On the Algorithmic Foundations of Task Assistance Planning

Eitan Bloch[1] and Oren Salzman[1]

Technion — Israel Institute of Technology Haifa, Israel.
{eitanbloch,osalzman}@cs.technion.ac.il

Abstract. In this work we introduce the problem of task assistance planning where we are given two robots $\mathsf{R}_{\mathbf{task}}$ and $\mathsf{R}_{\mathbf{assist}}$. The first robot, $\mathsf{R}_{\mathbf{task}}$, is in charge of performing a given task by executing a precomputed path. The second robot, $\mathsf{R}_{\mathbf{assist}}$, is in charge of assisting the task performed by $\mathsf{R}_{\mathbf{task}}$ using on-board sensors. The ability of $\mathsf{R}_{\mathbf{assist}}$ to provide assistance to $\mathsf{R}_{\mathbf{task}}$ depends on the locations of both robots. Since $\mathsf{R}_{\mathbf{task}}$ is moving along its path, $\mathsf{R}_{\mathbf{assist}}$ may also need to move to provide as much assistance as possible. The problem we study is how to compute a path for $\mathsf{R}_{\mathbf{assist}}$ so as to maximize the portion of $\mathsf{R}_{\mathbf{task}}$'s path for which assistance is provided. We limit the problem to the setting where $\mathsf{R}_{\mathbf{assist}}$ moves on a roadmap which is a graph embedded in its configuration space and show that this problem is NP-hard. Fortunately, we show that when $\mathsf{R}_{\mathbf{assist}}$ moves on a given path, and all we have to do is compute the times at which $\mathsf{R}_{\mathbf{assist}}$ should move from one configuration to the following one, we can solve the problem optimally in polynomial time. Together with carefully-crafted upper bounds, this polynomial-time algorithm is integrated into a Branch and Bound-based algorithm that can compute optimal solutions to the problem outperforming baselines by several orders of magnitude. We demonstrate our work empirically in simulated scenarios containing both planar manipulators and UR robots as well as in the lab on real robots.

Keywords: Motion and Path Planning · Algorithmic Completeness and Complexity.

1 Introduction

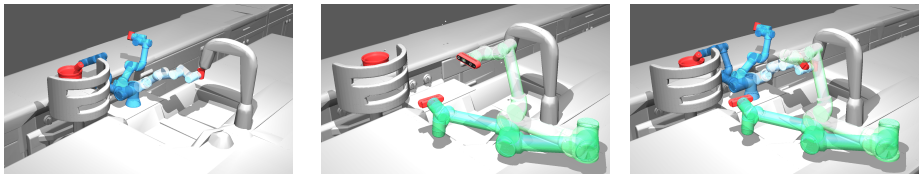
In this work we introduce the problem of *task assistance planning* (TAP) where we are given two robots $\mathsf{R}_{\mathbf{task}}$ and $\mathsf{R}_{\mathbf{assist}}$. The first robot, $\mathsf{R}_{\mathbf{task}}$, which we call the *task robot*, is in charge of performing a given task by executing a precomputed path. The second robot, $\mathsf{R}_{\mathbf{assist}}$, which we call the *assistance robot*, is in charge of assisting the task performed by $\mathsf{R}_{\mathbf{task}}$ using on-board sensors. The ability of $\mathsf{R}_{\mathbf{assist}}$ to provide assistance to $\mathsf{R}_{\mathbf{task}}$ depends on the locations of both robots. Since $\mathsf{R}_{\mathbf{task}}$ is moving along its path, $\mathsf{R}_{\mathbf{assist}}$ may also need to move to provide as much assistance as possible. In its simplest form, the problem calls for computing a path for $\mathsf{R}_{\mathbf{assist}}$ so as to maximize the portion of $\mathsf{R}_{\mathbf{task}}$'s path for which assistance is provided.

Examples of assistance include providing visual feedback, providing communication relays or providing a protective defensive coverage. An application where visual feedback is key is visual monitoring of a task, such as in household applications as visualized in Fig. 1. Another such example is semi-autonomous minimally-invasive robotic surgery where $\mathsf{R}_{\mathbf{task}}$ is a tool tele-operated by a surgeon who is tasked with suturing or removing a tumor and $\mathsf{R}_{\mathbf{assist}}$ is an autonomous endoscope capable of providing the surgeon with visual feedback by taking a path in which the surgeon's tool should be as visible as possible. An application where communication relays are key is search-and-rescue in a limited-communication region. Here, $\mathsf{R}_{\mathbf{task}}$ is an autonomous ground vehicle (AGV) that needs to transmit real-time data to a base in a disaster-ridden area. $\mathsf{R}_{\mathbf{assist}}$ is equipped with a communication-relay device and is in charge of autonomously providing the AGV with a stable communication link by taking a path in which it can retransmit data to the base. Finally, an application where protective defensive coverage is key is in military settings where a logistic unit travels along a threatened route. Here, $\mathsf{R}_{\mathbf{task}}$ is the logistic unit and the threats come from different directions according to the terrain and the location along the route. $\mathsf{R}_{\mathbf{assist}}$ is an autonomous drone capable of defending the logistic unit.

TAP requires solving the motion-planning problem [14,19] where we compute a collision-free path for a robotic system while also accounting for assistance constraints such as visibility constraints [23]. Unfortunately, the motion-planning problem is already computationally challenging [13,30] and adding assistance constraints only further complicates the problem. Roughly speaking, here we need to plan a path for the assistance robot while accounting for when and where assistance is provided. This results in a motion-planning variant where Bellman's principle of optimality does not hold (i.e., it may be worthwhile not to provide assistance at early stages of the task in order to be able to provide more assistance in later stages of the task). This makes existing motion-planning algorithms unsuitable to address this problem.

In this work we lay the algorithmic foundations for TAP. Specifically, we study the problem when planning is restricted to discrete graphs. Here, we follow common approaches to solve robotic motion-planning problems by discretizing the continuous configuration space (C-space) into a roadmap [19,28].

We start (Sec. 4) by considering the most simple setting where we are given the path of $\mathsf{R}_{\mathbf{assist}}$ as a sequence of vertices and only need to decide when it should transition from one vertex to the next. We show that although this is a continuous planning problem, computing transition times that maximize assistance provided can be done in polynomial time. Moving to the general problem of TAP on graphs, we start (Sec. 5) by proving that TAP is NP-hard by a reduction from the subset-sum problem. We proceed (Sec. 6) to present a Branch and Bound (B&B) algorithm that integrates the optimal algorithm for paths together with a method that allows to efficiently prune large parts of the search space. As we demonstrate empirically (Sec. 7), this algorithm allows to efficiently compute solutions for complex problems significantly outperforming the optimal baseline by roughly three to four orders of magnitude, and successfully computing a so-



(a) (b) (c)

Fig. 1: TAP in household applications. (a) Blue manipulator $\mathsf{R}_{\mathbf{task}}$ is tasked with transferring water in a cup from a faucet to a pot (light to dark blue correspond with initial to final configurations). (b) Green manipulator $\mathsf{R}_{\mathbf{assist}}$ is equiped with a camera that must detect if water is spilled from the cup to initiate clean-up and to ensure that the pot has enough water. Here, the light-green and dark-green robots depict configurations for which the cup is visible and non-visible by $\mathsf{R}_{\mathbf{assist}}$'s point of view, respectively. (c) A task assistance path maximizing the amount of time the cup is observed by $\mathsf{R}_{\mathbf{assist}}$ (light to dark green correspond eith initial to final configurations). Visualization adapted from [25].

lution for significantly larger graphs. Compared to a non-optimal baseline, our algorithm computes paths that can improve assistance by a factor of roughly 3×.

2 Related Work

Assisting agents in collaborative settings. Our work bares resemblance to research for enabling agents to assess their need for help and their ability to be helpful. This has been investigated using the notion of *Value of Information* [15,27,31] to quantify the impact information has on autonomous agents' decisions and utilities. However, in contrast to our setting, here there is typically no centralized control, thus requiring coming up with local decisions. Arguably, the most closely-related work to our new problem is recent work on computing the *Value of Assistance* [3,21] which allows to estimate the expected effect an intervention will have on a robot's belief. In contrast to our work, this problem is limited to providing assistance at one point along the task-robot's path and the question at hand is where should this assistance be given.

Visual assistance & planning with visual constraints. Variants of TAP where the assistance is visual feedback have been studied throughout the years but none of the tools developed are directly applicable to our setting. Specifically, our problem falls under the broad category of robot target detection and tracking which encompasses a variety of decision problems such as coverage, surveillance, and pursuit-evasion [26]. These kind of problems have typically been studied in the adversarial setting (see, e.g., [5]) where one group of robots attempts to track down members of another group, while we are interested in the *cooperative setting* where the task and assistance robots work in concert (or at least do not deliberately attempt to jeopardize task assistance). Moreover, existing work

typically considers relatively simple low-dimensional systems (see, e.g., [18,20]) in contrast to the high-dimensional ones that we are interested in.

Visual assistance is also closely related to planning camera motions (see, e.g., [11,12,22,24]) where we are tasked with planning the motions of a free-flying camera to follow a given object. However, these problems do not need to account for the robots' potentially high-dimensional configuration spaces and are often studied in relatively uncluttered environments. Finally, our problem bares resemblance to the scene-reconstruction problem which has to do with creating a digital model of a real-world scene from a set of images or other measurements of a scene (see, e.g., [4,7]). However, in contrast to our problem, here there is no need to account for (i) time, forcing us to capture images in a pre-defined order and (ii) collision avoidance, forcing us to account for the geometry of $\mathsf{R}_{\mathbf{assist}}$.

Inspection planning. Closely related to our work is the problem of *inspection planning* [8,9], or *coverage planning* [2,10]. Here, we are given a robot R with an on-board sensor, a region of interest (ROI) to be inspected by R, and the environment. A point on the ROI is considered inspected if R's sensor sees it without any object occluding the view. Inspection planning calls for computing a collision-free path for R that maximizes the portion of the ROI that is inspected while obeying R's kinematic constraints. Similar to visual assistance, the order in which POIs are inspected is unimportant which makes algorithms developed for inspection planning difficult to apply to our setting of TAP.

3 Notation & Problem Definitions

Let $G = (V, E)$ be a graph which we call a *task-assistance graph* corresponding to configurations of $\mathsf{R}_{\mathbf{assist}}$. Time is normalized to be in the range $[0, 1]$ and each vertex $v \in V$ is associated with *a set of time intervals* $\mathcal{I}(v)$ corresponding to the times where assistance can be provided from v (e.g., the times when $\mathsf{R}_{\mathbf{task}}$'s path can be inspected when the task is visual assistance).[1] Additionally, each vertex is associated with a set of valid intervals in which $\mathsf{R}_{\mathbf{assist}}$ is allowed to reside in that vertex (times in which $\mathsf{R}_{\mathbf{assist}}$ can't reside at a vertex allow our model to incorporate avoiding moving obstacles such as $\mathsf{R}_{\mathbf{task}}$).[2] Each edge $e \in E$ is associated with a length $\ell(e)$ and we assume for simplicity that (i) moving along an edge takes time that is identical to its length and that (ii) when moving along an edge $e = (u, v)$, assistance is defined as identical to the assistance at u and at v for the first and second half of the edge e, respectively.[3]

[1] Note that $\mathsf{R}_{\mathbf{task}}$'s path is not explicitly provided but it is implicitly defined in the task-assistance graph.

[2] To simplify exposition, unless stated otherwise, we assume that $\mathsf{R}_{\mathbf{assist}}$ is allowed to reside in every vertex during the whole task duration but all of our results can easily be adapted to the general setting.

[3] Our model doesn't account for dynamics such as bounded acceleration but is a sufficient first-order approximation.

Fig. 2: (a) Toy TAP problem. Above each vertex and edge are the intervals for which task assistance can be performed and the edge length, respectively. (b) Two timing profiles for path $\langle v_0, v_1, v_2 \rangle$. Here, each vertex is depicted together with the intervals for which task assistance can be performed. The timing profiles (blue and orange) consist of solid and dotted lines when task assistance can and can't be performed, respectively. Note that the slopes of moving from v_0 to v_1 and from v_1 to v_2 are different as the corresponding edge lengths are different (0.1 and 0.25, respectively).

Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path such that $v_i \in V$ and $(v_i, v_{i+1}) \in E$. As we will shortly see, it will be convenient to introduce several notation: We set $\ell_\pi(v_i)$ to be the length of the path from v_0 to v_i. Namely, $\ell_\pi(v_i) := \sum_{j=0}^{i-1} \ell(v_j, v_{j+1})$. Additionally, we set $\ell_\pi^-(v_i)$ and $\ell_\pi^+(v_i)$ to be the length of the path from v_0 to the middle of incoming and outgoing edge of v_i, respectively. Namely, $\ell_\pi^-(v_i) := \ell_\pi(v_i) - \frac{1}{2}\ell(v_{i-1}, v_i)$ and $\ell_\pi^+(v_i) := \ell_\pi(v_i) + \frac{1}{2}\ell(v_i, v_{i+1})$. Additionally, set $\ell_\pi^+(u, v) := \ell_\pi^+(v) - \ell_\pi^+(u)$. To simplify exposition we define $\ell_\pi(v_{-1}, v_0) := 0$ and $\ell_\pi(v_k, v_{k+1}) := 0$. When understoodfrom the context, we omit π from $\ell_\pi(v_i), \ell_\pi^+(v_i), \ell_\pi^-(v_i)$ and $\ell_\pi^+(u, v)$. Finally, we denote by $N_{\mathcal{I}}^\pi$ and $N_{\mathcal{I}}^G$ the total number of intervals of all vertices in a path π and a graph G, respectively.

Importantly, a path only defines where $\mathsf{R}_{\mathbf{assist}}$ is but not when it needs to transition from one vertex to another. Thus, paths need to be augmented with a sequence of timestamps representing the times at which $\mathsf{R}_{\mathbf{assist}}$ should transition from one vertex to another. Following our model, these timestamps are defined as the times at which the robot should reach the middle of each edge.

Definition 1 (Timing-profile)**.** *Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path. We define a* timing-profile *of π as a sequence of timestamps $\mathbb{T}_\pi = \langle t_0, \ldots, t_{k-1} \rangle$ such that: (i) $t_0 \geq \ell^+(v_0)$, (ii) $t_{i+1} \geq t_i + \ell^+(v_i, v_{i+1})$ and (iii) $t_{k-1} + \ell^+(v_{k-1}, v_k) \leq 1$. Note that given π and $\mathbb{T}_\pi$, it is straightforward to derive the times that $\mathsf{R}_{\boldsymbol{assist}}$ will arrive and leave each vertex.*

In order to quantify the effectiveness of the assistance provided, we define the reward as the portion of time where assistance is provided. Formally,

Definition 2 (Reward at a vertex)**.** *Let u be a vertex and let $0 \leq t \leq t' \leq 1$ be two times, we denote by $\mathcal{R}(u, t, t')$ the* reward at vertex *u obtained between times t and t'. Namely, $\mathcal{R}(u, t, t') = \sum_{I \in \mathcal{I}(u)} |[t, t'] \cap I|$.*

Definition 3 (Reward of a timing-profile)**.** *Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path, and let $\mathbb{T}_\pi = \langle t_0, \ldots, t_{k-1} \rangle$ be a timing-profile, we define $\mathcal{R}(\pi, \mathbb{T}_\pi)$ to be the* reward

of the timing-profile $\mathbb{T}_\pi$. *Namely, if we set* $t_{-1} = 0, t_k = 1$, *then we have that* $\mathcal{R}(\pi, \mathbb{T}_\pi) := \sum_{i=0}^{k} \mathcal{R}(v_i, t_{i-1}, t_i)$.

As an example, consider Fig. 2 depicting a TAP Instance. Fig. 2b depicts two timing profiles for path $\langle v_0, v_1, v_2 \rangle$: $\mathbb{T}_{\text{blue}} = \langle 0.1, 0.5 \rangle$ and $\mathbb{T}_{\text{orange}} = \langle 0.6, 0.825 \rangle$ (recall the times in a timing profile are the times at which the robot reaches the middle of the edges). $\mathbb{T}_{\text{blue}}$ obtains a reward of $0.1, 0.3$ and 0.35 at v_1, v_2 and v_3, respectively. Thus, $\mathcal{R}(\pi, \mathbb{T}_{\text{blue}}) = 0.75$ while $\mathbb{T}_{\text{orange}}$ obtains a reward of $0.4, 0.125$ and 0.05 at v_1, v_2 and v_3, respectively. Thus, $\mathcal{R}(\pi, \mathbb{T}_{\text{orange}}) = 0.575$.

We are finally ready to define our optimization problems:

Problem 1 (OTP)**.** *Let* π *be a path and let* $\mathcal{T}(\pi)$ *be the set of all possible timing profiles over* π. *The* Optimal Timing-Profile (OTP) *problem calls for computing a timing-profile* $\mathbb{T}^*$ *for* π *whose reward is maximal. Namely, compute* $\mathbb{T}^*$ *s.t.,*

$$\mathbb{T}^* \in \arg\max_{\mathbb{T} \in \mathcal{T}(\pi)} \mathcal{R}(\pi, \mathbb{T}).$$

Problem 2 (OPTP)**.** *Let* G *be a task assistance graph,* $v_0 \in V$ *a start vertex. Let* $\Pi(v_0)$ *be the set of paths in* G *starting from* v_0 *and* $\mathcal{T}(\pi)$ *be the set of all possible timing profiles over a given path* $\pi \in \Pi(v_0)$. *The* Optimal Path and Timing-Profile (OPTP) *problem calls for computing a path* π^* *and a timing profile* $\mathbb{T}^*$ *whose reward is maximal. Namely, compute* $\pi^*, \mathbb{T}^*$ *s.t.,*

$$\pi^*, \mathbb{T}^* \in \arg\max_{\substack{\pi \in \Pi(v_0), \\ \mathbb{T} \in \mathcal{T}(\pi)}} \mathcal{R}(\pi, \mathbb{T}).$$

4 Solving the OTP Problem (Prob. 1)

At first glance, given a specific path π, computing an optimal timing-profile $\mathbb{T}$ may seem to be a continuous optimization problem as each timestamp $t \in \mathbb{T}$ can be in the range $[0, 1]$. However, our first key insight is that we can consider a discrete set of *critical times*. Roughly speaking, these are the start and end time of intervals while accounting for edges length. This is because there are two conceptual reasons to leave a vertex u: either an interval I of u ended and there is no reason to stay at u (as no additional reward will be gained from I), or there is another interval I' at a successor of u we would like to reach further along π.

As an example, consider path $\pi = \langle v_0, v_1, v_2 \rangle$ and the timing profile $\mathbb{T}_{\text{blue}}$ (which is optimal) from Fig. 2. Here, $\mathbb{T}_{\text{blue}}$ leaves vertex v_0 at time 0.1 which is exactly the end time of the first interval of v_0. Next, $\mathbb{T}_{\text{blue}}$ leaves vertex v_1 at time 0.5 which is exactly the time at which the first interval of v_2 starts. As we will show, it is enough to consider only critical times to find an optimal timing profile. We Formally define critical times in Sec. 4.1 and show in Sec. 4.2 how they can be used to solve Prob. 1.

4.1 Critical Times

Given two vertices v_i, v_j in a path $\pi = \langle v_0, \ldots, v_k \rangle$, $v_i \prec_\pi v_j$ denotes that v_i lies before v_j in π (i.e., that $i < j$). Again, when clear, we omit π from $v_i \prec_\pi v_j$. Furthermore, we assume that the first and last vertex in π contain the intervals $[\ell^+(v_0), \ell^+(v_0)]$ and $[1 - \ell^+(v_{k-1}, v_k), 1 - \ell^+(v_{k-1}, v_k)]$, respectively[4]. We start by defining the critical times between two vertices in π.

Definition 4 (Vertex-pair critical times)**.** *Let u, v be vertices in path π such that $u \prec v$. The set of* vertex-pair critical times *$C_T(u, v)$ consists of two types of times, defined as follows:*

T1 *For any interval $I \in \mathcal{I}(u)$, the earliest time to leave u after I terminates is a type T1 time. Namely, all type T1 times of $C_T(u, v)$ are*

$$\bigcup_{[t_s, t_e] \in \mathcal{I}(u)} \{t_e\}.$$

T2 *For any interval $I \in \mathcal{I}(v)$, the latest time needed to leave u in order to reach v at the start of I is a type T2 time. Namely, all type T2 times of $C_T(u, v)$ are*

$$\bigcup_{[t_s, t_e] \in \mathcal{I}(v)} \{t_s - (\ell^-(v) - \ell^+(u))\}.$$

Each vertex-pair has its own critical times, but the critical times of two pairs sharing a vertex are tightly related. For example, given the path $\langle v_0, v_1, v_2 \rangle$ from Fig. 2a, the vertex-pair critical times are:

$$\begin{aligned} C_T(v_0, v_1) &= \{\mathbf{0}, \mathbf{0.05}, \mathbf{0.1}, \mathbf{0.6}, 0.7\} \\ C_T(v_0, v_2) &= \{\mathbf{0}, \mathbf{0.05}, \mathbf{0.1}, \mathbf{0.6}, \underline{0.325}, \underline{0.675}\} \\ C_T(v_1, v_2) &= \{0.4, \underline{0.5}, \underline{0.85}, 0.9\}. \end{aligned}$$

Notice that **bold** critical times are identical, and that underlined critical times of $C_T(v_1, v_2)$ are equal to underlined critical times of $C_T(v_0, v_2)$ shifted by $\ell^+(v_1, v_2) = 0.175$. We formalize this relation using the following observations.

Observation 1. *Let u, v, v' be vertices in path π such that $u \prec v \prec v'$. t is a type T1 in $C_T(u, v)$ iff t is a type T1 in $C_T(u, v')$.*

Observation 2. *Let u, u', v be vertices in path π such that $u \prec u' \prec v$. t is a type T2 in $C_T(u, v)$ iff $t + \ell^+(u, u')$ is a type T2 in $C_T(u', v)$.*

Next, we use the notion of vertex-pair critical times to define vertex-critical times which include the times from which $\mathsf{R}_{\mathbf{assist}}$ might want to leave a vertex.

[4] These represent the earliest time the first vertex can be left and the latest time the last vertex can be reached, respectively. Adding these intervals does not affect the reward as both interval lengths are zero and are only used to simplify the definitions.

Definition 5 (Vertex-critical times)**.** *Let v_i be a vertex in path π. The set of* vertex-critical times *C_{T_i} for v_i is defined as follows: Let $t_u \in C_T(u,v)$ be a vertex-pair critical time for some vertices u, v in π s.t. $u \prec v$. Then, if*

$\mathbf{v_i \prec u}$*, C_{T_i} includes the latest time needed to leave v_i to leave u at time t_u.*
$\mathbf{v_i = u}$*, C_{T_i} includes t_u.*
$\mathbf{u \prec v_i}$*, C_{T_i} includes the earliest time we can leave v_i given we left u at time t_u.*

Formally, $C_{T_i} = \bigcup_{u \prec v} \{t_u + \ell^+(u, v_i) \mid t_u \in C_T(u,v)\}$.

Similar to vertex-pair critical times, vertex-critical times of different vertices are tightly related as well. Following our example from Fig. 2a we have that:

$$\begin{aligned}
\mathrm{C}_{T_0} &= \{0, 0.05, 0.1, 0.225, 0.325, 0.6, 0.675, 0.7, 0.725\},\\
\mathrm{C}_{T_1} &= \{0.175, 0.225, 0.275, 0.4, 0.5, 0.775, 0.85, 0.875, 0.9\},\\
\mathrm{C}_{T_2} &= \{0.35, 0.4, 0.525, 0.625, 0.9, 0.975, 1.0, 1.025, 1.35\}.
\end{aligned}$$

Notice that the critical times are identical up to a constant shift. The following observation formalizes this relation.

Observation 3. *For any vertex v_i, the vertex-critical times C_{T_i} are equal to the vertex-critical times of C_{T_0} shifted by $\ell^+(v_0, v_i)$. Formally,*

$$C_{T_i} = \{t + \ell^+(v_0, v_i) \mid t \in C_{T_0}\}.$$

Lemma 1. *C_{T_0} can be computed in $\mathcal{O}(k + N_{\mathcal{I}}^{\pi})$ time.*

Proof (sketch). Following Obs. 1 and 2, there are $\mathcal{O}(N_{\mathcal{I}}^{\pi})$ type $T1$ and type $T2$ critical times in C_{T_0}, respectively. Computing these critical times can be done by iterating once over each interval which can be done in $\mathcal{O}(k + N_{\mathcal{I}}^{\pi})$ time.

Note. Following Obs. 3 we can compute C_{T_i} in $\mathcal{O}(|\mathrm{C}_{T_0}|)$ given C_{T_0}.

The next theorem states that an optimal timing-profile can be found by only considering vertex-critical times. To prove the theorem, we show that any optimal timing profile can be transformed to one that contains vertex-critical times only. See Appendix A for details.

Theorem 1. *For any path $\pi = \langle v_0, \ldots, v_k \rangle$, there exists an optimal timing profile $\mathbb{T}_\pi = \langle t_0, \ldots, t_{k-1} \rangle$ such that $\forall i : t_i \in C_{T_i}$.*

4.2 Algorithm

Thm. 1 allows us to present an efficient algorithm that computes the optimal reward (i.e., the reward obtained by following an optimal timing-profile). Specifically, it implies that there exists an optimal timing profile that belongs to $\mathrm{C}_{T_0} \times \ldots \times \mathrm{C}_{T_k}$. Clearly, one could iterate over all such timing profiles but this is highly-inefficient. Instead, we maintain for each vertex a set of so-called *time-reward pairs*. Such a time reward-pair (t, r) at vertex v_i represent a time $t \in \mathrm{C}_{T_i}$

Algorithm 1 OTP

Input: path: $\pi = \langle v_0, \ldots, v_k \rangle$; intervals: $\mathcal{I}$
Output: reward of $\mathbb{T}^*$: $R_{\max}$ // $\mathbb{T}^*$ can be immediately computed

1: ENTRY $\leftarrow \{(0,0)\}$; EXIT $\leftarrow \emptyset$
2: **for** $v_i \in \pi$ **do**
3: $R_{\max} \leftarrow 0$
4: **for** $t_{\text{exit}} \in \text{CT}_i$ **do** // computed using Lemma 1 and Obs. 3
5: $r \leftarrow$ `compute_best_reward`(ENTRY, v_i, t_{exit}) // Alg. 2
6: **if** EXIT $\neq \emptyset$ and $r \leq R_{\max}$ **then** // Pareto frontier optimization
7: **continue**
8: $R_{\max} \leftarrow r$
9: EXIT.insert((t_{exit}, r))
10: ENTRY $\leftarrow$ EXIT; EXIT $\leftarrow \emptyset$
11: **return** $R_{\max}$

and a reward r that can be obtained by reaching v_i while following a certain timing-profile until time t. These time-reward pairs are computed by iterating along the path vertices one at a timewhile removing time-reward pairs that can't belong to an optimal timing profile.

Our algorithm (Alg. 1) maintains two time-reward lists ENTRY, EXIT representing the list of optional entry and exit times to a certain vertex, respectively. They are initialized to $(0,0)$ (corresponding to the fact that the initial vertex is entered at time zero with zero reward) and an empty list, respectively (Line 1). The algorithm then proceeds by iterating over the vertices of π starting at v_0 (Lines 2-10). For each vertex v_i, it iterates over all optional exit times $t_{\text{exit}} \in \text{CT}_i$ (Lines 4-9) and for each such exit time t_{exit}, computes the best reward obtainable given that vertex v_i must be left at time t_{exit} (Line 5) using the function `compute_best_reward` (Alg 2). This function simply iterates over all time-reward pairs in ENTRY. For each entry time-reward pair $(t_{\text{entry}}, r_{\text{entry}})$, it computes the reward obtainable by entering vertex v_i at time t_{entry} and leaving it at time t_{exit} and adds it to the reward obtained prior to entering vertex v_i. After finishing v_i's iteration, it sets the exit times of v_i as the entry times of v_{i+1}, and sets the exit list to be empty (Line 10). Finally, the algorithm returns the maximal reward it obtained (Line 11).

Pareto-frontier optimization. Consider the path $\pi = \langle v_0, v_1, v_2 \rangle$ from Fig. 2a and recall that $\text{CT}_0 = \{0, 0.05, 0.1, 0.225, 0.325, 0.6, 0.675, 0.7, 0.725\}$. After the first iteration of the algorithm, the time-reward pairs of EXIT are:

$$\{(0.05, 0.05), (0.1, 0.1), (0.225, 0.1), (0.325, 0.125),$$
$$(0.6, 0.4), (0.675, 0.4), (0.7, 0.4), (0.725, 0.4), (1, 0.4)\}.$$

Now consider the two pairs $p = (0.1, 0.1)$ and $p' = (0.225, 0.1)$. Here p' can be pruned as it leaves the vertex after p and its reward is not better. In the general setting, we only need to maintain the Pareto frontier [29] as given

Algorithm 2 `compute_best_reward`$(\textsc{Entry}, v_i, t_{\text{exit}})$

Input: entry list of critical times: $\textsc{Entry}$; vertex: v_i; exit time: t_{exit}
Output: best reward given we leave v_i at time t_{exit} : R_{best}

1: $R_{\text{best}} \leftarrow 0$
2: **for** $(t_{\text{entry}}, r_{\text{entry}}) \in \textsc{Entry}$ **do**
3: **if** $t_{\text{entry}} + \ell^+(v_{i-1}, v_i) > t_{\text{exit}}$ **then** // t_{entry} is too late to leave at t_{exit}
4: **continue**
5: $r_{\text{exit}} \leftarrow r_{\text{entry}} + \mathcal{R}(v_i, t_{\text{entry}}, t_{\text{exit}})$ //reward if v_i is visited at $[t_{\text{entry}}, t_{\text{exit}}]$
6: $R_{\text{best}} \leftarrow \max\{R_{\text{best}}, r_{\text{exit}}\}$
7: **return** R_{best}

two time-reward exit pairs $(t_1, r_1), (t_2, r_2)$ such that $t_1 < t_2$, it is worth considering leaving v_i at time t_2 only if $r_2 > r_1$. As the time-reward exit pairs are ordered from earliest to latest, we add a pair (t_{exit}, r) to $\textsc{Exit}$ only if the reward is greater then the reward of the previous pair (Line 9). Returning to our example, after the first iteration $\textsc{Exit}$ will be pruned down to: $\{(0.05, 0.05), (0.1, 0.1), (0.325, 0.125), (0.6, 0.4)\}$.

Correctness & complexity (sketch). Following Thm. 1, the reward returned by the algorithm corresponds to the reward of an optimal timing-profile. In addition, we can trace back the time-reward pairs in order to find an optimal timing-profile. Given a path π with k vertices and a total of $N_{\mathcal{I}}^{\pi}$ intervals we can bound the size of C_{T_0} by $\mathcal{O}(N_{\mathcal{I}}^{\pi})$ (Lemma 1). In addition, $\textsc{Entry}$ and $\textsc{Exit}$ are both bounded by $\mathcal{O}(|\text{C}_{\text{T}_0}|)$. Thus, for each vertex, we call Alg. 2 $\mathcal{O}(N_{\mathcal{I}}^{\pi})$ times and it may take an additional $\mathcal{O}\left((N_{\mathcal{I}}^{\pi})^2\right)$ time as computing $\mathcal{R}(v_i, t_{\text{entry}}, t_{\text{exit}})$ may take $\mathcal{O}(N_{\mathcal{I}}^{\pi})$ time. Thus, the total runtime complexity is $\mathcal{O}\left(k \cdot (N_{\mathcal{I}}^{\pi})^3\right)$.

5 Hardness of the OPTP Problem

We now move on to the OPTP problem and show that it is NP-hard. Roughly speaking, the hardness of the problem comes from the fact that in the general setting, there may be an exponential number of paths in a graph and in contrast to the shortest-path problem, there is no notion of monotonicity.

Theorem 2. *The OPTP problem is* NP*-hard.*

Sketch. The proof is by a reduction from the subset-sum problem (SSP) [17]. Recall that the SSP is a decision problem where we are given a set $X = \{x_1, \ldots, x_n \mid x_i \in \mathbb{N}\}$ and a target $K \in \mathbb{N}^+$(for simplicity, we assume that $K > 0$ but this is a technicality only). The problem calls for deciding whether there exists a set $X' \subseteq X$ whose sum $\sum_{x \in X'} x$ equals K.

Given an SSP instance, we build a corresponding OPTP instance (to be explained shortly) and show that there exists a subset $X' \subseteq X$ such that $\sum_{x \in X'} x = K$ iff the optimal reward in our OPTP instance is $\frac{K}{\sum_{i=1}^{n} x_i} + 1$ thus proving the problem is NP-hard.

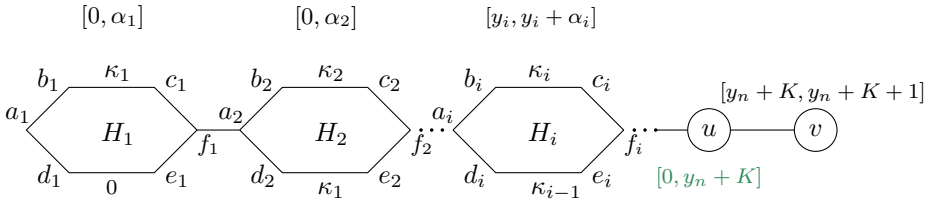


Fig. 3: Reduction graph (all edges are directed from left to right). When omitted, edge length equals zero.

W.l.o.g. assume that $x_1, \ldots, x_n$ are sorted from smallest to largest and set $\kappa_i := \sum_{j=1}^{i} x_j$, $\alpha_i := x_i/\kappa_n$ and $y_i = \sum_{j=1}^{i-1} \kappa_j$. The graph of our OPTP instance depicted in Fig. 3 contains n hexagons $H_1, \ldots H_n$ such that H_i contains vertices $a_i, b_i, c_i, d_i, e_i, f_i$. We call a_i and f_i the entry and exit vertices of H_i, b_i and c_i the top of H_i and d_i and e_i the bottom of H_i. Edges (b_i, c_i) and (d_i, e_i) at the top and bottom of H_i have length κ_i and κ_{i-1}, respectively (all other edge lengths are equal to zero). Edge (b_i, c_i) at the top of H_i contains the interval $[y_i, y_i + \alpha_i]$.[5] Finally, the exit f_n of the last hexagon H_n has an edge to a vertex u which has a valid interval $[0, y_n + K]$ and u has an edge to a vertex v whose assistance interval is $[y_n + K, y_n + K + 1]$. Note that here for ease of exposition, time is in the range $[0, y_n + K + 1]$ and *not* $[0, 1]$.

Our reduction is based on two key properties of the new OPTP instance:

P1 The shortest path to reach u is by taking the lower part of each H_i and its length is y_n.

P2 Let π be a path that passes through u. Going through the upper part of H_i adds additional x_i time to reach u and results in an additional reward of α_i.

Roughly speaking, the valid interval at u forces any path π to v to leave u before time $y_n + K$. As the minimal time to reach u is y_n (Property $P1$), π only has K time units to spend on earning rewards from the hexagons. Now, to obtain a reward of $K/\kappa_n + 1$, π must earn a reward of K/κ_n before reaching u. Since the upper part of H_i adds an additional time of x_i and a reward of α_i (Property $P2$), π must find a combination of upper parts I_{up} such that $\sum_{i \in I_{\text{up}}} x_i = K$ which is the exact solution to the subset-sum problem.

For additional details see Appendix B.

6 Solving the OPTP Problem (Prob. 2)

Given a task assistance graph $G = (V, E)$ and a start vertex v_0, we can iterate over all paths starting from v_0, and for each path run the OTP algorithm (Sec. 4). Unfortunately, the search space can be extremely large (Sec. 5) which deems this approach intractable.

[5] Strictly speaking, the interval belongs to the vertices b_i, c_i. However, it will be more convenient to consider the interval as belonging to the edge (b_i, c_i).

To this end, we suggest to apply the Branch and Bound (B&B) framework [6] to our setting. B&B is a general algorithmic technique used to solve optimization problems, particularly combinatorial-optimization problems, by systematically exploring the solution space in a structured manner. Conceptually, B&B divides the solution space into smaller subspaces (branching) and then systematically searches through these subspaces while keeping track of bounds on the optimal solution (bounding). This allows to prune branches of the search tree that are guaranteed not to contain an optimal solution, thereby reducing the size of the search space and improving efficiency.

Thus, we start (Sec. 6.1) by introducing approaches to bound the reward of partial solutions and then continue (Sec. 6.2) to detail our B&B-based algorithm.

6.1 Upper Bound

In the following, we overview how to bound the reward obtainable from (i) any path that starts at the beginning of an interval I, (ii) any path that starts at a vertex u starting at time t and (iii) from a prefix of a path π. Here, we give a high-level description regarding how these bounds are computed and refer the reader to Appendix C for additional details.

Bounding the reward obtainable from the beginning of an interval Let $I = [t_s, t_e]$ be an interval belonging to vertex u. We set $\mathcal{UB}_I^0 = |I|$ and iteratively compute $\mathcal{UB}_I^i$ using $\mathcal{UB}_I^{i-1}$. As we will see, $\mathcal{UB}_I^i$ will be an upper bound on the reward that may be obtained from interval I followed by at most i subsequent intervals. Note that this indeed holds for $i = 0$ and that if the invariant holds for every iteration, then after $N_{\mathcal{I}}^G$ iterations $\mathcal{UB}_I := \mathcal{UB}_I^{N_{\mathcal{I}}^G}$ is an upper bound on the total reward obtainable starting from interval I and continuing optimally. To compute $\mathcal{UB}_I^i$, we iterate over all intervals I', and bound the reward obtainable assuming the interval after I is I'. This is done by computing t_{unused}, which is the minimal portion of either I or I' in which no reward can obtained when traveling from I to I'. We then bound the reward as $|I| + \mathcal{UB}_{I'}^{i-1} - t_{\text{unused}}$. If the bound obtained is greater than $\mathcal{UB}_I^i$, we update it accordingly.

Importantly, $\mathcal{UB}_I^i$ stores an upper bound on the reward obtained from interval I as well as from future intervals. However, it does *not* store the time I is exited in order to obtain the reward from future intervals. Namely, $\mathcal{UB}_I^i$ does not account for the fact that I may have been entered at a time $t > t_s$. This results in (i) an upper bound on the true reward and (ii) an efficient algorithm whose complexity is cubic in $N_{\mathcal{I}}^G$ (the number of intervals) and not on the number of critical times which may be exponential in $N_{\mathcal{I}}^G$. Note that this process is computed *once*, before running our B&B-based algorithm and will be used to compute the other bounds required by our algorithm.

Before stating the correctness of $\mathcal{UB}_I$, we introduce the following definition:

Definition 6 (Partial reward)**.** *Let π be a path, let $\mathbb{T}$ be a timing-profile and let $t \in [0, 1]$. We define $\mathcal{R}(\pi, \mathbb{T}, t)$ to be the reward obtained from following the timing-profile $\mathbb{T}$ during the time interval $[t, 1]$.*

Lemma 2. *Let π and $\mathbb{T}$ be a path and timing-profile, respectively. Let $I = [t_s, t_e]$ be an interval $\mathbb{T}$ obtains reward from. Then it holds that $\mathcal{UB}_I \geq \mathcal{R}(\pi, \mathbb{T}, t_s)$.*

Bounding the reward obtainable from a vertex u starting at time t Let $u \in V$ be a vertex and $t \in [0, 1]$. Denote $\delta'(u, v)$ to be the minimum distance between u and v in G while not accounting for half of the first and last edge (this can be computed using one Dijkstra-like pass over the graph together with some additional processing). Now, consider an interval $I \in \mathcal{I}(v)$ associated with some vertex v. Intuitively, (i) the reward that I offers (assuming we start at vertex u at time t) cannot be obtained before time $t + \delta'(u, v)$ and (ii) $\mathcal{UB}_I$ bounds the reward that can be obtained from I. Together, these allow to bound the reward obtainable from u at t assuming I is the next interval. We iterate over all such intervals and set $\mathcal{UB}(u, t)$ to be the highest reward.

Lemma 3. *Let $u \in V$ be a vertex, and $t \in [0, 1]$. For every path π and timing-profile $\mathbb{T}$ s.t. $u \in \pi$ and that following $\mathbb{T}$ implies that at time t the assistance robot is at vertex u, it holds that $\mathcal{UB}(u, t) \geq \mathcal{R}(\pi, \mathbb{T}, t)$.*

Bounding the reward obtainable given the prefix of a path Given a path $\pi = \langle v_0, \ldots, v_k \rangle$, we wish to compute an upper bound on the reward that can be obtained from any path $\pi' = \langle v_0, \ldots, v_k, \ldots, v_m \rangle$ whose prefix is π.

Let $\mathbb{T}' = \langle t'_0, \ldots, t'_{m-1} \rangle$ be an optimal timing profile for π' (note that we don't have access to π' and $\mathbb{T}'$ but we will address this shortly). Furthermore, let $I = [t_s, t_e] \in \mathcal{I}(v_i)$ for some $v_i \in \pi$ be the last interval that $\mathbb{T}'$ obtained reward from before leaving v_k. We bound π''s reward by separating it into two parts: the reward obtained (i) until t'_i (the time $\mathbb{T}'$ exits v_i) and (ii) from t'_i.

To bound the reward obtained until t'_i (first part), recall that the OTP algorithm keeps track of time-reward pairs representing a time and the best reward that can be obtained until that time at a certain vertex. As I is the last interval in π that $\mathbb{T}'$ obtains reward from, the reward obtained until t'_i can be bounded by the reward obtained until t_e. This is exactly the reward of the pair (t_e, r) belonging to the Exit list of vertex v_i which can be computed by running the OTP algorithm on π. To bound the reward obtained from t'_i (second part) we make use of $\mathcal{UB}(u, t)$. Recall that we do not actually have access to t'_i but, since $\mathbb{T}$ obtains reward from I it must leave v_i after t_s, which allows us to bound the reward obtainable from t'_i by $\mathcal{UB}(v_i, t_s)$. In addition, since $\mathbb{T}'$ does not obtain any reward between the end of I and the time it leaves v_k, we can further tighten our bound to $\mathcal{UB}(v_k, t_s + \ell^+(v_i, v_k))$. Combining both parts, $\mathcal{R}(\pi', \mathbb{T}')$ can be bounded by $r + \mathcal{UB}(v_k, t_s + \ell^+(v_i, v_k))$.

As we don't have access to π', we can't know which interval I is the last interval $\mathbb{T}'$ obtains reward from. Thus, we must compute this bound for every interval I on the path π and use the maximal bound obtained to be $\mathcal{UB}(\pi)$, our bound on the reward obtainable from the prefix π of a path.

Theorem 3. *Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path. For any path $\pi' = \langle v_0, \ldots, v_k, \ldots, v_m \rangle$ extending π s.t., $\mathbb{T}'_{\pi'} = \langle t'_0, \ldots, t'_{m-1} \rangle$ is its optimal timing profile, it holds that $\mathcal{UB}(\pi) \geq \mathcal{R}(\pi', \mathbb{T}'_{\pi'})$.*

Complexity analysis Given a graph with n vertices, m edges, and $N_{\mathcal{I}}^G$ intervals, computing $\mathcal{UB}_I$ requires computing $\delta'(u, v)$ for every two vertices $u, v \in V$ which can be done in $\mathcal{O}(n \cdot m \cdot \log(n))$. In addition, each iteration i used to compute $\mathcal{UB}_I^i$ iterates over all pairs of I, I'. Since there are $N_{\mathcal{I}}^G$ iterations this takes $\mathcal{O}((N_{\mathcal{I}}^G)^3)$. Thus, computing $\mathcal{UB}_I$ can be done in $\mathcal{O}\left(n \cdot m \cdot \log(n) + (N_{\mathcal{I}}^G)^3\right)$. Computing $\mathcal{UB}(u, t)$ requires iterating once over each interval thus taking $\mathcal{O}(N_{\mathcal{I}}^G)$ time. Finally, given a path π with k vertices and $N_{\mathcal{I}}^\pi$ intervals, computing $\mathcal{UB}(\pi)$ requires running the OTP algorithm over π, and calling $\mathcal{UB}(u, t)$ once for every interval. Thus, computing $\mathcal{UB}(\pi)$ can be done in $\mathcal{O}(k \cdot (N_{\mathcal{I}}^\pi)^3 + N_{\mathcal{I}}^\pi \cdot N_{\mathcal{I}}^G)$.

6.2 Branch and Bound

We are finally ready to describe our B&B-based algorithm (Alg. 3). This recursive algorithm is given a path $\pi = \langle v_0, \ldots, v_k \rangle$ (initialized to the start vertex v_0) and returns the maximal reward obtainable by any path whose prefix is π. The algorithm starts by checking if the path can be traversed in $t < 1$ time (Line 1). If so, it runs the OTP algorithm to compute π's optimal reward (Line 3). Subsequently, it checks if any sub-path appended to π can improve the currently-stored best reward $R_{\max}$. This is done by checking if $\mathcal{UB}(\pi)$ is greater than $R_{\max}$ (Line 4). If this is the case, the algorithm iterates over all adjacent vertices to π's last vertex (Lines 6-9). For every such vertex v_{k+1} the algorithm appends v_{k+1} to π (Line 7), and performs a recursive call to compute the maximal reward obtainable from any path extending the new path (Line 8). It then updates the maximal reward if needed (Line 9). Finally, it returns the overall maximal reward obtained (Line 10).

Theorem 4. *Given a task-assistance graph $G = (V, E)$, an interval mapping $\mathcal{I}$ and a vertex $v_0 \in V$, Alg 3 solves the corresponding OPTP problem (Prob. 2).*

To improve the algorithm's runtime, we apply several optimizations:

Cycle pruning. In contrast to the shortest-path problem, cycles may be beneficial in our setting: They allow to return to previously-visited vertices in future timesteps to make use of multiple intervals associated with the same vertex. However, if a cycle contains only vertices that do not contain intervals, we remove the cycle from the graph. Note that this does not require preprocessing and can be done during the algorithm in $\mathcal{O}(1)$ time using some additional bookkeeping.

Interval splitting. Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path, $\mathbb{T}^*$ its optimal timing profile and $I = [t_s, t_e]$ the last interval $\mathbb{T}^*$ obtains reward from before v_k. Let v_i be the vertex I belongs to, one can show that $\mathcal{UB}(\pi)$ counts I twice since given the pair (t_e, r) from the EXIT list of v_i, it upper bounds the reward obtainable from that

Algorithm 3 Branch and Bound (B&B)

Input: graph: $G = (V, E)$; intervals: $\mathcal{I}$
path: $\pi = \langle v_0, \ldots, v_k \rangle$; // initialized to $\pi \leftarrow \langle v_0 \rangle$
reward: $R_{\max}$ // initialized to $R_{\max} \leftarrow 0$
Output: reward of optimal timing profile of π appended with best subpath

1: **if** $\ell^+(v_k) > 1$ **then** // minimal time to reach the end of π
2: **return** 0 // last vertex is not reachable
3: $R_{\max} \leftarrow \max\{R_{\max}, \mathbf{OTP}(\pi, \mathcal{I})\}$ // run OTP
4: **if** $\mathcal{UB}(\pi) \leq R_{\max}$ **then**
5: **return** $R_{\max}$ // prune all search space containing paths appended to π
6: **for each** v_{k+1} s.t. $(v_k, v_{k+1}) \in E$ **do**
7: $\pi' \leftarrow \langle v_0, \ldots, v_k, v_{k+1} \rangle$
8: $R \leftarrow \text{B\&B}(G, \mathcal{I}, \pi', R_{\max})$ // recursive call
9: $R_{\max} \leftarrow \max\{R_{\max}, R\}$
10: **return** $R_{\max}$

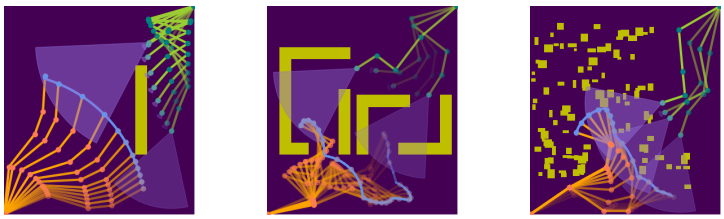
pair using $\mathcal{UB}(v_i, t_s)$ instead of $\mathcal{UB}(v_i, t_e)$ which allows for I to be counted once in the reward r and once in $\mathcal{UB}(v_i, t_s)$. Thus, the smaller the intervals are, the tighter $\mathcal{UB}(\pi)$ is. Thus, we introduce a parameter $\delta_{\max}$. Before running our algorithm, we split every interval of size larger than $\delta_{\max}$ to a sequence of intervals, each of size less than $\delta_{\max}$. Note that (i) as $\delta_{\max}$ decreases, $\mathcal{UB}(\pi)$ is tighter but the number of intervals increases which causes a cubic increase in the complexity of the OTP algorithm (see Sec. 4.2) which is used when computing $\mathcal{UB}(\pi)$ and (ii) this optimization does not affect the solution to the OPTP problem.

Upper bound caching & filtering. The computational bottleneck for computing $\mathcal{UB}(\pi)$ for some path π is the frequent calls to $\mathcal{UB}(u, t)$ (once for every interval in the path). Moreover, for every $\Delta t > 0$ it holds that $\mathcal{UB}(u, t) \geq \mathcal{UB}(u, t + \Delta t)$. Thus, every time $\mathcal{UB}(\pi, t)$ is computed for some π and t, this value is stored. If, in some subsequent iteration the algorithms requires to test if the value $\mathcal{UB}(u, t + \Delta t) < \tau$ for some $\Delta t > 0$ and some τ, we first test if $\mathcal{UB}(u, t) < \tau$. Only if this is not the case, we compute and store $\mathcal{UB}(u, t + \Delta t)$.

Bounded sub-optimality We introduce a second hyper-parameter $\varepsilon \geq 0$ and replace the condition $\mathcal{UB}(\pi) \leq R_{\max}$ in Line 4 with $\mathcal{UB}(\pi) \leq (1 + \varepsilon) \cdot R_{\max}$. This allows to dramatically prune the search space and one can easily show that if R_ε and R^* are the rewards obtained with this variation and the optimal reward, respectively, then $R_\varepsilon \geq R^*/(1 + \varepsilon)$.

7 Empirical Evaluation

To evaluate our B&B-based algorithm (Alg. 3) we consider the TAP-problem of visual assistance. Specifically, the assistance robot is equipped with a camera that should keep the end-effector of the task robot as-visible as possible.



(a) (b) (c)

Fig. 4: Simulated environments consisting of a task-robot (orange), assistance-robot (green) and obstacles (yellow). The task-robot end-effector follows a pre-defined path (blue) and needs to be located in the field of view of a limited-range camera located on the assistance robot's end effector (purple).

We consider a simulated environment in which both $\mathsf{R}_{\mathbf{assist}}$ and $\mathsf{R}_{\mathbf{task}}$ are four-link planar manipulators (Fig. 4) as well as the setting depicted in Fig. 1 wherein $\mathsf{R}_{\mathbf{assist}}$ is a UR5 and $\mathsf{R}_{\mathbf{task}}$ is a UR3 [1]. In each scenario, we fix the path of $\mathsf{R}_{\mathbf{task}}$ and generate a roadmap G using the RRG algorithm [16] containing between 10 to $1,000$ vertices (G was stored with increments of ten vertices).

As we optimize our B&B-based algorithm by employing interval splitting and by allowing for bounded sub-optimality, we present our B&B algorithm with two parameters indicating the interval size $\delta_{\max}$ used for interval splitting and the approximation factor ε. As we run our algorithm on graphs generated by adding additional vertices and edges, when running the algorithm on a graph with n vertices, we use the reward obtained from the previous iteration (on the graph with $n - 10$ vertices) to initialize $R_{\max}$. This allows our algorithm to get close to zero runtimes on iterations where the added vertices can not increase the maximal reward. Consequentially, we present accumulated runtimes.

As baselines to compare with, we suggest the following strawman algorithms: The first, which we term "δ-discretization" (DD(δ)) discretizes the time into steps of size δ. It runs a best-first search where nodes are pairs consisting of a vertex and a time with the initial node being $\langle v_0, 0\rangle$ (i.e., the start vertex and time $t = 0$). When expanding a node $\langle u, t\rangle$ it can either stay at u for δ time (resulting in the node $\langle u, t + \delta\rangle$) or move to a vertex v such that $(u, v) \in E$. Note that this algorithm does not guarantee to compute optimal solutions. The second algorithm, which we term DFS-OTP iterates over all possible paths in the graph in a DFS-like approach (stopping when the length of the path exceeds 1). When reaching the final vertex of a path, it runs the OTP algorithm on the path. Note that this algorithm is optimal and is in fact identical to the B&B approach with a trivial bound of 1 as it iterates over all paths in the graph.

All algorithms were implemented in C++ and run on a Linux, x86_64 server using an Intel Xeon CPU at 2.10GHz with a timeout of one hour. Code will be made publicly available upon paper acceptance.

For each one of the four scenarios (three planar manipulators in Fig. 4 and the UR from Fig. 1), we fixed the path of the task robot and generated ten roadmaps.

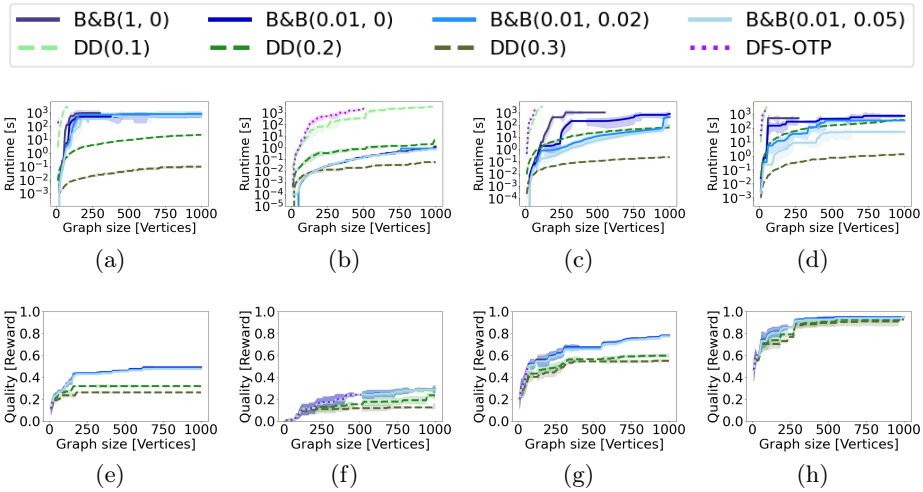


Fig. 5: Running time and quality of solution as a function of graph size for variants of our algorithm and the two baselines. Here, plots (a)-(c) and (e)-(g) correspond to the environments depicted in Fig. 4a-4c, respectively while (d) and (h) correspond to the environment depicted in Fig. 1. Results are averaged over ten roadmaps with the shaded part corresponding to one standard deviation.

We report in Fig. 5 the average running time and the reward for each algorithm as a function of the number of graph vertices. We present four versions of our B&B algorithm, one with $\delta_{\max} = 1, \varepsilon = 0$ (i.e., an optimal algorithm without interval splitting) and three with the same value for $\delta_{\max}$ (for interval splitting) but different values of ε (the approximation factor) as well as DD(δ) with three different values of δ (smaller δ values are expected to run longer but obtain higher-quality results) and the DFS-OTP algorithm.

When compared to the baseline optimal algorithm (DFS-OPT), our algorithm allows for an improved runtime by roughly three orders of magnitude for a given graph size and allows to compute solutions for far larger graphs within the allotted planning time of one hour. When compared to the baseline heuristic algorithm (DD(δ)), while being slower, our algorithm obtains higher-quality results by a factor of up to 3× on the three planar scenarios. For the URs, the problem is much easier containing fewer intervals, thus both algorithms produce comparable results (though ours provides guarantees on the solution quality).

As depicted in Fig. 5, $\delta_{\max}$ and ε can have a large effect on the performance of our B&B algorithm. Roughly speaking, $\delta_{\max}$ balances how much time each OTP call takes versus how tight the upper bound is, and ε reduces the search space size. Recall (Sec. 6), that when computing $\mathcal{UB}(\pi)$, the main gap between the real reward obtainable and the upper bound comes from allowing one interval to be counted twice. Thus, smaller intervals result in a smaller gap. However, $\delta_{\max}$ results in more intervals which increases the runtime of the OTP algorithm. If the approximation factor ε is larger than the aforementioned gap, it will allow

the algorithm to explore only paths whose quality is significantly better that the incumbent solution.

Finally, we include in the supplementary material a video of UR robots in our lab running our algorithm on a scenario mimicking the one in Fig. 1.

8 Future Work

In this work we lay the algorithm foundation for TAP. Here, we assumed that (**A1**) the roadmap G is provided and that (**A2**) the path of $\mathsf{R}_{\mathbf{assist}}$ is known. In future work we wish to relax both assumptions. To relax **A1** we intend to build in RRT-like algorithm that reasons about where to sample while taking the task timing into account. To relax **A2** we suggest to estimate the path of $\mathsf{R}_{\mathbf{assist}}$ using some learning algorithm and then iteratively run our B&B algorithm in a model-predictive control (MPC) fashion.

Acknowledgments

We wish to thank Shaull Almagor for his assistance in proving the hardness of the OTPT problem (Sec. 5) and Dan Elbaz and Ofek Gottlieb for their assistance in the empirical evaluation (Sec. 7).

References

1. Universal robots, https://www.universal-robots.com
2. Almadhoun, R., Taha, T., Seneviratne, L., Dias, J., Cai, G.: A survey on inspecting structures using robotic systems. I. J. Advanced Robotics Systems **13**(6) (2016)
3. Amuzig, A., Dovrat, D., Keren, S.: Value of assistance for mobile agents. CoRR **abs/2308.11961** (2023)
4. Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., Siegwart, R.: Receding horizon next-best-view planner for 3d exploration. In: ICRA. pp. 1462–1468. IEEE (2016)
5. Chung, T.H., Hollinger, G.A., Isler, V.: Search and pursuit-evasion in mobile robotics - A survey. Auton. Robots **31**(4), 299–316 (2011)
6. Clausen, J.: Branch and bound algorithms-principles and examples (2003)
7. Connolly, C.: The determination of next best views. In: ICRA. vol. 2, pp. 432–435 (1985)
8. Fu, M., Kuntz, A., Salzman, O., Alterovitz, R.: Toward asymptotically-optimal inspection planning via efficient near-optimal graph search. In: RSS (2019)
9. Fu, M., Salzman, O., Alterovitz, R.: Computationally-efficient roadmap-based inspection planning via incremental lazy search. In: ICRA. pp. 7449–7456 (2021)
10. Galceran, E., Carreras, M.: A survey on coverage path planning for robotics. Robotics and Autonomous systems **61**(12), 1258–1276 (2013)
11. Geraerts, R.: Camera planning in virtual environments using the corridor map method. In: Motion in Games (MIG). vol. 5884, pp. 194–206 (2009)
12. Goemans, O.C., Overmars, M.H.: Automatic generation of camera motion to track a moving guide. In: WAFR. vol. 17, pp. 187–202 (2004)

13. Halperin, D., Salzman, O., Sharir, M.: Algorithmic motion planning. In: Csaba D. Toth, Joseph O'Rourke, J.E.G. (ed.) Handbook of Discrete and Computational Geometry, chap. 50, pp. 1307–1338. CRC Press, Inc., 3rd edn. (2017)
14. Hauser, K.: Motion and Path Planning, pp. 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg (2020)
15. Howard, R.A.: Information value theory. IEEE Transactions on systems science and cybernetics **2**(1), 22–26 (1966)
16. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. of Rob. Res. **30**(7), 846–894 (2011)
17. Kleinberg, J., Tardos, E.: Algorithm design. Pearson Education India (2006)
18. Laguna, G.J., Bhattacharya, S.: Path planning with incremental roadmap update for visibility-based target tracking. In: IROS. pp. 1159–1164 (2019)
19. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
20. LaValle, S.M., González-Baños, H.H., Becker, C., Latombe, J.: Motion strategies for maintaining visibility of a moving target. In: ICRA. pp. 731–736. IEEE (1997)
21. Masarwy, M., Goshen, Y., Dovrat, D., Keren, S.: Value of assistance for grasping (2023)
22. Nieuwenhuisen, D., Overmars, M.H.: Motion planning for camera movements. In: ICRA. pp. 3870–3876 (2004)
23. O'Rourke, J.: Visibility. In: Csaba D. Toth, Joseph O'Rourke, J.E.G. (ed.) Handbook of Discrete and Computational Geometry, chap. 33, pp. 875–896. CRC Press, Inc., 3rd edn. (2017)
24. Rakita, D., Mutlu, B., Gleicher, M.: An autonomous dynamic camera method for effective remote teleoperation. In: HRI. pp. 325–333 (2018)
25. Roberts, M., Leboutet, Q., Prakash, R., Wang, R., Zhang, H., Tang, R., Ferragut, M., Leutenegger, S., Richter, S.R., Koltun, V., Müller, M., Ros, G.: SPEAR: A simulator for photorealistic embodied ai research (2022)
26. Robin, C., Lacroix, S.: Multi-robot Target Detection and Tracking: Taxonomy and Survey. Autonomous Robots **40**(4), 729–760 (2016)
27. Russell, S., Wefald, E.: Principles of metareasoning. Artificial intelligence **49**(1-3), 361–395 (1991)
28. Salzman, O.: Sampling-based robot motion planning. Commun. ACM **62**(10), 54–63 (2019)
29. Salzman, O., Felner, A., Hernández, C., Zhang, H., Chan, S., Koenig, S.: Heuristic-search approaches for the multi-objective shortest-path problem: Progress and research opportunities. In: ijcai. pp. 6759–6768 (2023)
30. Solovey, K.: Complexity of Planning, pp. 1–8. Springer Berlin Heidelberg, Berlin, Heidelberg (2020)
31. Zilberstein, S., Lesser, V.: Intelligent information gathering using decision models. CS Department, U. of Massachusetts, Boston, Massachusetts (1996)

A OPT—Proof of Thm. 1

Theorem 1. *For any path $\pi = \langle v_0, \ldots, v_k \rangle$, there exists an optimal timing profile $\mathbb{T}_\pi = \langle t_0, \ldots, t_{k-1} \rangle$ such that $\forall i : t_i \in C_{T_i}$.*

Proof. To prove the theorem, we will show that any optimal timing profile can be transformed to one that contains vertex-critical times only.

Let $\mathbb{T}^* = \langle t_0^*, \ldots, t_{k-1}^* \rangle$ be an optimal timing-profile for π, and let i be the first index such that $t_i^* \notin \text{C}_{T_i}$. We describe a procedure in which we create another timing-profile $\mathbb{T}' = \langle t_0', \ldots, t_{k-1}' \rangle$ such that (i) $\mathcal{R}(\pi, \mathbb{T}') \geq \mathcal{R}(\pi, \mathbb{T}^*)$ and that (ii) $t_j' = t_j^*$ for all $j < i$ and $t_i' \in \text{C}_{T_i}$. Consequently, we can take an optimal timing-profile and run this procedure until an optimal timing-profile exists s.t. $\forall i \; t_i^* \in \text{C}_{T_i}$ (this takes at most k iterations).

Our procedure considers two cases: Either there is or there isn't an interval $I \in \mathcal{I}(v_i)$ such that $t_i^* \in I$.

Case 1: There is an interval $I \in \mathcal{I}(v_i)$ such that $t_i^* \in I$.

Here, we consider two different sub-cases corresponding to whether I is the last interval that $\mathbb{T}^*$ obtains reward from or not.

1.1: I is the last interval that $\mathbb{T}^*$ obtains reward from.
Since I is the last interval $\mathbb{T}^*$ obtains reward from, $\mathbb{T}'$ can leave later without losing any reward. Recall (Sec. 4) the last vertex on the path contains the interval $[1-\ell^+(v_{k-1}, v_k), 1-\ell^+(v_{k-1}, v_k)]$, thus $1-\ell^+(v_{k-1}, v_k) \in \text{C}_{T_k}$. From Def. 1, it holds that $t_k^* \leq 1-\ell^+(v_{k-1}, v_k)$, thus, by setting $t_j' := 1-\ell^+(v_j, v_k)$ for every $j \geq i$ we get that $t_j' \in \text{C}_{T_j}$ (Obs. 2) and $t_j' \geq t_j^*$. Hence, if we define $\mathbb{T}' = \langle t_0^*, \ldots t_{i-1}^*, t_i', \ldots, t_k' \rangle$ it holds that:

$$\mathcal{R}(\pi, \mathbb{T}') - \mathcal{R}(\pi, \mathbb{T}^*) = \mathcal{R}(\pi, \mathbb{T}', t_i^*) - \mathcal{R}(\pi, \mathbb{T}^*, t_i^*) = \mathcal{R}(\pi, \mathbb{T}', t_i^*) - 0 \geq 0.$$

1.2: I is not the last interval that $\mathbb{T}^*$ obtains reward from.
Let $I = [t_s, t_e] \in \mathcal{I}(v_i)$ be an interval such that $t_i^* \in I$, and I is not the last interval $\mathbb{T}^*$ obtains reward from. Let $I' = [t_s', t_e'] \in \mathcal{I}(v_j)$ be the next interval $\mathbb{T}^*$ obtains reward from after I. Note that $v_i \prec v_j$. Here we distinguish between the setting where $t_{j-1}^* \in (t_s, t_e]$ and $t_{j-1}^* \notin (t_s, t_e]$:

1.2.1: $t_{j-1}^* \in (t_s, t_e]$.
Here, we will change t_i' to leave vertex v_i earlier than t_i^*. Intuitively, $\mathbb{T}'$ will lose some reward from I but, it will earn the same reward back from I'. First, we show that $t_{j-1}^* = t_i^* + \ell^+(v_i, v_{j-1})$. Assume by contradiction that $t_{j-1}^* \neq t_i^* + \ell^+(v_i, v_{j-1})$, thus $t_{j-1}^* > t_i^* + \ell^+(v_i, v_{j-1})$. Set $t_m' := t_i' + \ell^+(v_i, v_m)$ for $i < m \leq j-1$. It holds that:

$$\begin{aligned}
\mathcal{R}(\pi, \mathbb{T}') - \mathcal{R}(\pi, \mathbb{T}^*) &= \sum_{m=i}^{j+1} \mathcal{R}(v_m, t_{m-1}', t_m') - \sum_{m=i}^{j+1} \mathcal{R}(v_m, t_{m-1}^*, t_m^*) \\
&= 0 + \mathcal{R}(v_j, t_{j-1}', t_j^*) - (0 + \mathcal{R}(v_j, t_{j-1}^*, t_j^*)) \\
&= \mathcal{R}(v_j, t_{j-1}', t_{j-1}^*) + \mathcal{R}(v_j, t_{j-1}^*, t_j^*) - \mathcal{R}(v_j, t_{j-1}^*, t_j^*) \\
&= \mathcal{R}(v_j, t_{j-1}', t_{j-1}^*) \\
&= t_{j-1}^* - \max\{t_{j-1}', s\} > 0.
\end{aligned}$$

Which means that $\mathcal{R}(\pi, \mathbb{T}') > \mathcal{R}(\pi, \mathbb{T}^*)$ which contradicts the optimality of $\mathbb{T}^*$. Thus indeed, $t^*_{j-1} = t^*_i + \ell^+(v_i, v_{j-1})$.
Next, we will look at $\hat{t}_1 := t^*_{i-1} + \ell^+(v_{i-1}, v_i)$, which is the earliest time $\mathbb{T}'$ can leave v_i, and at $\hat{t}_2 := t'_s - \ell^+(v_i, v_{j-1})$, which is the latest time $\mathbb{T}'$ can leave v_i to reach v_j at time t'_s. Set $t'_i := \max\{\hat{t}_1, \hat{t}_2\}$, from Obs. 1 and 2 it holds that $t'_i \in \text{C}\text{T}_i$, and since $t^*_{j-1} = t^*_i + \ell^+(v_i, v_{j-1})$ and $t^*_j > t_s$ it holds that $t'_i \leq t^*_i$. Now set $t'_m := t'_i + \ell^+(v_i, v_m)$ for $i < m \leq j-1$, and $\mathbb{T}' = \langle t^*_1, \dots t^*_{i-1}, t'_i, \dots t'_{j-1}, t^*_j, \dots, t^*_k \rangle$. It holds that:

$$\begin{aligned}
\mathcal{R}(\pi, \mathbb{T}') - \mathcal{R}(\pi, \mathbb{T}^*) &= \sum_{m=i}^{j+1} \mathcal{R}(v_m, t'_{m-1}, t'_m) - \sum_{m=i}^{j+1} \mathcal{R}(v_m, t^*_{m-1}, t^*_m) \\
&= \mathcal{R}(v_i, t'_{i-1}, t'_i) + \mathcal{R}(v_j, t'_{j-1}, t'_j) - \mathcal{R}(v_i, t^*_{i-1}, t^*_i) - \mathcal{R}(v_j, t^*_{j-1}, t^*_j) \\
&= -\mathcal{R}(v_i, t'_i, t^*_i) + \mathcal{R}(v_j, t'_{j-1}, t^*_{j-1}) \\
&= -(t^*_i - t'_i) + t^*_{j-1} - t'_{j-1} \\
&= t'_i - t^*_i + t^*_i + \ell^+(v_i, v_{j-1}) - t'_i - \ell^+(v_i, v_{j-1}) = 0
\end{aligned}$$

1.2.2: $t^*_{j-1} \notin (t_s, t_e]$.
Since $\mathbb{T}'$ obtains reward from I', it must hold that $t^*_{j-1} \leq t_s$. We look at $\hat{t}_1 := t_s - \ell^+(v_i, v_{j-1})$ which is the latest time $\mathbb{T}'$ can leave v_i to arrive to v_j at time t'_s, and $\hat{t}_2 := t^*_j - \ell^+(v_i, v_j)$, which is the latest time $\mathbb{T}'$ can leave v_i to be able to leave v_j at time t^*_j. Set $t'_i := \min\{\hat{t}_1, \hat{t}_2\}$, from Obs. 2 it holds that $t'_i \in \text{C}\text{T}_i$ and from its definition, it holds that $t'_i \geq t^*_i$. Set $t'_m := t'_i + \ell^+(v_i, v_m)$ for $i < m \leq j-1$, and $\mathbb{T}' = \langle t^*_1, \dots t^*_{i-1}, t'_i, \dots t'_{j-1}, t^*_j, \dots, t^*_k \rangle$. It holds that:

$$\begin{aligned}
\mathcal{R}(\pi, \mathbb{T}') - \mathcal{R}(\pi, \mathbb{T}^*) &= \sum_{m=i}^{j+1} \mathcal{R}(v_m, t'_{m-1}, t'_m) - \sum_{m=i}^{j+1} \mathcal{R}(v_m, t^*_{m-1}, t^*_m) \\
&= \mathcal{R}(v_i, t'_{i-1}, t'_i) - \mathcal{R}(v_i, t^*_{i-1}, t^*_i) \\
&= \mathcal{R}(v_i, t^*_{i-1}, t^*_i) + \mathcal{R}(v_i, t^*_i, t'_i) - \mathcal{R}(v_i, t^*_{i-1}, t^*_i) \\
&= \mathcal{R}(v_i, t^*_i, t'_i) \geq 0
\end{aligned}$$

Case 2: There is no interval $I \in \mathcal{I}(v_i)$ such that $t^*_i \in I$.

Since t^*_i is not part of an interval, $\mathbb{T}'$ can leave vertex v_i earlier without losing any reward. The two times that are of interest are the earliest time we can leave v_i given we entered at time t^*_{i-1}, and the end time of the last interval of v_i before t^*_i. By definition, the earliest time $\mathbb{T}'$ can leave vertex v_i is $\hat{t}_1 := t^*_{i-1} + \ell^+(v_{i-1}, v_i)$. Since $\hat{t}_1$ is the earliest time $\mathbb{T}'$ can leave v_i it holds that $\hat{t}_1 \leq t^*_i$ and since $t^*_{i-1} \in \text{C}\text{T}_{i-1}$ it holds that $t'_i \in \text{C}\text{T}_i$ (Obs. 2). If v_i has at least one interval that ends before t^*_i starts, we set $\hat{t}_2$ to be the end time of the last interval ending before t^*_i. It holds that $\hat{t}_2 \leq t^*_i$ and from Def. 5 it holds that $\hat{t}_2 \in \text{C}\text{T}_i$. If v_i has no such interval we set $\hat{t}_2 := \hat{t}_1$. We set $t' := \max\{\hat{t}_1, \hat{t}_2\}$ and $\mathbb{T}' = \langle t^*_0, \dots, t^*_{i-1}, t', t^*_{i+1}, \dots, t^*_{k-1} \rangle$. It holds that:

$$\mathcal{R}(\pi, \mathbb{T}') - \mathcal{R}(\pi, \mathbb{T}^*) = \mathcal{R}(v_{i+1}, t', t^*_i) - \mathcal{R}(v_i, t', t^*_i) = \mathcal{R}(v_{i+1}, t', t^*_i) - 0 \geq 0.$$

B OPTP Hardness—Proof of Thm. 2

Theorem 2. *The OPTP problem is* NP*-hard.*

The proof is by a reduction from the subset-sum problem (SSP) [17]. Recall that the SSP is a decision problem where we are given a set $X = \{x_1, \ldots, x_n \mid x_i \in \mathbb{N}\}$ and a target $K \in \mathbb{N}^+$ (for simplicity, we assume that $K > 0$ but this is a technicality only). The problem calls for deciding whether there exists a set $X' \subseteq X$ whose sum $\sum_{x \in X'} x$ equals K.

Given an SSP instance, we build a corresponding OPTP instance (to be explained shortly) and show that there exists a subset $X' \subseteq X$ such that $\sum_{x \in X'} x = K$ iff the optimal reward in our OPTP instance is $\frac{K}{\sum_{i=1}^{n} x_i} + 1$ thus proving the problem is NP-hard.

W.l.o.g. assume that $x_1, \ldots, x_n$ are sorted from smallest to greatest and set $\kappa_i := \sum_{j=1}^{i} x_j$. $\alpha_i := x_i/\kappa_n$ and $y_i = \sum_{j=1}^{i-1} \kappa_j$. The graph of our OPTP instance depicted in Fig. 3 contains n hexagons $H_1, \ldots H_n$ such that H_i contains vertices $a_i, b_i, c_i, d_i, e_i, f_i$. We call a_i and f_i the entry and exit vertices of H_i, b_i and c_i the top of H_i and d_i and e_i the bottom of H_i. Edges (b_i, c_i) and (d_i, e_i) at the top and bottom of H_i have length κ_i and κ_{i-1}, respectively (all other edge lengths are equal to zero). Vertices b_i, c_i at the top of H_i contains the interval $[y_i, y_i + \alpha_i]$. Finally, the exit f_n of the last hexagon H_n has an edge to a vertex u which has a valid interval $[0, y_n + K]$ and u has an edge to a vertex v whose assistance interval is $[y_n + K, y_n + K + 1]$. Note that here for ease of exposition, time is in the range $[0, y_n + K + 1]$ and *not* $[0, 1]$.

Before we prove Thm. 2, we state several observations and Lemmas.

Observation 4. *Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path and $\mathbb{T} = \langle t_0, \ldots, t_{k-1} \rangle$ a timing profile. If $t_i = t_{i-1} + \ell^+(v_{i-1}, v_i)$ then π does not wait at any vertex (except maybe the last one).*

Observation 5. *By choosing to go through the lower part of all hexagons (which is the fastest way to reach u) a timing profile can reach u at time y_n*

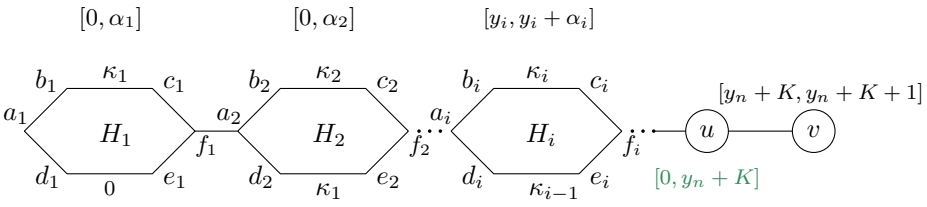


Fig. 3: Reduction graph (all edges are directed from left to right). When omitted, edge length equals zero. (This figure is identical to Fig. 3 in Sec. 5 and is added to make the appendix self-contained).

To see why Obs. 5 holds, let $\mathbb{T} = \langle t_0, t_1, \ldots, t_{k-1} \rangle$ such that $t_0 = 0$, and $t_i = t_{i-1} + \ell^+(v_{i-1}, v_i)$. Now we have that indeed,

$$t_{k-1} = t_0 + \sum_{i=1}^{k-1} \ell^+(v_{i-1}, v_i) = \sum_{i=1}^{n-1} \kappa_i = \sum_{i=1}^{n-1} \sum_{j=1}^{i} x_j = y_n.$$

Observation 6. *Let π be a path, for every hexagon H_i where π goes through the upper part, it adds an additional x_i to the time it takes to reach u.*

Lemma 4. *Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path and $\mathbb{T} = \langle t_0, \ldots, t_{k-1} \rangle$ a timing profile such that $t_i = t_{i-1} + \ell^+(v_{i-1}, v_i)$. The reward obtained from choosing the upper part of hexagon H_i is exactly α_i.*

Proof. We starting by proving via induction over i that π can reach hexagon H_i (i.e., reach a_i) only between $[y_{i-1}, y_i]$ (we set y_0 and y_1 to be 0).

Base ($i = 1$): Since π starts at the beginning of the first hexagon H_1 it reaches a_1 at time 0 and indeed $0 \in [0, 0] = [y_0, y_1]$.

Step: We assume that the induction hypothesis holds for hexagon H_i and we will prove it for hexagon H_{i+1}. π can either go through the lower or upper part of hexagon H_i, and following Obs. 4 we know that π can't wait at any vertex of H_i. If π goes through the lower part of hexagon H_i, it will reach hexagon H_{i+1} between $[y_{i-1} + \kappa_{i-1}, y_i + \kappa_{i-1}] = [y_i, y_{i+1} - x_i]$. If π goes through the upper part of H_i, it will reach H_{i+1} between $[y_{i-1} + \kappa_i, y_i + \kappa_i] = [y_i + x_i, y_{i+1}]$. Taking a union over both options, we get that π will reach H_{i+1} between $[y_i, y_{i+1} - x_i] \cup [y_i + x_i, y_{i+1}] = [y_i, y_{i+1}]$ which completes our induction.

Now we can complete the proof of Lemma 4. If π traverses the upper part of hexagon H_i, it will reach b_i between $[y_{i-1}, y_i]$. Thus, it will reach c_i between $[y_{i-1} + \kappa_i, y_i + \kappa_i] = [y_i + x_i, y_{i+1}]$ which means π must be on the edge (b_i, c_i) between $[y_i, y_i + x_i]$ (since π never waits at a vertex). In addition $\alpha_i = x_i/\kappa_i \leq 1 \leq x_i$ thus $[y_i, y_i + \alpha_i] \subseteq [y_i, y_i + x_i]$ and since both b_i and c_i have an interval between $[y_i, y_i + \alpha_i]$ we get that π must obtain a reward of α_i. Since this is the only interval in H_i, π obtains a reward of exactly α_i.

Lemma 5. *The maximal reward obtainable for any path π before reaching vertex v is bounded by* 1.

Proof. The maximal reward that can be obtained without reaching vertex v can be upper bounded by the union of all the intervals that do not belong to v. Let $R^{\not v}_{\max}$ be the maximal reward obtainable without reaching v, it holds that:

$$R^{\not v}_{\max} \leq \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} x_i/\kappa_n = \frac{1}{\kappa_n} \cdot \sum_{i=1}^{n} x_i = 1.$$

Lemma 6. *The maximal reward attainable is exactly $1 + K/\kappa_n$.*

Proof. Let π be an optimal path. From Lemma 5 we know that π must reach v, and since there is a valid interval on vertex u, π must reach it before $y_n + K$.

Let X' be the set of items represented by the hexagons at which π chose the upper part, formally: $X' = \{x_i \mid b_i \in \pi\}$. Following Obs. 5, 6 we get that: $\sum_{x_i \in X'} x_i \leq K$. From Lemma 4 we get that the reward obtained from choosing the upper part of hexagon H_i is x_i/κ_n therefore the maximum reward that can be obtained from the hexagons is K/κ_n. We can add the reward we obtain at vertex v and get that the maximal reward is $1 + K/\kappa_n$.

We are finally ready to prove our theorem.

Proof. Let $X' \subseteq X$ be a subset such that $\sum_{x_i \in X'} x_i = K$. We build our path π as follows: for each i, if $x_i \in X'$ append $\langle a_i, b_i, c_i \rangle$ to the path else append $\langle a_i, d_i, e_i \rangle$. Finally, append u, v to the end of the path. Next, we build or timing profile $\mathbb{T} = \langle t_0, \ldots, t_{k-1} \rangle$ by setting $t_0 := 0$, and $t_i := t_{i-1} + \ell^+(v_{i-1}, v_i)$. We start by showing that we arrive at vertex v at time $y_n + K$:

$$\begin{aligned} t_{k-1} &= t_0 + \sum_{i=1}^{k-1} \ell^+(v_{i-1}, v_i) \\ &= \sum_{x_i \in X'} \kappa_i + \sum_{x_i \notin X'} \kappa_{i-1} \\ &= \sum_{i=1}^{n} \kappa_{i-1} + \sum_{x_i \in X'} x_i \\ &= \sum_{i=1}^{n-1} \kappa_i + \sum_{x_i \in X'} x_i = y_n + K \end{aligned}$$

Thus, our path and timing profile obtain a reward of 1 in the time interval $[y_n + K, y_n + K + 1]$. We can now analyze the reward obtained during the time interval $[0, y_n + K]$. From Lemma 4, we get that the reward obtained during the time $[0, y_n + K]$ is equal to:

$$\begin{aligned} \sum_{x_i \in X'} \alpha_i &= \sum_{x_i \in X'} x_i/\kappa_i \\ &= \frac{1}{\kappa_n} \cdot \sum_{x_i \in X'} x_i \\ &= K/\kappa_n \end{aligned}$$

Thus, the total reward of our path and timing profile is $1 + K/\kappa_n$ and from Lemma 6 we know it is the maximal reward thus our path and timing profile are optimal.

Now, let $\pi, \mathbb{T}$ be an optimal path and timing profile such that $\mathcal{R}(\pi, \mathbb{T}) = 1 + K/\kappa_n$. From Lemma 5 we know that π must reach vertex v, and since it is possible to reach v only before $y_n + K$ we know that π earns a reward

of 1 at v. Thus, it must earn a reward of K/κ_n until v. Let X' be the set of items represented by the hexagons at which π chose the upper part. Formally, $X' = \{x_i \mid b_i \in \pi\}$. From Lemma 4 we know that the reward obtained from choosing the upper part of hexagon i is exactly x_i/κ_n thus:

$$K/\kappa_n = \sum_{x_i \in X'} x_i/\kappa_n$$
$$\Rightarrow K = \sum_{x_i \in X'} x_i,$$

which completes our reduction.

C OPTP Upper Bounds—Additional Details

We start by formally defining $\delta'(u, v)$. We then describe in detail our algorithm used to compute $\mathcal{UB}_I$ which was briefly described in Sec 6.1. We then proceed to prove Lemmas 2, 3 and Thm. 3.

Definition 7. *Let $G = (V, E)$ be a task-assistance graph. We define $\delta'(u, v)$ to be the minimum distance between u and v in G while not accounting for half of the first and last edge. Namely, if $\Pi(u, v)$ is the set of all paths from u to v then*

$$\delta'(u, v) := \min_{\pi \in \Pi(u,v)} \{\ell^-_\pi(v) - \ell^+_\pi(u)\}$$

if $\Pi(u, v) \neq \emptyset$ and ∞ if no such path exists.

Alg. 4 provides the pseudo code to compute $\mathcal{UB}_I$. It starts by initializing IntervalSet to be a set containing all intervals, $\mathcal{UB}_I$ to be the length of interval I and $N^G_{\mathcal{I}}$ to be the number of intervals in G (Lines 1-6). It then iterates over all intervals $N^G_{\mathcal{I}}$ times (Lines 7-14). For each interval I, the algorithm iterates over all intervals I' and bounds the maximal reward obtainable assuming the next interval after I is I' (Lines 11-14). This is done by computing t_{unused}, which is the portion of either I or I' in which no reward can be obtained when traveling from I to I', and subtracting it from $|I| + \mathcal{UB}_{I'}$. If the bound obtained is greater then the current $\mathcal{UB}_I$, the algorithm updates $\mathcal{UB}_I$ accordingly (Line 14).

Lemma 2. *Let π, $\mathbb{T}$ be a path and timing-profile. Let $I = [t_s, t_e]$ be an interval $\mathbb{T}$ obtains reward from. Then it holds that $\mathcal{UB}_I \geq \mathcal{R}(\pi, \mathbb{T}, t_s)$.*

Proof. Let $I_0, \ldots, I_m$ be the list of intervals $\mathbb{T}$ obtains reward from ordered from last to first. We will prove the lemma by induction over i such that at the end of every iteration i, it holds that $\mathcal{UB}^i_{I_i} \geq \mathcal{R}(\pi, \mathbb{T}, t_s)$ when $I_i = [t_s, t_e]$.

Base ($i = 0$): Since $I_0 = [t_s, t_e]$ is the last interval $\mathbb{T}$ obtains reward from it holds that $\mathcal{R}(\pi, \mathbb{T}, t_s) \leq |I_0| = \mathcal{UB}^0_{I_0}$.

Step: We assume that the induction hypothesis holds for $i-1$ and we will prove for i. Let u be the vertex $I_i = [t_s, t_e]$ belongs to, and v the vertex $I_{i-1} = [t'_s, t'_e]$

Algorithm 4 compute intervals upper bounds

Input: graph $G = (V, E)$; intervals: $\mathcal{I}$
Output: upper bound $\mathcal{UB}_I$ for each interval I

```
1: IntervalSet ← ∅                                   // gather all intervals into a set
2: for u ∈ V do
3:     for I = [t_s, t_e] ∈ I(u) do
4:         IntervalSet.insert(I);
5:         UB_I ← |I|                                 // initialize bound for I
6: N_I^G ← |IntervalSet|
7: loop N_I^G times
8:     for I = [t_s, t_e] ∈ IntervalSet do
9:         u ← I.vertex()                             // vertex that I belongs to
10:        for I' = [t'_s, t'_e] ∈ IntervalSet do
11:            v ← I'.vertex()                        // vertex that I' belongs to
12:            if I' is reachable from I then
13:                t_unused ← max{0, t_e + δ'(u,v) − t'_s}  // time of I, I' that cant be used
14:                UB_I ← max{UB_I, |I| + UB_I' − t_unused}
15: return {UB_I | I ∈ IntervalSet}
```

belongs to. $\mathbb{T}$ can not earn reward for at least $t_{\text{unused}} := \max\{0, t_e + \delta'(u,v) - t'_s\}$ time when moving from u to v. Thus,

$$\mathcal{R}(\pi, \mathbb{T}, t_s) \leq |I| + \mathcal{R}(\pi, \mathbb{T}, t'_s) - w.$$

From the induction assumption we get that

$$\mathcal{R}(\pi, \mathbb{T}, t_s) \leq |I| + \mathcal{UB}^{i-1}_{I_{i-1}} - w.$$

And after the end of the i-th iteration it holds that

$$\mathcal{UB}^{i}_{I_i} \geq |I| + \mathcal{UB}^{i-1}_{I_{i-1}} - w.$$

Thus, $\mathcal{UB}^{i}_{I_i} \geq \mathcal{R}(\pi, \mathbb{T}, t_s)$ which concludes the proof.

Lemma 3. *Let $u \in V$ be a vertex, and $t \in [0, 1]$. For every path π and timing-profile $\mathbb{T}$ s.t. $u \in \pi$ and that following $\mathbb{T}$ implies that at time t the robot is at vertex u, it holds that $\mathcal{UB}(u, t) \geq \mathcal{R}(\pi, \mathbb{T}, t)$.*

Proof. Let $I = [t_s, t_e] \in \mathcal{I}(v)$ for some $v \in V$ be the first interval $\mathbb{T}$ obtains reward from after time t, and let $t' \geq t$ be the time $\mathbb{T}$ enters that interval (or t if $t' < t$). Since $\delta'(u, v)$ is the minimal time with no reward between u and v it holds that $t' \geq t + \delta'(u, v)$. Recall that $t_{\text{unused}} := \max\{0, t + \delta'(u, v) - t_s\}$ is the time that must pass since t_s in which $\mathbb{T}$ can not obtain reward from I, then from Lemma 2 it holds that $\mathcal{R}(\pi, \mathbb{T}, t) \leq \mathcal{UB}_I - t_{\text{unused}}$, And from its definition, it holds that $\mathcal{UB}(u, t) \geq \mathcal{UB}_I - t_{\text{unused}}$, thus, we get that $\mathcal{UB}(u, t) \geq \mathcal{UB}_I - t_{\text{unused}} \geq \mathcal{R}(\pi, \mathbb{T}, t)$.

Theorem 3. *Let $\pi = \langle v_0, \ldots, v_k \rangle$ be a path. For any path $\pi' = \langle v_0, \ldots, v_k, \ldots, v_m \rangle$ extending π s.t., $\mathbb{T}'_{\pi'} = \langle t'_0, \ldots, t'_{m-1} \rangle$ is its optimal timing profile, it holds that $\mathcal{UB}(\pi) \geq \mathcal{R}(\pi', \mathbb{T}'_{\pi'})$.*

Proof. Let $I = [t_s, t_e] \in \mathcal{I}(v_i)$ for some $v \in \pi$ be the last interval $\mathbb{T}'$ obtains reward from before leaving v_k. We separate the reward obtained by π''s into two parts: (i) the reward obtained until t'_i (the time $\mathbb{T}'$ exits v_i) and (ii) the reward obtained from t'_i.

Since $v_i \in \pi$, it holds that t_e is a Vertex-critical time of vertex v_i in π. Thus, given the time-reward pair (t_e, r) belonging to the Exit list of v_i when running the OTP algorithm on π[6] it holds that $\mathcal{R}(\pi', \mathbb{T}') - \mathcal{R}(\pi', \mathbb{T}', t_e) \leq r$, and since $\mathcal{R}(\pi', \mathbb{T}', t_e) \leq \mathcal{R}(\pi', \mathbb{T}', t'_i)$ we get that $\mathcal{R}(\pi', \mathbb{T}') - \mathcal{R}(\pi', \mathbb{T}', t'_i) \leq r$.

Additionally, $\mathbb{T}'$ does not obtain reward from t'_i to t'_k, and it holds that $t'_k \geq t'_i + \ell^+(v_i, v_k) \geq t_s + \ell^+(v_i, v_k)$. Thus, from Lemma 3 we get that $\mathcal{R}(\pi', \mathbb{T}', t'_i) \leq \mathcal{UB}\left(v_k, t_s + \ell^+(v_i, v_k)\right)$. Finally:

$$\begin{aligned}\mathcal{R}(\pi', \mathbb{T}') &= \mathcal{R}(\pi', \mathbb{T}') - \mathcal{R}(\pi', \mathbb{T}', t'_i) + \mathcal{R}(\pi', \mathbb{T}', t'_i) \\ &\leq r + \mathcal{UB}\left(v_k, t_s + \ell^+(v_i, v_k)\right) \leq \mathcal{UB}(\pi).\end{aligned}$$

[6] In order for the next part to be correct in the case where $t'_i > t_e$, we must consider entering vertex v_i at time $t'_{i-1} > t_e - \ell^+(v_{i-1}, v_i)$ which is not considered when computing the pair (t_e, r). To do so, we update the "if" statement (Line 3) in `compute_best_reward` (Alg. 2) to be: "if $t_{\text{entry}} > t_{\text{exit}}$" when running the OTP algorithm for $\mathcal{UB}(\pi)$.