast algorithm for Steiner trees,” Acta Inf., vol. 15, no. 2, pp. 141–145, 1981.

J. B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” Proc. Amer. Math. Soc., vol. 7, no. 1, pp. 48–50, 1956.

J. J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in Proc. ICRA. Millennium Conf., IEEE Int. Conf. Robot. Automation, Symposia, Apr. 2000, pp. 995–1001.

C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, “Obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 27, no. 4, pp. 643–653, Apr. 2008.

L. Liu, H. Wei, H. Wang, H. Li, and H. Wang, “Integrated task allocation and path coordination for large-scale robot networks with uncertainties,” IEEE Trans. Autom. Sci. Eng., vol. 19, no. 4, pp. 2750–2761, Oct. 2022.

V. J. Lumelsky and A. A. Stepanov, “Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape,” Algorithmica, vol. 2, nos. 1–4, pp. 403–430, Nov. 1987.

D. G. Macharet and M. F. Campos, “A survey on routing problems and robotic systems,” Robotica, vol. 36, no. 12, pp. 1781–1803, 2018.

J. McMahon and E. Plak, “Autonomous underwater vehicle mine countermeasures mission planning via the physical traveling salesman problem,” in Proc. OCEANS, Oct. 2015, pp. 1–5.

J. McMahon and E. Plak, “Dynamic multi-goal motion planning with range constraints for autonomous underwater vehicles following surface vehicles,” in Proc. IEEE 17th Int. Conf. Autom. Sci. Eng. (CASE), Aug. 2021, pp. 704–711.

K. Melihorn, “A faster approximation algorithm for the Steiner problem in graphs,” Inf. Process. Lett., vol. 27, no. 3, pp. 125–128, Mar. 1988.

P. Oberlin, S. Rathinam, and S. Darbha, “Today’s traveling salesman problem,” IEEE Robot. Autom. Mag., vol. 17, no. 4, pp. 70–77, Dec. 2010.

A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch, “Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey,” Networks, vol. 72, no. 4, pp. 411–458, Dec. 2018.

S. Rathinam, R. Sengupta, and S. Darbha, “A resource allocation algorithm for multivehicle systems with nonholonomic constraints,” IEEE Trans. Autom. Sci. Eng., vol. 4, no. 1, pp. 98–104, Jan. 2007.

M. Saha, T. Roughgarden, J.-C. Latombe, and G. Sánchez-Ante, “Planning tours of robotic arms among partitioned goals,” Int. J. Robot. Res., vol. 25, no. 3, pp. 207–223, Mar. 2006.

O. Salzman and D. Halperin, “Asymptotically-optimal motion planning using lower bounds on cost,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2015, pp. 4167–4172.

J. T. Schwartz and M. Sharir, “On the ‘piano movers’ problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers,” Commun. Pure Appl. Math., vol. 36, no. 3, pp. 345–398, 1983.

I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” IEEE Robot. Autom. Mag., vol. 19, no. 4, pp. 72–82, Dec. 2012. [Online]. Available: https://ompl.kavrakilab.org

F. Suárez-Ruiz, T. S. Lembono, and Q.-C. Pham, “RoboTSP—A fast solution to the robotic task sequencing problem,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2018, pp. 1611–1616.

P. Vanek, J. Faigl, and D. Masri, “Multi-goal trajectory planning with motion primitives for hexapod walking robot,” in Proc. 11th Int. Conf. Robot. Autom. (ICRA), May 2018, pp. 1611–1616.

V. Vonásek and R. Penicka, “Space-filling forest for multi-goal path planning,” in Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFa), Sep. 2019, pp. 1587–1590.

Y. Warsame, S. Edelkamp, and E. Plak, “Energy-aware multi-goal motion planning guided by Monte Carlo search,” in Proc. IEEE 16th Int. Conf. Autom. Sci. Eng. (CASE), Aug. 2020, pp. 335–342.

A.-C. Závoianu et al., “Multi-objective optimal design of obstacle-avoiding two-dimensional Steiner trees with application to ascent assembly engineering,” J. Mech. Des., vol. 140, no. 6, Jun. 2018, Art. no. 061401.