2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

A fast two-stage approach for multi-goal path planning in a fruit tree

Werner Kroneman[1], João Valente[2], and A. Frank van der Stappen[1,3]

[1]Department of Engineering, University College Roosevelt, Middelburg, The Netherlands, {w.kroneman,a.f.vanderstappen}@ucr.nl
[2]Information Technology Group, Wageningen University, Wageningen, The Netherlands, joao.valente@wur.nl
[3]Department of Information and Computing Sciences, Utrecht University, The Netherlands, a.f.vanderstappen@uu.nl

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10160281

***Abstract*—We consider the problem of planning the motion of a drone equipped with a robotic arm, tasked with bringing its end-effector up to many (150+) targets in a fruit tree; to inspect every piece of fruit, for example. The task is complicated by the intersection of a version of Neighborhood TSP (to find an optimal order and a pose to visit every target), and a robotic motion-planning problem through a planning space that features numerous cavities and narrow passages that confuse common techniques. In this contribution, we present a framework that decomposes the problem into two stages: planning approach paths for every target, and quickly planning between the start points of those approach paths. Then, we compare our approach by simulation to a more straightforward method based on multi-query planning, showing that our approach outperforms it in both time and solution cost.**

***Index Terms*—motion planning, multi-goal, drone, robotics, task-sequencing**

I. Introduction

The potential of robotics in agriculture to reduce labor costs and enable new, currently prohibitively expensive tasks is great, despite the numerous remaining challenges. In fact, autonomous robots are already being adopted for use in various tasks across the industry; drones (multirotor UAVs) form an important subset of these robots, which tend to focus on a variety of tasks, such as monitoring and inspection [5]. In fruit tree farming, for instance, we might wish to perform an up-close inspection of a large number of targets (e.g. pieces of fruit or blossoms) using a drone. Such a task would require a drone equipped with an arm to bring a sensor or a tool of some sort up-close to every target, while avoiding the tree's numerous branches (see Figure 1).

It is a complex problem for various reasons. First, there are the common combinatorial reasons: what is the optimal order (out of $n!$ possibilities for n targets) to visit every target and from which pose to do so? This is a Traveling Salesperson Problem with Neighborhoods (TSPN) [12, 24], a generalization of the Traveling Salesperson Problem, the canonical example of an NP-Hard problem: good approximating techniques exist [6, 1, 21], but an efficient, optimal solution remains elusive despite extensive study. Unfortunately, for n targets, existing TSP(N) solvers expect the cost of moving between all $O(n^2)$ pairs of targets to be known a priori; this requirement is hard to meet, as three- or higher-dimensional planning spaces often lack the structural properties that facilitate the efficient computation of optimal shortest paths [2, 4].

This research was partially funded by Interreg Europe as part of the CIMAT agricultural robotics project.

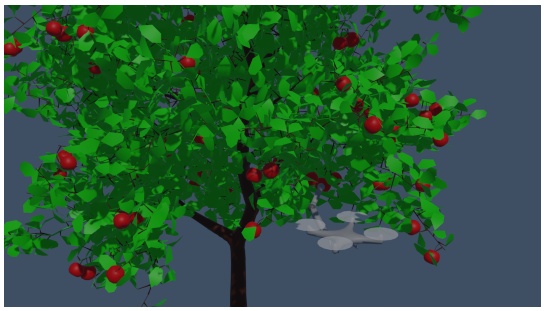

Fig. 1. Rendering of our planning scene: the drone must bring its end-effector to every apple (red) in the apple tree. Due to the large number of branches and complex structure of the tree, navigation between apples is difficult. Note that, while leaves are ignored as collision objects, the (many) twigs that leaves are attached to *do* count.

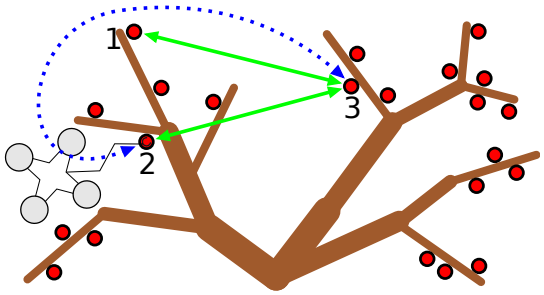

Fig. 2. Schematic, 2D representation of a planning scene illustrating the difficulty in estimating the cost of moving between two targets within the branches of a fruit tree: note how, in this scene for instance, targets 1 and 2 are at a similar distance to target 3 as-the-crow-flies, yet the distance that must be traveled between targets 2 and 3 (dotted blue line) is far greater than between 1 and 3 in reality. Of course, this diagram does not capture the extra complexity from working in 3D.

Second, the complex branch structure of the tree, full of cavities and potential narrow passages, presents significant practical complications to planning paths through the tree even when start- and end-points are known. Distance heuristics that ignore obstacles, such as the Euclidean distance between

targets or some distance metric between robot configurations, are difficult to apply as fruit may appear close together while actually being positioned on opposite sides of a tree branch, while other fruit may be far apart while growing within the same cavity within the tree, making movement between them unexpectedly easy. See Figure 2 for an illustration.

We study the intersections of these problems in this paper. The planning environment that we consider in this paper consists of a 3D model of a fruit tree; examples are shown in Figures 1 and 5. To focus on the combinatorial challenges, we assume that such a model and a set of target regions is available at the time of planning and that this model does not change during the operational phase of the drone.

A. Related work

Motion planning is a well-known and highly studied problem domain. Famous algorithms like Dijkstra's [8] and A* [13] efficiently give exact, optimal solutions in discrete planning spaces, while methods based on roadmaps and cell decomposition do the same in some continuous planning spaces, though often at a cost exponential in the degrees of freedom [18]. In these spaces, sampling-based methods such as the Probabilistic Roadmap (PRM) [16] and the Rapidly-Exploring Random Tree (RRT) [17] certainly proved their worth. Many such methods are *probabilistically complete*: given enough time and memory, the probability of finding a path (if it exists) converges to 1. Variants of these methods such as RRT* [15] and PRM* [15] are *asymptotically optimal*: they give a further guarantee that the length of the path they find will asymptotically converge to the optimal.

In multi-goal settings, Vicencio *et al.* [24] and Faigl *et al.* and [9] approximately solve the TSPN through genetic algorithms and various heuristics, but they do not consider obstacles. A pair of publications by Janoš, Vonásek and Pěnička [14, 25] grow RRT-like trees from a set of individual goal configurations in 2D, connecting them into a roadmap when they touch.

For robotic arms, Wurll *et al.* [26] show how to progressively build a cost matrix between pairs of individual goals configurations, RoboTSP[22] presents a method based on task-space distance heuristics for redundant arms (and thus multiple configurations per goal) for an industrial hole-drilling application on uncluttered planar surfaces, and Edan *et al.* consider the TSP for a mobile fruit-picking robot in an obstacle-free setting.

B. Our contribution

In this paper, our contributions include:

- A decomposition of the motion planning problem into local approach planning steps and fast global planning steps, reminiscent of retraction-based methods [19, 20].
- A method to solve the approach-planning step, in a way that injects limited collision information into the global path-planning step.
- A method for near-instant motion planning in the global stage and how to apply it to formulate a cost matrix for a TSP solver.
- A comparison by simulation of our approach with a more straightforward algorithm based on the multi-query capabilities of PRM* [15].

In the remainder of this paper, we shall formalize our problem in Section II, and then explain our problem decomposition in Section III. For comparison, we present a more straightforward alternative in Section IV, which we compare by simulation to our method in Section V. Finally, we conclude our paper in Section VI.

II. Problem formulation

First, we formally define some concepts and notation in order to formulate our problem in an abstract, formal manner. In Section II-B, we then describe how the fruit tree inspection problem is defined in these terms.

A. Abstract formulation and definitions

A *configuration* $c \in \mathcal{C}$ (where $\mathcal{C}$ is the *configuration space*) determines all degrees of freedom of a robot, fully defining the pose and joint angles. A configuration $c \in \mathcal{C}$ is *free* if the robot does not collide with any obstacle in the pose defined by c; we denote the *free configuration space*, the set of all free configurations, as $\mathcal{C}_{\text{free}}$. For a given configuration space, we assume a distance metric $d : \mathcal{C}^2 \to \mathbb{R}^+ \cup \{0\}$ exists between pairs of configurations.

A *path* is a continuous curve through the configuration space, formally defined as a mapping with type $[0, 1] \to \mathcal{C}$. A path Π is *collision-free* if for all $t \in [0, 1]$, $\Pi(t) \in \mathcal{C}_{\text{free}}$.

The length of a path Π is defined as

$$|\Pi| = \sup_{k\in\mathbb{N}, 0=t_0<t_1<\cdots<t_k=1} \sum_{i=0}^{k-1} d\left(\Pi(t_i), \Pi(t_{i+1})\right),$$

based on the definition from [15], though generalized from Euclidean vectors to arbitrary $\mathcal{C}$. Intuitively, this is analogous to an integral of the form $\int_0^1 |\nabla\Pi(t)|dt$ if Π were differentiable in t.

Finally, we define a *goal region* $\mathcal{G}_i \in \mathscr{G}$ as a given set of free configurations ($\mathcal{G}_i \subseteq \mathcal{C}_{\text{free}}$) of which one must be visited, where $\mathscr{G}$ is the set of given goal regions. A goal (region) $\mathcal{G}_i$ is *reachable* from a given initial configuration $c_0 \in \mathcal{C}_{\text{free}}$ if there exists a collision-free path Π where $\Pi(0) = c_0$ and $\Pi(1) \in \mathcal{G}_i$. A path Π is said to *visit* a goal (region) $\mathcal{G}_i$ if there exists a $t \in [0, 1]$ such that $\Pi(t) \in \mathcal{G}_i$. Furthermore, we assume that a procedure GoalSample($\mathcal{G}_i$) exists that returns a configuration uniformly at random from $\mathcal{G}_i$.

Formally, we define our problem as follows: "For a given set of goal regions $\mathscr{G}$ and an initial configuration $c_0 \in \mathcal{C}_{\text{free}}$, find a short collision-free path Π, with $\Pi(0) = c_0$, that visits all reachable goal regions $\mathcal{G}_i \in \mathscr{G}$."

B. Specifics of our planning scene

To make the above definitions more concrete, we define how they apply to our planning scene depicted in Figure 1.

The robot in question consists of a flying (floating) base, assumed to always fly upright, fitted with a 3-link robotic arm connected with revolute joints. Hence, the configuration space $\mathcal{C}$ is defined as the set of all tuples of the form $(\vec{t}, r, \theta_0, \theta_1, \theta_2)$, where $\vec{t} \in \mathbb{R}^3$ is the translation of the flying base, $r \in \mathbb{H}$[1] with $|r| = 1$ is the rotation of the flying base (assumed to always be upright), and $\theta_0, \theta_1, \theta_2 \in [-\pi, \pi]$ are the rotation angles of every arm joint in radians. Moreover, let $e(c)$ be the position of the end effector for some $c \in \mathcal{C}$.

The n goal regions are defined by target points (e.g. corresponding to fruit) in the tree: for a given point $t_i \in \mathbb{R}^3$ ($i \in \{0 \dots n-1\}$), the goal region $\mathcal{G}_i \in \mathscr{G}$ is the set of all $c \in \mathcal{C}_{\text{free}}$ where the Euclidean distance $|e(c), t_i|$ is less than a given threshold $\epsilon \geqslant 0$. To implement GoalSample, we simply generate a configuration $c \in \mathcal{C}$ uniformly at random, then apply a translation such that $|e(c), t| \leqslant \epsilon$, then reject samples where $c \notin \mathcal{C}_{\text{free}}$

The distance $d(c_i, c_j)$ for any given $c_i = (\vec{t}_i, r_i, \theta_{0,i}, \theta_{1,i}, \theta_{2,i}) \in \mathcal{C}$ and $c_j = (\vec{t}_j, r_j, \theta_{0,j}, \theta_{1,j}, \theta_{2,j}) \in \mathcal{C}$ is defined as

$$d(c_i, c_j) = |\vec{t}_i - \vec{t}_j| + \arccos|r_i \cdot r_j| + \sum_{k \in \{0,1,2\}} |\theta_{k,i} - \theta_{k,j}|,$$

which corresponds to the standard definition used in MoveIt [7] with all joints given equal weight.

Collision checking is used to check whether a configuration $c \in \mathcal{C}$ is in free space $\mathcal{C}_{\text{free}}$: $c \in \mathcal{C}_{\text{free}}$ if and only if no part of the robot collides with a wooden part of the tree (brown in Figure 1) or with the xy-plane (the ground, with z being the vertical axis). As a general rule of thumb, collision checking is the most expensive part of a motion planning algorithm and should therefore be minimized when possible. We assume that leaves are pushed aside when collided with, and thus ignore collisions with them for simplicity.

III. Shell/Approach Decomposition

In this section, we describe our main contribution: the decomposition of our robotic TSPN problem into a local *approach planning* stage and a *fast global planning* stage. We first present our approach in an abstract manner, where we convey our main intuition and prerequisites, and then show how it can be implemented for our fruit tree scene.

A. Abstract approach

The intuition behind our approach is to designate a set of *shell configurations* $\mathcal{C}_{\text{shell}} \subseteq \mathcal{C}$ that form a boundary around some cluttered portion of $\mathcal{C}$, structured such that finding a short collision-free path through $\mathcal{C}_{\text{shell}}$ (a *shell path*) between two configurations $c_i, c_j \in \mathcal{C}_{\text{shell}}$ is easy enough to do $O(n^2)$ times (for n goal regions) and then planning *approach paths* between $\mathcal{C}_{\text{shell}}$ and configurations outside this subspace. An analogy can be made to retraction-based methods [19, 20], although in our case retraction takes place onto a two-dimensional surface rather than a one-dimensional network of arcs.

[1] $\mathbb{H}$ is the set of all quaternions

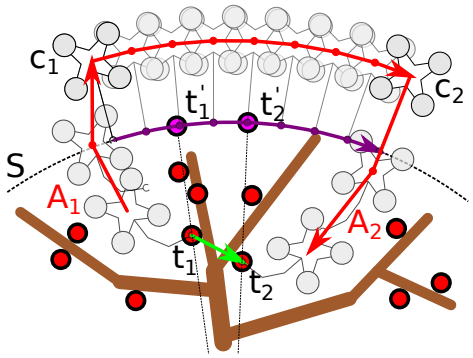

Fig. 3. Schematized 2D illustration of a goal-to-goal path between two targets $t_1, t_2 \in T$, created by composition of two approach paths and a path along a spherical shell S around the obstacles. The robot first retreats from t_1 via A_1, then travels along S (path through $\mathcal{C}_{\text{shell}}$), then approaches t_2 via A_2. Note how c_1, c_2 are not at the projections of t'_1, t'_2, respectively; a planner may pick any configuration in $\mathcal{C}_{\text{shell}}$ as the start of the approach path, which effectively injects some limited collision-related information into the global planning stage.

First, in the *local stage*, we construct a set of *approach paths* A by planning a short collision-free path A_i for every $\mathcal{G}_i \in \mathscr{G}$, such that $A_i(0) \in \mathcal{C}_{\text{shell}}$ and $A_i(1) \in \mathcal{G}_i$; if no path is found, $\mathcal{G}_i$ is marked *unreachable*. Similarly, we plan a short collision-free *initial approach path* I such that $I(0) = c_0$ and $I(1) \in \mathcal{C}_{\text{shell}}$; we assume $\mathcal{C}_{\text{shell}}$ is reachable from c_0.

Now, to compute a *goal-to-goal* path $\Pi_{i,j}$ between two goal regions $\mathcal{G}_i, \mathcal{G}_j \in \mathscr{G}$ with respective approach paths $A_i, A_j \in A$ (such that $\Pi_{i,j}(0) \in \mathcal{G}_i$ and $\Pi_{i,j}(1) \in \mathcal{G}_j$), we compose paths such that the robot first backs away from $\mathcal{G}_i$ by following A_i in reverse, then travels along the shell path $B_{i,j}$ between $A_i(0)$ and $A_j(0)$, and finally approaches $\mathcal{G}_j$ by following A_j; such a movement is illustrated in Figure 3. As moving all the way out to $\mathcal{C}_{\text{shell}}$ then back into the cluttered region can be inefficient for targets in close proximity, we further recommend locally optimizing the resulting path. Formally, we thus define

$$\Pi_{i,j} = \textsc{LOpt}\big(\text{Rev}(A_i) * B_{i,j} * A_j\big)$$

where Rev reverses a path, $*$ is a path concatenation operator, and LOPT is a standard local optimization procedure, such as PathSimplifier in OMPL[23], which shortens a path by repeated short-cutting and perturbations until convergence; start and end configurations are preserved.

Then, in the *global stage*, to determine the order in which to visit every approach path $A_i \in A$ (and every reachable $G_i \in \mathscr{G}$ by proxy), we call out to an approximate TSP solver to find a permutation A' of A that approximately minimizes

$$d_{\text{shell}}\left(I(1), A'_0(0)\right) + \sum_{i=0}^{i=|A'|-2} d_{\text{shell}}\left(A'_i(0), A'_{i+1}(0)\right)$$

where for any $c_i, c_j \in \mathcal{C}_{\text{shell}}$, $d_{\text{shell}}(c_i, c_j)$ is the length of the shell path between c_i, c_j. In our implementation we use

the routing problem solver in Google OR-Tools [21]; instead of considering all $O(n!)$ possibilities, it works by computing an initial guess, then progressively refining it in an anytime fashion; we used default settings. Note how, by approximating the length of the true shortest path between goal regions by d_{shell}, which is fast to compute by definition, we avoid the problem that computing all $O(n^2)$ costs by just planning paths is prohibitively expensive. This does come with the assumption that the shell paths will have a mostly dominant impact on the length of actual paths between goals.

The final path Π is then defined as

$$\Pi = \text{LOPT}(I * B_0 * A'_0) * \mathop{\ast}_{i=0}^{i=|A'|-2} \Pi_{i,i+1}$$

where B_0 is the shell path between $I(1)$ and $A'_0(0)$, and $*$ concatenates a sequence of paths.

B. Implementation

In the previous section, we treated $\mathcal{C}_{\text{shell}}$, as well as planning paths both from and to $\mathcal{C}_{\text{shell}}$ as well as through $\mathcal{C}_{\text{shell}}$, in an abstract manner. In this section, we shall implement these notions for our fruit tree planning problem.

We define $\mathcal{C}_{\text{shell}}$ first. Let S be a minimum enclosing sphere [10] around all branches and leaves of the tree, ignoring the trunk. For any $p \in \mathbb{R}^3$ let $\text{Proj}_S(p)$ be the projection of p on the surface of S, and let $\text{SConf}(p)$ be the configuration $c = (\vec{t}, r, \theta_0, \theta_1, \theta_2) \in \mathcal{C}$ with $\theta_0 = \theta_1 = \theta_2 = 0$ (arm straight out), r chosen such that the ray from $\vec{t}$ through $e(c)$ intersects the vertical line through the center of S (robot facing the tree), and $\vec{t}$ is chosen such that $e(c) = \text{Proj}_S(p)$. Finally, let $\mathcal{C}_{\text{shell}}$ be the set of all $\text{SConf}(p)$ for every $p \in \mathbb{R}^3$ on the surface of S.

1) Approach paths: Given a goal region $\mathcal{G}_i \subseteq \mathcal{G}$ we must plan a short collision-free path A_i between $\mathcal{G}_i$ and $\mathcal{C}_{\text{shell}}$.

As a starting point, let $c_i = \text{SConf}(t_i)$ where t_i is the target associated with $\mathcal{G}_i$; c_i is thus a shell configuration close to $\mathcal{G}_i$, serving as an initial guess for $A_i(0)$.

First, check if the straight-line motion from c_i to c'_i is a collision-free path; otherwise, we use any sampling-based motion planner.

In our case, we use PRM* [15] to plan from c_i to any configuration in $\mathcal{G}_i$. As a stopping criterion, we define time limits $t_{\max}$ and t_{patience} with $t_{\max} > t_{\text{patience}} > 0$; we stop iterating if $t_{\max}$ is exceeded and no solution has been found, or if t_{patience} has elapsed since the last improved solution has been found; the intuition behind t_{patience} is to allow PRM* to discover easy optimizations while avoiding diminishing returns from longer runtimes; due to the large number of targets, we consider it acceptable to skip a small number where determining reachability takes excessively long.

Inspired by [11], we restrict our sampler by repeatedly generating a motion m between c_i and a sample $g \in \mathcal{G}_i$, taking a randomly-interpolated configuration $s_i = \text{Lerp}(c_i, g, u)$ where u is a random value between 0 and 1, then for $r > 0$ picked from a half-normal distribution around 0, we sample a configuration s'_i such that $d(s_i, s'_i) \leqslant r$. Finally, we always apply local optimization as a post-processing step.

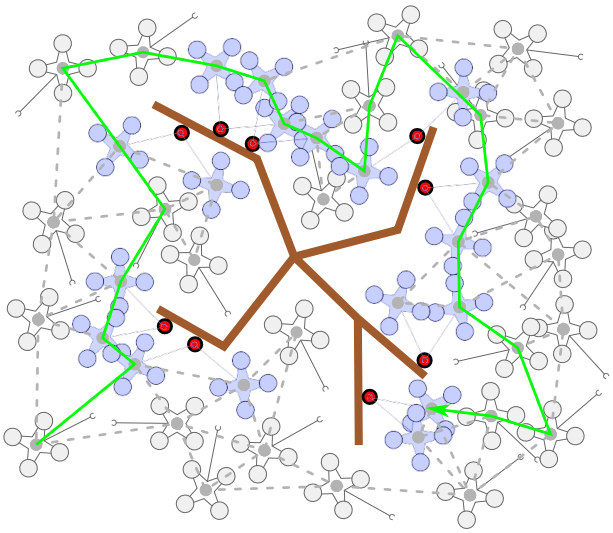

Fig. 4. Schematized 2D illustration of the TSP-over-PRM* algorithm: the PRM*[15] algorithm takes random configuration samples, and attempts to make straight-line connections between nearby samples. For every goal region, k samples are taken and added to the graph in a similar manner, effectively reducing our TSP(N) in the full configuration space to a finite, discrete version of the problem. The final path is highlighted in green. Note that in reality, the workspace is in 3D, bringing with it significant increases in complexity.

2) Shell paths: Finally, to construct the *shell path* $B_{i,j}$ between any given $c_i, c_j \in \mathcal{C}_{\text{shell}}$, let g be the geodesic on S between $e(c_i), e(c_j)$, and for every $t \in [0,1]$, define $B_{i,j}(t) = \text{SConf}(p_t)$ where p_t is a point interpolated along g by parameter $t \in [0,1]$. A special case arises where Π collides with the trunk of the tree. It is possible to avoid this case by deforming g such that we have $\text{SConf}(p_t) \in \mathcal{C}_{\text{free}}$ for all $p_t \in g$. In practice, however, we have found that simply allowing the collision, then running OMPL's [23] check-and-repair procedure works well enough to handle this case.

IV. Monolithic approach: TSP(N) over PRM*

In this section, to compare it with our decomposition-based approach, we define an algorithm that relies on the multi-query capabilities of PRM* [15], which works by first building up a probabilistic roadmap graph of the whole space that can then be queried relatively inexpensively many times. Thus, this algorithm treats $\mathcal{C}$ as a monolith, rather than distinguishing a special subspace like $\mathcal{C}_{\text{shell}}$.

A. Algorithm description

The *TSP-over-PRM** algorithm, illustrated by Figure 4, is based on this idea: it first lets PRM* build a roadmap graph. Then, it takes a configurable k goal samples per goal region, and connects these to the graph as if they were samples drawn by the original algorithm. Finally, $O(n^2k^2)$ paths are queried between every pair of goal samples using A*, and OR-Tools [21] is used to solve the TSP(N) within the graph.

First, we follow PRM* [15] to build a roadmap graph $R = (V, E)$, as implemented in [23]. We do so by repeatedly taking a sample $c \in \mathcal{C}$; the translation component is sampled

uniformly from $[-r, r] \times [-r, r] \times [0, r]$ where $r > 0$ is a parameter chosen such that enough margin is left on all sides of the tree model to capture the connectivity of that space; the z component is limited to $z \geqslant 0$ to avoid sampling configurations underneath the ground plane. The configuration c is then collision-checked; if $c \in \mathcal{C}_{\text{free}}$, a set of $k' \in \mathbb{N}$ nearest neighbors to c are looked up (see [15] for how to pick k' to guarantee asymptotic optimality); for every c_i (with $i \in \{0 \dots k' - 1\}$), the motion (c, c_i) is collision checked and added to E if collision-free. Samples are taken until a configurable time t_R has passed. Then, for every $\mathcal{G}_i \in \mathscr{G}$, a configurable $k \in \mathbb{N}$ configuration samples are taken from $\mathcal{G}_i$, and added to R using the same procedure as before. Let S be the set of all sampled goal configurations. Finally, add the initial configuration c_0 to the graph like the other nodes.

Now that our roadmap R is available, we first compute a shortest path in R from c_0 to every $c_i \in S$ using A* (as is typical for path queries in PRM*); drop c_i from S if no path is found. If for some $\mathcal{G}_i \in \mathscr{G}$, all samples are dropped from S, $\mathcal{G}_i$ is marked as *unreachable*. Then, for every unordered pair $c_i, c_j \in S$, compute a shortest path from c_i to c_j in R using A*. Every time a path Π is found with A*, store the length of Π in a distance lookup table $D : (\{c_0\} \cup S)^2 \to \mathbb{R}^+ \cup \{0\}$, which can then be queried to find the cost to move between pairs of configurations from $(\{c_0\} \cup S)$.

Note the omission of an LOPT step here: for n goals, D contains $O(n^2 \cdot k^2)$ entries; running LOPT for each would be prohibitively expensive since it relies on collision checks. Quite paradoxically, increasing t_R leads to a bigger R, which in turn increases the cost of running A*, and subsequently of building D. This is the fundamental issue that our Shell/Approach framework aims to solve.

Similar to what we did in Section III-A, we call upon an existing TSP(N) solver to find a permutation S' of a subset of S where for every reachable $\mathcal{G}_i \in \mathscr{G}$, there is a $c_j \in S'$ such that $c_j \in \mathcal{G}_i$, while minimizing

$$D(c_0, S'_0) + \sum_{i=0}^{i=|S|-2} D(S'_i, S_{i+1}).$$

We again use Google OR-Tools [21] for this purpose, as the "disjunction" feature allows us to specify the necessary constraints.

Finally, call upon A* again to reconstruct an initial path I from c_0 to S, and a path $\Pi_{i,i+i}$ for every consecutive pair c_i, c_{i+1} in S'. Finally, we define

$$\Pi = \text{LOPT}(I) * \mathop{\circledast}_{i=0}^{i=|A'|-2} \text{LOPT}(\Pi_{i,i+i})$$

as our output path.

V. Comparison in Simulation

In this section, we experimentally compare our decomposition-based method (see Section III) to the TSP-over-PRM* method (see Section IV) by running them on a simulated version of the fruit tree inspection problem presented in the introduction.

Fig. 5. Rendering of the tree tree models used in the simulation whose results are depicted in Figure 6. The model on the left is the same as depicted in Figure 1.

A. Set-up

To test our algorithms, we used MoveIt [7] to model our scene and check for collisions; we continuous collision detection to prevent tunneling through thin structures. A total of 3 fruit tree models (see Figure 5) were used, each for a different planning scene. While more models were available, results did not vary much. Each consists of three components: the wooden parts (used for collision detection), leaves (ignored, as these are soft obstacles), and fruit (the robot must visit these); the trees contained 163, 300 and 205 pieces of fruit, respectively. By visual observation, in the first tree, fruit is distributed in small clusters, whereas the second and third trees have a more uniform distribution of fruit. The relative scale of our robot to the tree models is shown in 1.

The wooden parts had triangle meshes with varying complexity (38264, 106286, and 12838 triangles) due to thin twigs and curved tree parts. The HACD algorithm in Bullet Extras [3] was used for convex decomposition. All simulations were run on a Lenovo IdeaCenter 5 14ACN6 with 16 GiB RAM, which features an AMD® Ryzen 7 5700g APU.

We generated a set of 120 planning problems as follows: 10 times for each fruit tree model, for every $n \in \{10, 50, 100, 150\}$, we pick n fruit as goals, and a random starting configuration c_0 outside the tree. For each problem, we run the Shell/Approach-based planner and TSP-over-PRM* planner with different parameter values. Specifically:

- The Shell/Approach Planner (Section III) with a spherical shell, with $t_{\max}$ picked from 400ms, 500ms, and 1000ms and t_{patience} set at 25 ms.
- The TSP-over-PRM* planner (Section IV), with a warm-up planning time t_R picked from $\{1s, 2s, 5s, 10s, 15s, 20s\}$ seconds and $k \in \{2, \dots, 10\}$.

In total, this makes for 57 algorithm-parameter combinations, most of which are for TSP-over-PRM* since that is the point of comparison that we wish to improve upon with any set of parameters for the spherical Shell/Approach planner. Each of these is run once for every planning problem (for a total

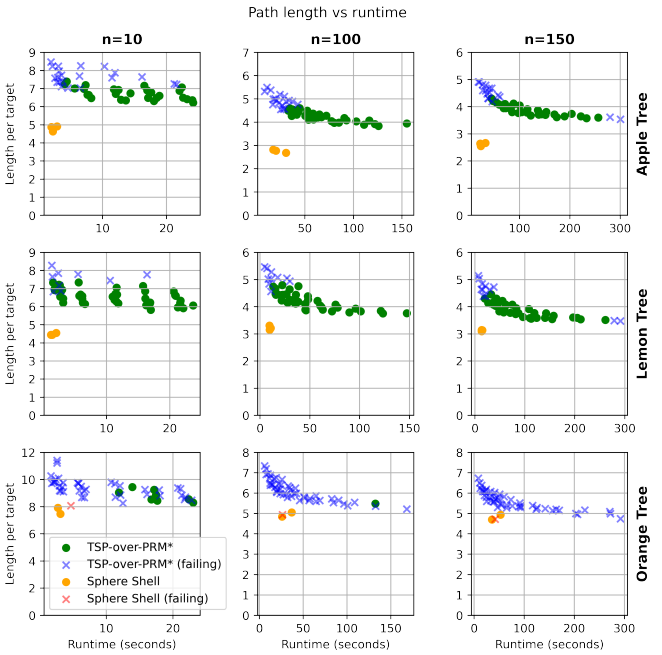

Fig. 6. Mean path length ($\hat{l}$) per visited goal, versus mean required CPU time ($\hat{t}$) in seconds to compute the path. Every row corresponds to a different tree model, every column to a different target set size n; $n = 50$ was removed to save space, but shows identical trends. Every • marker represents a different algorithm/parameter combination; a × marker is used for planners marked as *failing* due to skipping excessively many targets.

of 8040 runs), with 16 runs performed in parallel (matching the thread count of our APU); for each run, we record the following metrics:

- l: Total path length divided by the number of targets visited.
- p: Proportion of total targets visited.
- $t_{\max}$: Total time taken to run the algorithm.

B. Analysis

Once the l, p, t of reach run are recorded, runs are grouped per scene and total number of targets and we compute the means $\hat{l}, \hat{p}, \hat{t}$ within each group. There is thus one tuple of $\hat{l}, \hat{p}, \hat{t}$ for each combination of planner and parameters, number of apples, and tree model.

As is unavoidable with sampling-based techniques, a planner may erroneously mark a target as unreachable. Therefore, we mark a planner as *failing* if $\hat{p} < \hat{p}_{\max} - 0.05$, where $\hat{p}_{\max}$ is the highest $\hat{p}$ for a given tree model, effectively quantifying the difficulty of reaching targets in that scene. Only one of the three scenes had $\hat{p}_{\max} < 1$, making two rejection thresholds of $\hat{p} < 0.95$, and one of $\hat{p} < 0.82$; one scene was clearly more difficult. Error margins in $\hat{l}, \hat{p}, \hat{t}$ were negligible.

The resulting data is plotted in Figure 6. Observing the plot, the Shell/Approach Planner is effectively dominant in performance across scenarios, though margins vary; these are notably tightest on the third tree, which also had a significantly lower $\hat{p}_{\max}$, although it should be noted that TSP-over-PRM* failed far more frequently than ours in this case.

Together, our experiments show that our decomposition-based approach (with spherical shell) significantly outperformed TSP-over-PRM* in terms of both path length per target $\hat{l}$ and running time $\hat{t}$. In fact, our approach often runs in an order of magnitude less time for lower $\hat{l}$. Curiously, a failing planner (which skips targets) does not achieve lower path lengths, improving the success rate of TSP-over-PRM* is therefore unlikely to improve $\hat{l}$.

Indeed, the $O(n^2k^2)$-time computation of the distance matrix formed the worst bottleneck for the TSP-over-PRM* planner. In relative terms, t_R of only a few seconds appeared best, with an increase in k having a stronger effect on quality - at a severe computational cost.

In terms of our approach, we were also quite surprised at the effectiveness of simply trying the straight-line motion first; this succeeded in about half of all cases, replacing an expensive path-planning operation with a single motion collision check, while improving path quality. Apparently, fruit is quite-well reachable from the outside of the tree in our models, which we only realized when our approach was apparently able to take advantage of it so well. Also, while we chose S as the minimum enclosing sphere, results barely changed when we inflated the radius, or moved the center of S, indicating a fair amount of robustness to the choice of S.

VI. Conclusions and Future Work

In this paper, we contributed a general framework for the decomposition of the multi-goal motion-planning problem by designating a subspace $\mathcal{C}_{\text{shell}} \subseteq \mathcal{C}$, then solving two subproblems: how to plan from $\mathcal{C}_{\text{shell}}$ to every goal, and how to plan between the starting points of the approach paths within $\mathcal{C}_{\text{shell}}$, exploiting structural properties of $\mathcal{C}_{\text{shell}}$ to do this quickly and efficiently. Then, in a simulation, we compared a practical implementation of this framework to the more straightforward approach based on the multi-query planner PRM*, with our planner showing superior performance.

However, we believe that this kind of breakdown of the problem shows potential beyond simply improved performance compared to monolithic approaches. For example, the spherical shell in Section III-B was chosen as a simple proof of concept; more complex definitions based on a convex hull or α-shape may yield significantly better performance. Also, with the difficulty of multiple targets removed, the approach-planning stage could be examined more; while our current method works well, it feels somewhat ad-hoc and could be replaced by an algorithm with better theoretical properties and performance.

On top of improvements to the implementation, we would like to explore the power of the problem decomposition in lifting some of the assumptions made about the environment. For instance, this separation should prove useful in the face information that is discovered as the drone flies, or of more dynamic environments involving trees swaying in windy weather conditions or rotor downdrafts. Also, while the algorithm is described and tested for a drone, it should work for any robot for which $\mathcal{C}_{\text{shell}}$ can be implemented.

REFERENCES

[1] Sanjeev Arora. "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems". In: *Journal of the ACM (JACM)* 45.5 (1998), pp. 753–782.

[2] Mark de Berg et al. *Computational Geometry*. Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-77974-2. URL: https://doi.org/10.1007/978-3-540-77974-2.

[3] *Bullet Physics SDK*. original-date: 2011-04-12T18:45:08Z. July 11, 2022. URL: https://github.com/bulletphysics/bullet3 (visited on 07/11/2022).

[4] John F. Canny. *The Complexity of Robot Motion Planning*. Cambridge, MA, USA: MIT Press, 1988. ISBN: 0262031361.

[5] Jaime del Cerro et al. "Unmanned Aerial Vehicles in Agriculture: A Survey". In: *Agronomy* 11.2 (2021). ISSN: 2073-4395. DOI: 10.3390/agronomy11020203. URL: https://www.mdpi.com/2073-4395/11/2/203.

[6] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[7] T. Coleman David. *Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study*. en. 2014. DOI: 10.6092/JOSER_2014_05_01_P3. URL: https://aisberg.unibg.it//handle/10446/87657.

[8] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[9] Jan Faigl, Petr Váňa, and Jindřiška Deckerová. "Fast Heuristics for the 3-D Multi-Goal Path Planning Based on the Generalized Traveling Salesman Problem With Neighborhoods". In: *IEEE Robotics and Automation Letters* 4 (2019), pp. 2439–2446.

[10] Kaspar Fischer, Bernd Gärtner, and Martin Kutz. "Fast smallest-enclosing-ball computation in high dimensions". In: *Proc. 11th European Symposium on Algorithms (ESA*. SpringerVerlag, 2003, pp. 630–641.

[11] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

[12] Iacopo Gentilini, François Margot, and Kenji Shimada. "The travelling salesman problem with neighbourhoods: MINLP solution". In: *Optimization Methods and Software* 28.2 (2013), pp. 364–378. DOI: 10.1080/10556788.2011.648932. URL: https://doi.org/10.1080/10556788.2011.648932.

[13] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[14] Jaroslav Janoš, Vojtěch Vonásek, and Robert Pěnička. "Multi-Goal Path Planning Using Multiple Random Trees". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 4201–4208. DOI: 10.1109/LRA.2021.3068679.

[15] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761. eprint: https://doi.org/10.1177/0278364911406761. URL: https://doi.org/10.1177/0278364911406761.

[16] Lydia E Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[17] Steven M LaValle et al. "Rapidly-exploring random trees: A new tool for path planning". In: (1998).

[18] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, May 2006. DOI: 10.1017/cbo9780511546877. URL: https://doi.org/10.1017/cbo9780511546877.

[19] Colm ó'Dúnlaing, Micha Sharir, and Chee K. Yap. "Retraction: A New Approach to Motion-Planning". In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC '83. New York, NY, USA: Association for Computing Machinery, 1983, pp. 207–220. ISBN: 0897910990. DOI: 10.1145/800061.808750. URL: https://doi.org/10.1145/800061.808750.

[20] Colm Ó'Dúnlaing and Chee K Yap. "A "retraction" method for planning the motion of a disc". In: *Journal of Algorithms* 6.1 (1985), pp. 104–111. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-6774(85)90021-5. URL: https://www.sciencedirect.com/science/article/pii/0196677485900215.

[21] Laurent Perron and Vincent Furnon. *OR-Tools*. Version 7.2. Google, July 19, 2019. URL: https://developers.google.com/optimization/.

[22] Francisco Suárez-Ruiz, Teguh Santoso Lembono, and Quang-Cuong Pham. "RoboTSP – A Fast Solution to the Robotic Task Sequencing Problem". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 1611–1616. DOI: 10.1109/ICRA.2018.8460581.

[23] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). https://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[24] Kevin Vicencio, Brian Davis, and Iacopo Gentilini. "Multi-goal path planning based on the generalized Traveling Salesman Problem with neighborhoods". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2985–2990. DOI: 10.1109/IROS.2014.6942974.

[25] Vojtěch Vonásek and Robert Pěnička. “Space-filling forest for multi-goal path planning”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2019, pp. 1587–1590. DOI: 10.1109/ETFA.2019.8869521.

[26] Christian Wurll, Dominik Henrich, and Heinz Wörn. “Multi-Goal Path Planning for Industrial Robots”. In: (1999). URL: http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-9637.