Optimizing Motion-Planning Problem Setup via Bounded Evaluation with Application to Following Surgical Trajectories

Sherdil Niyaz[1], Alan Kuntz[2], Oren Salzman[3], Ron Alterovitz[2], and Siddhartha S. Srinivasa[1]

***Abstract*— A motion-planning problem's setup can drastically affect the quality of solutions returned by the planner. In this work we consider optimizing these setups, with a focus on doing so in a computationally-efficient fashion. Our approach interleaves optimization with motion planning, which allows us to consider the actual motions required of the robot. Similar prior work has treated the planner as a black box: our key insight is that opening this box in a simple-yet-effective manner enables a more efficient approach, by allowing us to bound the work done by the planner to optimizer-relevant computations. Finally, we apply our approach to a surgically-relevant motion-planning task, where our experiments validate our approach by more-efficiently optimizing the fixed insertion pose of a surgical robot.**

I. INTRODUCTION

In this work, we approach optimizing the setups of motion-planning problems to maximize the utility of a given task. We define the "setup" of a planning problem as any parameters we have control over that must be set before the planner is invoked. This includes, for example, the kinematic design of the robot. Another example of problem setup is the fixed insertion pose of a surgical robot tasked with following a surgeon-provided path, where a poor insertion pose can severely hinder the robot's ability to do so (Fig. 1).

This example demonstrates how a problem's setup can dramatically affect the quality of solutions generated by a motion planner. However, selecting the best problem setup from a set of candidates is challenging. The motions required to complete a given task are naturally complex, as they must both avoid collisions in constrained areas and minimize some objective. This makes it difficult to evaluate the cost of motions required by a problem setup, and thus the quality of the setup, without the use of a motion planner itself.

We therefore apply an approach that interleaves optimization with motion planning to evaluate the quality of candidate problem setups. While we are not the first to do so [3], [15], previous approaches treat the planner as a black box. Our key insight is that opening this black box enables a more computationally-efficient approach—namely by using the optimizer to restrict work done by the planner to only optimization-relevant computations. We present a simple-yet-effective implementation of this idea that resembles branch and bound [16].

[1]Paul G. Allen School of Computer Science and Engineering, University of Washington. {sniyaz, siddh}@cs.washington.edu

[2]Department of Computer Science, University of North Carolina at Chapel Hill. {adkuntz, ron}@cs.cs.unc.edu

[3]The Robotics Institute, Carnegie Mellon University School of Computer Science. osalzman@andrew.cmu.edu

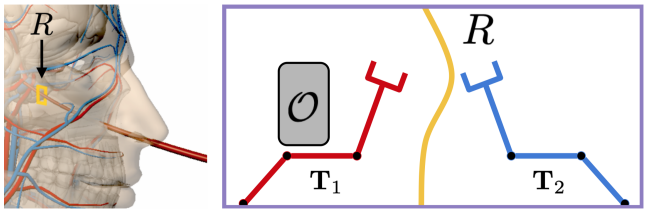

Fig. 1. **Left:** A concentric tube robot deployed with a fixed insertion pose through the sinus. A path R to cut using the robot's end-effector, or tip, is shown in yellow. **Right:** The insertion pose $\mathbf{T}_2$ on the right is preferable. It enables the robot (depicted here as a planar manipulator) to follow the path R without colliding with obstacle $\mathcal{O}$.

Specifically, we use the optimizer to derive some bound $\mathcal{B}$ for each candidate setup evaluated by the motion planner, where any setup with cost greater than $\mathcal{B}$ will not affect the optimization process. Fittingly, many planners are able to derive a lower bound $\mathcal{L}$ on the cost of their final solution before fully computing it. Should $\mathcal{L}$ exceed $\mathcal{B}$, the planner can abort given that additional work would not affect the optimization. We also demonstrate how a planner can be modified to maximize this approach's efficiency gains.

Our approach can be used to optimize the setups of various motion-planning problems, ranging from the placements of robots on a factory floor to the design space of a reconfigurable robot. In this paper we explore the surgical applications that motivated this work. Our surgical robot, a Concentric Tube Robot (CTR), enables new minimally-invasive surgeries in constrained areas due to its dexterity and small diameter [10]. Often these surgeries require the robot to follow with its tip some path R dictated by the surgeon, for example to cut a window in the skull during pituitary gland surgery (Fig. 1). With this in mind, we developed the Nearest-Neighbor Fréchet (NNF) planner in [19] that computes a CTR motion plan closely following such a path.

A key weakness of this planner is that it fails to consider the insertion pose $\mathbf{T}$ of the CTR, instead committing to one arbitrarily set. Unfortunately, the CTR's unintuitive kinematics make it difficult for a human to perfectly discern the reachable workspace given an insertion, likely making the initial insertion pose sub-optimal. Our experiments support this by optimizing $\mathbf{T}$, while also applying our approach to reduce convergence time by approximately $2\times$.

We begin with our approach to interleaving optimization with motion planning, which more-efficiently uses the planner to evaluate candidate problem setups (Sec. III). We then apply this approach to our specific surgical task (Sec. IV) and modify our NNF planner to maximize its computational efficiency (Sec. V). Finally, we discuss results in Sec. VI.

II. RELATED WORK

Optimizing the insertion pose of a CTR is similar to the problem of base placement for a mobile manipulator. Though approaches such as inverse reachability analysis [18], [24] have been applied to this problem, they do not account for the motions required by the specific *task* at hand. This is critical in our domain: while an entire path R may be reachable in the absence of obstacles, the interaction of the CTR's kinematics with the highly-constrained environments of surgery drastically affects the planner's ability to follow paths [19]. This induces a complex mapping from $\mathbf{T}$ to solution cost, requiring us to consider the actual motions required of the robot given $\mathbf{T}$.

Our optimization of the CTR's insertion pose is also similar to optimizing the placement of surgical ports, devices that provide entry vectors for drugs and medical tools. Feng et al. [8] optimize port placements for robotically-assisted cholecystectomy, while Hayashi et al. [12] do so for laparoscopic gastrectomy. While these approaches reason spatially about the relationship between port placement and reachable regions of the anatomy, they do not account for the series of motions required during their respective procedures.

Yet another similar problem involves motion planning with steerable needles, where the needle's insertion into the patient can drastically affect the set of locations reachable by its tip [2], [25]. However, a distinction must be drawn between CTRs and steerable needles. A CTR is endowed with holonomic kinematics similar to a serial-link manipulator, where each DOF of the robot can be controlled independently [22]. By contrast, a steerable needle is a nonholonomic system more similar in control to a wheeled vehicle than a traditional manipulator.

Some prior CTR-specific works have focused on optimizing the kinematic design of the robot to avoid obstacles while reaching a set of target points [4] or maximizing coverage of a volume [5]. Much like inverse reachability analysis, these geometric approaches consider only the ability to reach placements of the robot's tip, rather than the motions required to complete a task. They thus refrain from using a motion planner in the optimization process, which hinders their applicability to highly-constrained tasks. This is especially true in our domain, where the CTR is required to follow an entire *path* with its tip rather than reaching disjoint points.

The prior works most similar to our approach come from Baykal et al. [3] and Kuntz et al. [15]. These approaches use Adaptive Simulated Annealing (ASA) [17] to optimize the kinematic designs of surgical robots, together with a motion planner to evaluate the quality of candidate designs. In doing so, they account for the series of motions needed to complete their respective tasks. The former work seeks to optimize the design of piece-wise cylindrical robots in order to reach a set of target points. The latter aims to optimize the design of a parallel, needle-diameter surgical robot for the purpose of inspection planning [1]. In addition to having different motion-planning tasks from our work, these approaches do not use the optimizer to bound the work done by the planner.

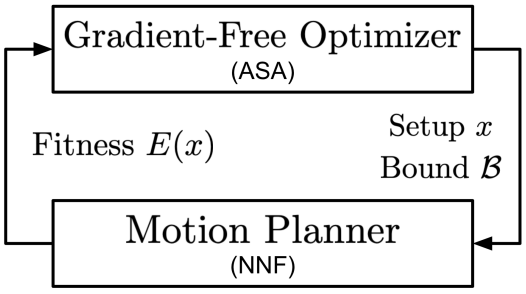

Fig. 2. An overview of our approach. In addition to sending a problem setup x to the planner for evaluation, the optimizer also sends a bound $\mathcal{B}$. Once this cost is exceeded, further evaluation by the planner will not change the steps taken by the optimizer. We note that in our *specific* application, ASA is used as the optimizer and NNF as the planner.

III. EFFICIENT SETUP EVALUATION

In our approach of interleaving optimization and motion planning, we focus on the use of gradient-free optimizers [17], [21]. We do so because it is unclear how to derive the closed-form gradient of solution cost with respect to problem-setup parameters for many motion planners, including our NNF planner.

Almost all gradient-free optimizers operate by sampling a state x, evaluating its fitness (or quality) via some function $E(x)$, and then using this value in a comparison that informs the optimizer's actions. In our case x is a candidate problem setup and $E(x)$ is the cost returned by invoking a motion planner given that problem setup. With our specific surgical application, this would then make x a candidate insertion pose $\mathbf{T}$ and $E(x)$ the corresponding cost returned by NNF. As we will demonstrate, this cost captures the deviation of the CTR's tip from the dictated path.

We focus our analysis in this section primarily on fitness evaluations and their subsequent comparisons. Specifically, we consider Adaptive Simulated Annealing [17] (ASA), an asymptotically-optimal gradient-free optimizer that makes minimal assumptions about the topology of the space. (We briefly discuss other optimizers at the end of this section.) At each iteration, ASA samples $u \in [0, 1]$ and a new problem setup x, then computes the setup's fitness $E(x)$. It "accepts" the new setup if:

$$\exp(-(E(x) - e')/T) > u. \tag{1}$$

Here, e' is the fitness of the most recently accepted setup and T is the temperature, a parameter that slowly decays as ASA converges. While ASA may accept an inferior setup with some probability in order to escape local minima, the likelihood that it will accept x decreases as $E(x)$ increases. That is, the worse a setup, the less likely the optimizer is to utilize it. This is characteristic of many gradient-free optimizers. In the case of ASA, we can show from (1) that no setup will be accepted with:

$$E(x) \geq e' - (T \cdot \ln(u)). \tag{2}$$

We note that the value of this bound, which we denote $\mathcal{B}$, changes each iteration of ASA as a function of u, T, and e'.

Such a bound is useful given that many motion planners are able to lower-bound the cost of their final solution *before* they have finished computing it. Consider, for example, a search-based planner that finds the shortest path on a graph $\mathcal{G}$ using the algorithm Dijkstra [7]. At each iteration the algorithm removes (or "expands") a node from its priority queue, the distance (or "cost-to-come") value of which lower-bounds the cost of the final path.

In our approach, this property can be used to derive a straightforward abort strategy similar to branch and bound [16]. In addition to passing a problem setup x to the planner for evaluation, ASA also computes and passes the current bound $\mathcal{B}$ from (2). As the planner works to generate a solution and its fitness $E(x)$, it also updates a lower bound $\mathcal{L}$ on the solution's final cost. Should $\mathcal{L}$ exceed $\mathcal{B}$, the candidate setup x is guaranteed not be to accepted. The planner should then abort immediately to prevent unnecessary computation (Fig. 2).

Another Optimizer: While we have performed this analysis for ASA, we note that similar bounds can be derived for many other gradient-free optimizers, making our abort strategy a general one. For example, consider cross-entropy minimization (CEM) [21], which on each iteration i) samples n problem setups from distribution $\mathcal{M}$, ii) evaluates $E(x_i)$ for each sampled x_i, and iii) picks the k best setups and refits $\mathcal{M}$ to them. The bound $\mathcal{B}$ when evaluating some $E(x_i)$ is the fitness of the kth best setup evaluated so far on that iteration, given that a sampled setup with worse fitness is guaranteed not to affect $\mathcal{M}$.

IV. SURGICAL APPLICATION

In the previous section we laid out a general approach for efficiently interleaving optimization with motion planning. We now apply this approach to our specific problem domain: following a path R specified by a human using the tip of a highly-dexterous surgical tool known as a Concentric Tube Robot [10], or CTR (Fig. 3). We begin with key definitions and then detail our planner for this task, NNF, which we proceed to use as the bottom half of Fig. 2 to optimize the insertion pose $\mathbf{T}$ of the robot.

A. Fundamental Definitions

We take as input a surgeon-specified reference path R. We represent this path as an ordered set of waypoints $r_0, \ldots, r_n$ in task-space, defined as the $\mathbb{R}^3$ position of the CTR's tip. Formally, R is encoded as a one-dimensional graph (V_R, E_R) in which directed edges connect subsequent waypoints (Fig. 4). We note that while our task-space definition is adequate for cutting with heat or a laser, our approach can trivially be used with other task-space definitions that account for the tip's orientation, such as $SE(3)$.

We wish to generate a path in the configuration-space $\mathcal{X}$ (the space of all possible configurations of the robot) that "closely" follows R with the tip. Each configuration q_i is a d-dimensional point uniquely defining the robot's shape, and thus must encode the translation and rotation of each

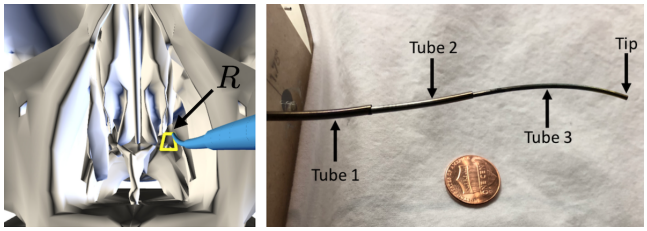

Fig. 3. **Left:** A CTR (blue) follows a reference path R (yellow) with its tip to cut a window through the skull in simulation. **Right:** A three-tube CTR alongside a coin for scale. Because each tube rotates and translates, this particular CTR has 6-DOF.

of its tubes (Fig. 3). Therefore, for a CTR composed of t tubes $\mathcal{X} \subseteq \mathcal{SO}(2)^t \times \mathbb{R}^t$.

We use the forward kinematics ($\text{FK}_\mathbf{T}$) operator to map points and paths in $\mathcal{X}$ to points and paths in task-space, and the inverse kinematics ($\text{IK}_\mathbf{T}$) operator to map a point in task-space to one or more points, or "solutions", in $\mathcal{X}$. Thus:

$$\begin{aligned} &\text{FK}_\mathbf{T}(q_i) : \mathcal{SO}(2)^t \times \mathbb{R}^t \longmapsto \mathbb{R}^3 \\ &\text{IK}_\mathbf{T}(r_i) : \mathbb{R}^3 \longmapsto \mathcal{SO}(2)^t \times \mathbb{R}^t. \end{aligned} \tag{3}$$

We denote the set of collision-free paths in $\mathcal{X}$ as $\Gamma_\mathbf{T}$, and note that all of $\text{FK}_\mathbf{T}$, $\text{IK}_\mathbf{T}$, and $\Gamma_\mathbf{T}$ depend on the insertion pose $\mathbf{T}$ of the CTR.

To quantify the notion of how "closely" the tip's path follows R, we use a well-studied measure of deviation or error between two paths known as the Fréchet distance [26]. This metric can be explained intuitively via an analogy, wherein a dog on a leash traverses one path with speed parameterization α as its owner traverses the other with parameterization β. (We note that α and β are independent.) Here, the Fréchet distance is the shortest leash length required for the dog and its owner to stay connected, assuming the two are moving optimally. The metric therefore respects the "flow" of the two paths, allowing a user to specify tasks such as tissue manipulation where the direction of motion matters. This makes it a natural error metric in our domain.

B. Algorithmic Background

Given these definitions and a preset insertion pose $\mathbf{T}$, our original NNF planner [19] minimizes:

$$\arg\min_{\gamma \in \Gamma_\mathbf{T}} \mathcal{F}(\text{FK}_\mathbf{T}(\gamma), R) \tag{4}$$

where $\mathcal{F}$ is the discrete Fréchet distance. We use this approximation, where the "leash" between the two paths is computed only for a discrete set of points, because computing the continuous Fréchet is computationally challenging [23].

The NNF planner approaches this objective by building a graph $N_k = (V_{N_k}, E_{N_k})$ in $\mathcal{X}$ (Fig. 4). To do so, n $\text{IK}_\mathbf{T}$ solutions are randomly sampled for points on R, and each is connected to its k nearest-neighbors in $\mathcal{X}$. Rather than connecting two samples q_1 and q_2 directly with an edge, we add a path of j configurations interpolated between them. N_k also contains start and goal nodes q_s and q_g connected to $\text{IK}_\mathbf{T}$ solutions for r_0 and r_n, the first and final waypoints on R, respectively. Note that the edges of N_k are directed and constrained to monotonically follow R.

Algorithm 1 Fitness of Candidate Insertion Pose $\mathbf{T}$
Input: Pose $\mathbf{T}$, Bound $\mathcal{B}$

$N_k \leftarrow$ SAMPLE_NN_GRAPH$(R, \mathbf{T})$
$\Phi \leftarrow$ INIT_IMPLICIT_TENSOR(N_k, R)
LPA $\leftarrow$ INIT_LPA_STAR$(\Phi, \mathcal{B})$ ▷ Also sets abort bound $\mathcal{B}$.
while LPA.goal_reachable() **do**
  $\tau \leftarrow$ LPA.min_bottleneck_path()
  if τ was aborted **then** ▷ Bound $\mathcal{B}$ exceeded.
    break
  for $e \in \tau$ **do**
    collision $\leftarrow$ VALIDATE(e) ▷ Lazy collision-checks.
    if collision **then**
      Φ.set_edge_weight(e, ∞)
      LPA.update_edge_weight(e, ∞)
  $\mathcal{F} \leftarrow$ BOTTLENECK_COST(τ)
  if $\mathcal{F} < \infty$ **then**
    return $\mathcal{F}$ ▷ Path τ is fully valid. Return fitness.
return ∞

Once N_k has been sampled, NNF applies the approach of Holladay et al. [13] to find a path $\gamma \in N_k$ minimizing $\mathcal{F}(\text{FK}_{\mathbf{T}}(\gamma), R)$. In doing so, it constructs a directed tensor-product graph $\Phi = (V_\Phi, E_\Phi)$. V_Φ contains one node (r, q) for every possible combination of a node $r \in V_R$ and a node $q \in V_{N_k}$. Each node (r_x, q_x) has an out-edge to:

$$\begin{aligned}(r_y, q_x) \text{ iff } r_x \to r_y \in E_R \\ (r_x, q_y) \text{ iff } q_x \to q_y \in E_{N_k} \\ (r_y, q_y) \text{ iff } r_x \to r_y \in E_R \wedge q_x \to q_y \in E_{N_k}.\end{aligned} \tag{5}$$

Less formally, (r_x, q_x) has out-edges to all nodes that represent taking a "step" on the reference path R from r_x, on the sampled graph N_k from q_x, or both. Finally, the weight of an edge $(r_x, q_x) \to (r_y, q_y)$ is:

$$\max\{\mathcal{D}(r_x,\ \text{FK}_{\mathbf{T}}(q_x)),\ \mathcal{D}(r_y,\ \text{FK}_{\mathbf{T}}(q_y))\} \tag{6}$$

where $\mathcal{D}$ is the Euclidean distance.

After constructing Φ, NNF searches it for the minimal-bottleneck path τ from $(r_0,\ q_s)$ to $(r_n,\ q_g)$, i.e. the path with the lowest possible maximum edge weight. We note that this path can be computed simply by applying a modified shortest-path algorithm, such as Dijkstra, that uses a max operator instead of addition to combine the distance value of a node with the weight of an outgoing edge. By extracting q_i from each node $(r_i, q_i) \in \tau$, we can then recover the path γ on N_k minimizing $\mathcal{F}(\text{FK}_{\mathbf{T}}(\gamma), R)$, where the bottleneck cost of τ (the weight of its heaviest edge) is the objective value [13]. The interpolated nodes of N_k better enable this value to approximate the true, continuous Fréchet [13].

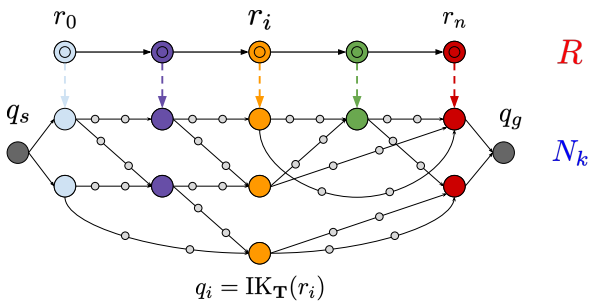

Fig. 4. NNF constructs a graph N_k in configuration-space to follow a reference path R in task-space. Note that a single waypoint $r_i \in R$ may have multiple $\text{IK}_{\mathbf{T}}$ solutions sampled. In this example $\text{IK}_{\mathbf{T}}$ solutions are connected by $j = 2$ interpolated configurations, represented as grey nodes.

C. Problem Statement

In our previous work with NNF [19], $\mathbf{T}$ was (arbitrarily) set and the problem called for optimizing (4). However, this initial pose is likely sub-optimal given the highly-constrained anatomy and the unintuitive, nonlinear kinematics of the CTR. Thus, we expand our problem statement by both optimizing the insertion pose $\mathbf{T}$ and generating a corresponding motion plan γ for this pose. Formally, our optimization problem is now:

$$\arg\min_{\mathbf{T} \in SE(3)} \arg\min_{\gamma \in \Gamma_{\mathbf{T}}} \mathcal{F}(\text{FK}_{\mathbf{T}}(\gamma), R). \tag{7}$$

To do this, we apply our approach to optimizing the problem setup detailed in Fig. 2. ASA is applied over $SE(3)$, the space of insertion poses. During this process NNF is used to evaluate the fitness of each candidate pose, done by returning the value of the objective in (4) after planning.

V. Designing a Lazy Motion Planner

A. Motivation

Using the NNF motion planner to evaluate the fitness of candidate problem setups (i.e. insertion poses) allows us to accurately account for the motions required to follow the path R under each one. We showed previously in Sec. III that we can utilize the bound $\mathcal{B}$ from (2) in this process to limit the computation done by the planner in each evaluation: once this cost is exceeded, the planner should abort given that any additional work it performs is irrelevant to the optimizer.

Notably, motion planners can be redesigned to better take advantage of such a bound. A search-based planner operating on a graph $\mathcal{G}$, for example, should be made as *lazy* as possible: instead of incurring a large computational cost on initialization, the planner should defer expensive computations until the corresponding nodes and edges of $\mathcal{G}$ are processed by the search. One example would be collision-checking edges of $\mathcal{G}$ as the search discovers them, rather than removing all in-collision edges of $\mathcal{G}$ before the search begins.

In this case, aborting the planner limits the set of processed nodes and thus naturally prunes many of these expensive computations. Given that NNF is one such search-based planner, we redesign it in a lazy fashion to increase the effectiveness of our bound. An outline of our lazy NNF implementation is given in Alg. 1.

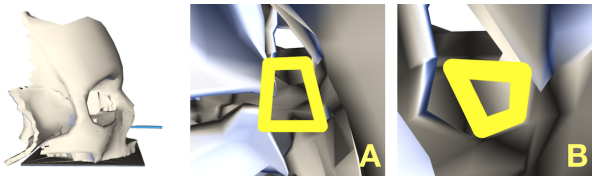

Fig. 5. We use two scenarios in our evaluation, each of which seeks to follow one of these two paths with the tip of the robot. These paths are specified in the back of a human skull-base model, into which the CTR is deployed (left).

B. Lazy Modifications

As is true of many motion planners, one of NNF's largest overheads is collision-checking [19]. We recall that the planner searches for a path $\gamma \in \mathcal{X}$ by first computing a minimal-bottleneck path τ on the tensor-product graph Φ using a modified shortest-path algorithm. In order to validate an edge $(r_x, q_x) \rightarrow (r_y, q_y)$ of Φ, the edge $q_x \rightarrow q_y$ on $\mathcal{X}$ must be collision-checked. To perform these checks lazily [11], we apply the approach of Dellin and Srinivasa [6]. The planner performs no collision-checks initially and assumes that all edges of Φ are valid. The minimal-bottleneck path τ is then generated, the edges of which are validated until either i) the path is found to be completely free, or ii) some edge $e \in \tau$ is found to be in collision, at which point its weight is set to ∞. The search will then repeat and find the new minimal-bottleneck path given this weight change.

This process repeats until all paths are found to be in collision or a solution is found. Although this lazy approach limits collision-checks to edges that may be on the solution path, it also requires many reruns of the search given the frequent weight changes. Thus, rather than using Dijkstra as our search algorithm we instead use bottleneck LPA* [14], which is able to reuse information from prior searches to accelerate future ones. We also memoize the results of collision-checking each edge $q_x \rightarrow q_y$ on $\mathcal{X}$ to avoid repeat computation.

Another significant overhead NNF incurs is invoking $\text{FK}_{\mathbf{T}}$, required to define the weights of E_Φ in (6). This is because computing $\text{FK}_{\mathbf{T}}$ for a continuum robot such as the CTR requires modeling complex phenomena, such as elastic interactions between tubes [22]. In our original NNF implementation [19], Φ was constructed explicitly before the search was run. Not only did this use $\text{FK}_{\mathbf{T}}$ to compute weights of edges never processed by the search, it required a large computational cost simply to initialize the planner.

To avoid this, our lazy NNF implementation represents Φ *implicitly*. An implicit representation only constructs nodes and edges of Φ when they are queried for by the search, with the exception of the start node $(r_0,\ q_s)$. We note that given the definition of E_Φ in (5), we can determine and construct the requested neighbors of any node $(r_x,\ q_x)$ simply by inspecting those of $r_x \in V_R$ (where R is given as input) and $q_x \in V_{N_k}$ (where N_k is still constructed explicitly). Only then will $\text{FK}_{\mathbf{T}}$ be invoked as in (6) to calculate the weights of the corresponding edges.

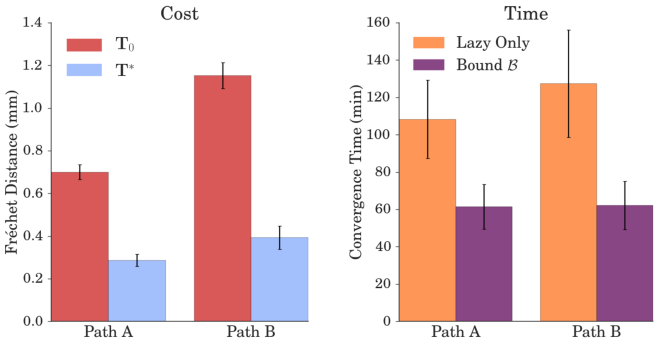

Fig. 6. At left, we compare the costs for each path using the initial insertion pose $\mathbf{T}_0$ (red) and the final insertion pose $\mathbf{T}^*$ (blue). At right, we report convergence times for each path with (purple) and without (orange) use of the bound $\mathcal{B}$.

C. Aborting Evaluation

Having incorporated a large degree of laziness into our planner, we now utilize the bound $\mathcal{B}$. We compare the priority of each node expanded by bottleneck LPA* to $\mathcal{B}$, and abort further evaluation once it is exceeded. Since we do not use a heuristic, these priorities in LPA* are analogous to the distance values used in Dijkstra. The new lazy design of our planner makes this bound on the set of expanded nodes especially effective: fewer paths must be collision checked lazily, and fewer edges and nodes of Φ must be constructed. The latter in turn leads to fewer invocations of $\text{FK}_{\mathbf{T}}$.

Similar to Dijkstra, the priority of the node being expanded lower bounds the cost of the minimal-bottleneck path $\tau \in \Phi$, and thus the objective $\mathcal{F}(\text{FK}_{\mathbf{T}}(\gamma), R)$ representing the fitness of the candidate pose $\mathbf{T}$ [14]. Given that the priorities of expanded nodes are strictly non-decreasing over time [14], one can view this as initial optimism in which we assume an $\mathcal{F}$ of zero (the priority of the start node), and slowly increase $\mathcal{F}$ over time as Φ is constructed and edges are invalidated.

VI. EXPERIMENTS AND RESULTS

A. Experiment Design

We evaluate our approach in two scenarios inspired by minimally-invasive surgery of the pituitary gland, which is located adjacent to the brain. Each scenario requires the CTR to follow with its tip one of two paths located in the back of a human skull-base (Fig. 5). We foresee such paths being used to cut windows into the skull, thereby allowing the surgeon access to the pituitary gland.

Each scenario initializes ASA with a unique insertion pose $\mathbf{T}_0$, which is specified by a human familiar with the CTR's kinematics. While both paths are kinematically reachable under their initial insertions, each presents a unique challenge. For Path A, $\mathbf{T}_0$ angles the base of the CTR in a manner that restricts its range of motion around certain portions of the path. Meanwhile, the initial insertion for Path B makes it impossible for the CTR to reach one of the path's corners given obstacles.

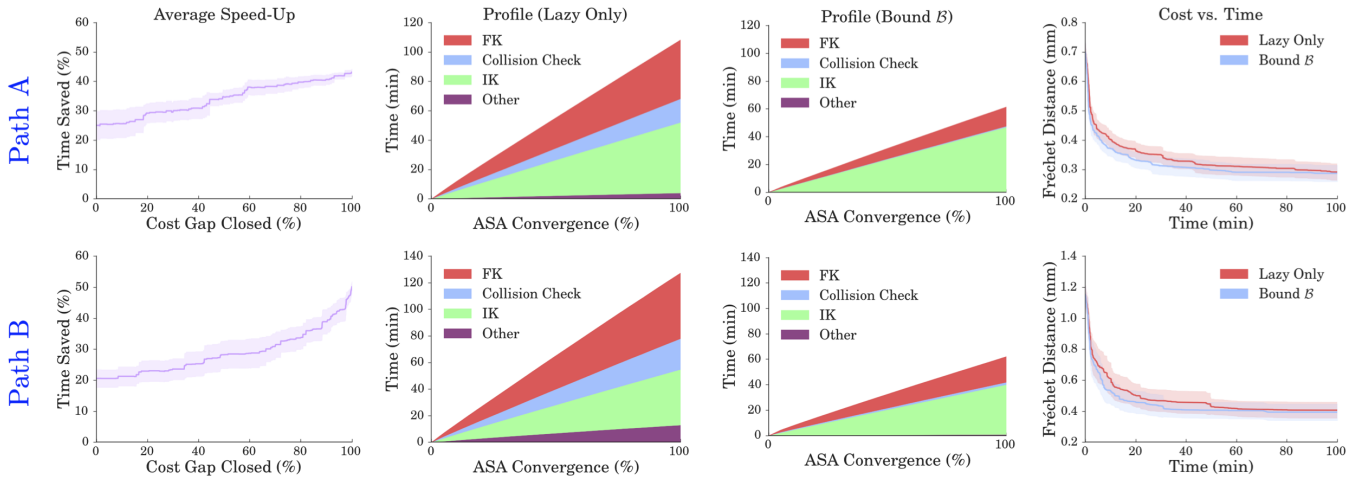

Fig. 7. From left to right: i) The percentage of total time saved as a function of error improvement. ii) A profile of runtime for the optimization both without and with the bound $\mathcal{B}$ used. iii) Fréchet as a function of time with and without $\mathcal{B}$.

Implementation Details: For each path and initial insertion, we run ASA until convergence. We present metrics tied to computational efficiency and improvement in solution quality, all of which are averaged over 20 seeds. We set the parameters for NNF at $n = 150$ $\text{IK}_\mathbf{T}$ samples, $k = 10$ nearest-neighbors, and $j = 3$ interpolated configurations. We use FCL [20] to perform collision-checks, and the kinematic model from [22] to perform $\text{FK}_\mathbf{T}$ and $\text{IK}_\mathbf{T}$. All experiments were run on a 3.50GHz Intel Core i5 CPU with 16GB RAM.

B. Results and Discussion

Overall Results: For both paths, we compare the Fréchet error returned by NNF using the initial insertion pose $\mathbf{T}_0$ and the best pose $\mathbf{T}^*$ found by ASA (Fig. 6). We observe that in both cases $\mathbf{T}^*$ corresponds to a significantly lower error than $\mathbf{T}_0$. This amounts to a decrease in average Fréchet error of **59.1%** for Path A, and **65.9%** for Path B. We depict $\mathbf{T}_0$ and $\mathbf{T}^*$ for a particular run with Path B in Fig. 8. We also demonstrate how $\mathbf{T}^*$ enables the CTR to avoid obstacles and reach the previously infeasible corner with its tip.

To also evaluate the effectiveness of our bound $\mathcal{B}$ (Fig. 2) at increasing computational efficiency, we compare convergence times of ASA for both paths with and without the use of this bound (Fig. 6). In both cases we use the lazy NNF implementation detailed in Sec. V. We see that our approach, which bounds the work done by the planner in evaluating each candidate pose, leads to a dramatic decrease in average convergence time for both paths. This amounts to a **43.2%** drop in average time for Path A and a **51.2%** drop for Path B.

Given the non-interactive timescale of such a process, this optimization would be done offline before the medical procedure itself. Such a dramatic speed-up would be of use in such a scenario, given that it would allow the process to be run with double the number of initializations to produce the best possible result. In addition, many procedures will realistically require the CTR to follow a large set of paths rather than just one. The savings from our speed-up would thus be scaled across planning time for the entire optimization process.

Additional Metrics: We report additional metrics tied to computational efficiency in Fig. 7. These results are discussed in the order they appear from left to right.

We are interested not only in the overall time savings afforded by our bound $\mathcal{B}$, but also how the bound prunes computation as the optimization progresses. Towards this end, the first of these results correlates the average speed-up within each run with the error reduction accomplished by ASA. More technically, we report the percent reduction in compute time required for ASA to close a given percentage of the "cost gap" within each run. We define the cost gap as the difference in Fréchet error from $\mathbf{T}_0$ to $\mathbf{T}^*$: that is, the gap has been 0% closed when ASA evaluates the initial problem setup $\mathbf{T}_0$, and 100% closed when it evaluates the best setup $\mathbf{T}^*$. We note that ASA usually converges at a later point in time than when the cost gap is 100% closed.

Each path behaves differently in this regard. In the case of Path A, the portion of total time saved is roughly linear in the portion of error reduction that has been accomplished. However, the portion of total time saved with Path B increases at a higher rate when more of the cost gap is closed. By the time $\mathbf{T}^*$ is discovered by ASA, runs using Path A save on average **43.1%** of total compute time, while runs using Path B save on average **50.1%**. Although convergence times vary, these results demonstrate that the average *fraction* of time saved to reach the best score remains fairly consistent.

Our next results profile the runtime of the ASA process to demonstrate the proportion of computation spent on various operations. We perform this profiling both without and with using the bound $\mathcal{B}$ to reveal the sources of our efficiency gains. The runtime profiles for both paths are fairly similar, with collision-checks and $\text{FK}_\mathbf{T}$ together constituting a majority of compute time. Given that our lazy NNF implementation makes the cost of these operations proportional to the number of nodes expanded by LPA*, aborting NNF using $\mathcal{B}$ is very effective at reducing this cost and thus the overall runtime of the optimization. This is reflected in the

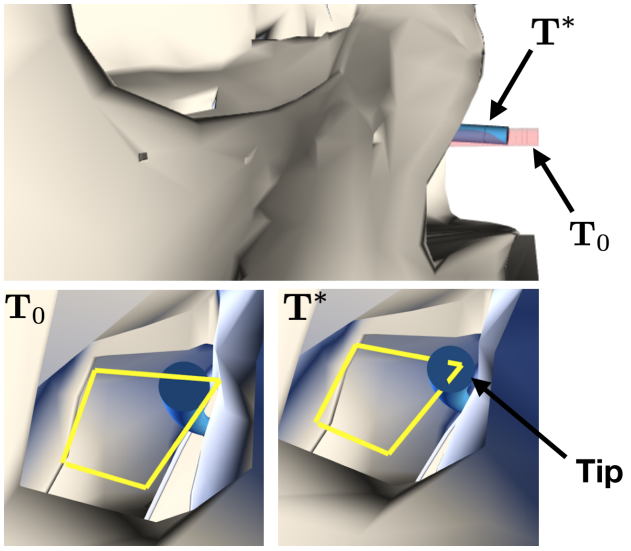

Fig. 8. We depict both the initial insertion $\mathbf{T}_0$ and final insertion $\mathbf{T}^*$ for a run using Path B. We note that under $\mathbf{T}_0$, the CTR's tip is unable to reach the top-right corner of the path due to obstacles. However, $\mathbf{T}^*$ enables the CTR to maneuver around these obstacles.

profiling plots as well, where the only operation not flattened is $\text{IK}_{\mathbf{T}}$. This is logical, as constructing the graph N_k by sampling $\text{IK}_{\mathbf{T}}$ solutions is an initialization cost of NNF.

Finally, we report Fréchet error as a function of runtime. We do so by averaging the lowest error found by each run within a given time budget. We see that ASA makes substantial gains early in the optimization, but displays a fairly long tail required to reach the best insertion pose $\mathbf{T}^*$. Notably, applying $\mathcal{B}$ increases the rate of progress and dramatically shortens this tail.

VII. CONCLUSION AND FUTURE WORK

We present an approach to more-efficiently interleaving optimization with motion planning by using the optimizer to place bounds on the work done by the planner. We then apply this approach to our specific use case of following surgical trajectories by optimizing the insertion pose of the robot. Finally, we present results in this domain that demonstrate significant improvements in computational efficiency.

One of our key limitations in this work is that we consider the parameters of our motion planner to be fixed during the optimization process. As such, one of our next goals is balancing our search over the space of insertion poses with a search over the space of NNF parameters. We intend to use our work in this paper as a fundamental building block in such a process, allowing us to use the motion planner more efficiently as its parameters are tuned.

Unlike the geometric approaches presented in [4] and [5], using NNF to evaluate the fitness of problem setups allows us to consider the motions required of the robot. However, these geometric methods are still less computationally-expensive than ours. Thus, we will explore integrating these less-costly approaches into ours to produce a hybrid approach. One example involves using these geometric approaches to reject candidate poses that would not place the entire path within the reachable task-space of the CTR given the anatomy.

Our approach in this work is simple albiet effective, and we intend to build upon it to further explore tight integration between the optimizer and the planner. For example, in addition to using $\mathcal{B}$ to *abort* the operations of a planner, i.e. one based on graph search, we will investigate ways to *incorporate* this bound into the operations of the planner itself. One approach could use an informed RRT [9] that only samples points which keep the length of paths in the tree below $\mathcal{B}$. We also intend, of course, to use this approach with a wide variety of planners and optimizers.

REFERENCES

[1] R. Almadhoun, T. Taha, L. Seneviratne, J. Dias, and G. Cai. A survey on inspecting structures using robotic systems. *International Journal of Advanced Robotic Systems*, 13(6), 2016.

[2] R. Alterovitz, M. Branicky, and K. Goldberg. Motion planning under uncertainty for image-guided medical needle steering. *IJRR*, 27:1361–1374, 2008.

[3] B. Baykal and R. Alterovitz. Asymptotically optimal design of piecewise cylindrical robots using motion planning. In *RSS*, pages 1–10, 2017.

[4] C. Bergeles, A. Gosline, N. Vasilyev, P. J. Codd, P. del Nido, and P. Dupont. Concentric tube robot design and optimization based on task and anatomical constraints. *IEEE Transactions on Robotics*, 31:1–18, 2015.

[5] J. Burgner-Kahrs, H. Gilbert, and R. J. Webster. On the computational design of concentric tube robots: Incorporating volume-based objectives. In *ICRA*, pages 1193–1198, 2013.

[6] C. Dellin and S. Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *ICAPS*, pages 459–467, 2016.

[7] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[8] M. Feng, X. Jin, W. Tong, X. Guo, J. Zhao, and Y. Fu. Pose optimization and port placement for robot-assisted minimally invasive surgery in cholecystectomy. *International Journal of Medical Robotics and Computer Assisted Surgery*, 2017.

[9] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics*, 34:966–984, 2018.

[10] H. Gilbert, D. Rucker, and R. Webster. Concentric tube robots: The state of the art and future directions. In *ISRR*, pages 253–269, 2013.

[11] N. Haghtalab, S. Mackenzie, A. Procaccia, O. Salzman, and S. Srinivasa. The provable virtue of laziness in motion planning. In *ICAPS*, pages 106–113, 2018.

[12] Y. Hayashi, K. Misawa, and K. Mori. Optimal port placement planning method for laparoscopic gastrectomy. *International Journal of Computer Assisted Radiology and Surgery*, 12:1677–1684, 2017.

[13] R. M. Holladay, O. Salzman, and S. S. Srinivasa. Minimizing task space frechet error via efficient incremental graph search. *RAL*, 2019. To appear.

[14] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A*. *Artif. Intell.*, 155(1-2):93–146, 2004.

[15] A. Kuntz, C. Bowen, C. Baykal, A. W. Mahoney, P. L. Anderson, F. Maldonado, R. James Webster, and R. Alterovitz. Kinematic design optimization of a parallel surgical robot to maximize anatomical visibility via motion planning. In *ICRA*, pages 926–933, 2018.

[16] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 1966.

[17] M. Locatelli. *Simulated Annealing Algorithms for Continuous Global Optimization*, pages 179–229. Springer, 2002.

[18] A. Makhal and A. K. Goins. Reuleaux: Robot base placement by reachability analysis. *CoRR*, abs/1710.01328, 2017.

[19] S. Niyaz, A. Kuntz, O. Salzman, R. Alterovitz, and S. S. Srinivasa. Following surgical trajectories with concentric tube robots. In *ISER*, 2018.

[20] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *ICRA*, pages 3859–3866, 2012.

[21] R.Y. Rubinstein, A. Ridder, and R. Vaisman. *Fast Sequential Monte Carlo Methods for Counting and Optimization*. Wiley, 2013.
[22] D. Rucker. *The mechanics of continuum robots: model-based sensing and control*. PhD thesis, Vanderbilt University, 2011.
[23] K. Solovey and D. Halperin. Sampling-based bottleneck pathfinding with applications to fréchet matching. In *European Symposium on Algorithms, ESA*, pages 1–16, 2016.
[24] N. Vahrenkamp, T. Asfour, and R. Dillmann. Robot placement based on reachability inversion. In *ICRA*, pages 1970–1975, 2013.
[25] R. J. Webster, J. Memisevic, and A. M. Okamura. Design considerations for robotic needle steering. In *ICRA*, pages 3588–3594, 2005.
[26] T. Wylie et al. *The discrete Fréchet distance with applications*. PhD thesis, Montana State University-Bozeman, College of Engineering, 2013.