Integrating Temporal Reasoning and Sampling-Based Motion Planning for Multi-Goal Problems with Dynamics and Time Windows

Stefan Edelkamp[1], Morteza Lahijanian[2], Daniele Magazzeni[1], and Erion Plaku[3]

***Abstract*—Robots used for inspection, package deliveries, moving of goods, and other logistics operations are often required to visit certain locations within specified time bounds. This gives rise to a challenging problem as it requires not only planning collision-free and dynamically-feasible motions but also reasoning temporally about when and where the robot should be. While significant progress has been made in integrating task and motion planning, there are still no effective approaches for multi-goal motion planning when both dynamics and time windows must be satisfied. To effectively solve this challenging problem, this paper develops an approach that couples temporal planning over a discrete abstraction with sampling-based motion planning over the continuous state space of feasible motions. The discrete abstraction is obtained by imposing a roadmap which captures the connectivity of the free space. At each iteration of a core loop, the approach first invokes the temporal planner to find a solution over the roadmap abstraction. In a second step, the approach uses sampling to expand a motion tree along the regions associated with the discrete solution. Experiments are conducted with second-order ground and aerial vehicle models operating in complex environments. Results demonstrate the efficiency and scalability of the approach as we increase the number of goals and the difficulty of satisfying the time bounds.**

***Index Terms*—Motion and Path Planning, Task Planning, Motion Control, Autonomous Vehicle Navigation.**

I. Introduction

Time is money, and plans that lack temporal constraints are often impractical as robots are increasingly utilized in complex missions in home, social, and industrial applications. Planning in these cases, however, is challenging. First, tasks are complex and highly time sensitive, often requiring a robot to finish the task or reach certain locations within specified time bounds. Second, the robot motions must satisfy the underlying dynamics, which are often nonlinear, noholonomic, and high dimensional. While significant progress has been made in integrating task and motion planning, there has not been any approach for multi-goal motion planning that can effectively take into account both dynamics and time windows. The main contribution of this paper is the first effective approach for this challenging problem.

While there are temporal planners [1], [2] that can handle tight temporal constraints and plan tasks for complex systems [3], [4], it remains difficult to incorporate motion planning with dynamics directly into temporal planners. Some work has been done in this direction by considering an abstract symbolic representation of the continuous state space based on waypoints [5]. While such an approach can work with small problems with simple dynamics, it cannot scale to large problems with high-dimensional continuous state spaces and complex dynamics, which is the focus of this paper. The dynamics give rise to two-boundary value problems which often make it impossible to precisely connect the waypoints [6].

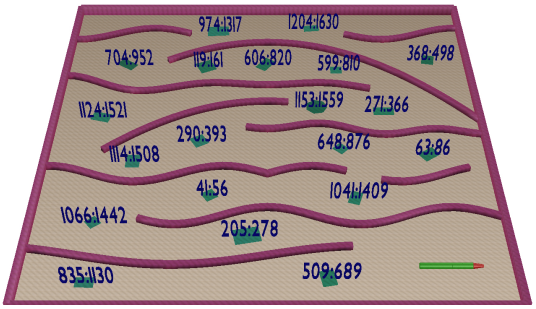


Fig. 1. An example of a multi-goal motion-planning problem with time windows where the snake model is required to reach each goal within a specified time window. Images are best viewed in colors and on screen. Video of solutions obtained by our approach for this and other scenarios can be found at `https://goo.gl/LKrU9Z`

On the other hand, motion planners based on probabilistic sampling have made it possible to explore high-dimensional continuous state spaces, taking into account the obstacles and the robot dynamics [7]. Motion planners such as DROMOS [8] can even plan motions to reach multiple goals by using a TSP solver to guide sampling-based motion planning. DROMOS, however, does not take into account time windows for visiting the goals or any other temporal information.

Incorporating the temporal information is necessary in many logistics applications where the robot has to reach certain locations within specified time windows. Fig. 1 provides an illustration. Time windows make planning harder since planning decisions in the beginning can have detrimental effects later on as trajectories should satisfy the temporal constraints. As an example, the robot may have to go through narrow passages to reduce the distance traveled instead of open areas, which are easier to plan for but could lead to longer solutions that would violate time windows.

To effectively solve multi-goal problems with dynamics and time windows, we develop a multi-layered framework that uses temporal planning to guide sampling-based motion planning.

[1]S. Edelkamp and D. Magazzeni are with the Dept. of Informatics, King's College London, UK {*firstname.lastname*}@kcl.ac.uk

[2]M. Lahijanian is with the Dept. of Computer Science, University of Oxford, Oxford, UK morteza.lahijanian@cs.ox.ac.uk

[3]E. Plaku is with the Dept. of Electrical Engineering and Computer Science, Catholic University of America, Washington, DC, USA plaku@cua.edu

This is a key contribution over DROMOS, which does not incorporate any temporal information. To facilitate temporal reasoning, a discrete abstraction is obtained via a roadmap that captures the connectivity of the free space but ignores dynamics. To account for the dynamics, a motion tree is expanded in the continuous state space by adding collision-free and dynamically-feasible trajectories as branches. A key contribution is the partition of the motion tree into equivalence classes based on the discrete abstraction and temporal information. The partition allows leveraging temporal reasoning to obtain temporal plans that indicate the order and times in which the remaining goals should be visited. This information guides the sampling-based layer which seeks to expand each selected equivalence class along its temporal plan. To promote efficiency, preference is given to equivalence classes associated with short temporal plans. When the expansion along a temporal plan becomes challenging, penalties are applied to promote expansions along other temporal plans. The synergy of these layers makes it possible to intelligently explore the order of goal visits while accounting for the continuous dynamics and time constraints.

To the best of our knowledge, the proposed framework is the first approach for multi-goal motion planning that can effectively take into account both complex dynamics and time windows over goal visits. The framework is agnostic to the inner workings of the temporal planner, so it can be coupled with any temporal planner. Experiments with ground and aerial vehicle models operating in complex environments demonstrate the efficiency and scalability of the approach as we increase the number of goals and tighten the time bounds.

Related Work: AI planning has made great progress in handling tasks and temporal constraints. PDDL2.1 [9] extends PDDL to allow durative actions, continuous resources, and time windows. Several temporal planners are available [1], [2], [10], [11].

For motion planning, sampling-based approaches have been effective in solving challenging problems for high-dimensional systems with nonlinear dynamics by selectively exploring the continuous state space of feasible motions [7]. Such success has been possible by considering simpler tasks, such as moving to a goal region while avoiding collisions. In recent years, there has been a push to increase the capabilities of sampling-based approaches. There are now motion planners that incorporate specifications given by Linear Temporal Logic (LTL) [12], [13]. LTL has also been used for controller synthesis [14]. LTL, however, cannot express time durations, only the order of events. Sampling-based motion planning has also been used with a STRIPS planner for a pick-and-place task [15], but without any temporal constraints. As a result, none of these approaches can be used to solve the multi-goal motion-planning problem with time windows.

Signal Temporal Logic (STL) [16] extends LTL to allow time windows. STL planners [17], [18] follow an optimization formulation to incorporate nonlinear constraints and system dynamics. This is computationally expensive, e.g., mixed integer-linear programs are NP-hard, making STL planners mostly suitable for systems with simple dynamics. For realistic systems that have complex dynamics, the existing STL planning techniques are impractical.

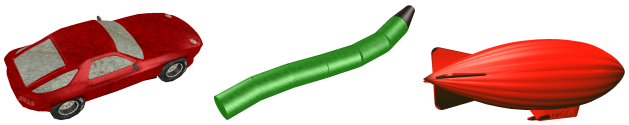


Fig. 2. Vehicle models of a car, snake, and blimp used in the experiments.

II. Problem Formulation

1) World and Robot Models: The world $\mathcal{W}$ has obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$ and goal regions $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$. The robot dynamics are described by differential equations $\dot{s} = f(s, u)$, where $s \in \mathcal{S}$ (state space) and $u \in \mathcal{U}$ (control space). The robot motion is encapsulated by the function

$$s_{\text{new}} \leftarrow \text{SIMULATE}(s, u, f, dt),$$

which numerically integrates f to compute the new state s_{new} obtained by applying u to s for one time step dt.

Fig. 2 shows the car, snake, and blimp models used in the experiments. The motion equations of the car are defined as

$$\dot{x} = v\cos(\theta)\cos(\psi), \dot{y} = v\sin(\theta)\cos(\psi), \quad (1)$$

$$\dot{\theta} = v\sin(\psi)/L, \dot{v} = u_{\text{acc}}, \dot{\psi} = u_{\omega}, \quad (2)$$

where $(x, y, \theta, v, \psi) \in \mathcal{S}$ denotes the position, orientation, velocity, and steering angle; $(u_{\text{acc}}, u_{\omega}) \in \mathcal{U}$ denotes the acceleration and steering rate; and L denotes the axle length.

The vehicle can be made to fly by adding acceleration along the z axis as control input and augmenting the motion equations with $\dot{z} = v_z, \dot{v}_z = u_{\text{acc}_z}$.

As another example, a snake model can be obtained as a car pulling several trailers by setting the hitch distance H between the links to a small value and augmenting f to include the changes that occur to each trailer as

$$\dot{\theta}_i = \frac{v}{H}(\sin(\theta_{i-1}) - \sin(\theta_0))\prod_{j=1}^{i-1}\cos(\theta_{j-1} - \theta_j), \quad (3)$$

where $\theta_0 = \theta$, N is the number of trailers, and θ_i is the orientation of the i-th trailer.

2) Motion Trajectory: A dynamically-feasible trajectory $\zeta : \{1, \ldots, \ell\} \rightarrow \mathcal{S}$ is obtained by starting at a state s and applying a sequence of controls $[u_i]_1^{\ell-1}$ in succession, where $\zeta(1) = s$ and $\forall i \in \{2, \ldots, \ell\}$:

$$\zeta(i) \leftarrow \text{SIMULATE}(\zeta(i-1), u_{i-1}, f, dt). \quad (4)$$

The trajectory ζ is said to have reached region $\mathcal{R}_j \subseteq \mathcal{W}$ at time $dt * i$ if and only if $\zeta(i)$ positions the vehicle inside $\mathcal{R}_i$.

Definition 1: (Multi-Goal Motion Planning with Dynamics and Time Windows) Given

- a world (bounding box) $\mathcal{W}$
- a set of obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$, where $\mathcal{O}_i \subseteq \mathcal{W}$
- a set of goals $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, where
  - $\mathcal{G}_i = \langle \mathcal{R}_i, [t_i^{\text{start}}, t_i^{\text{end}}] \rangle$, $\mathcal{R}_i \subseteq \mathcal{W}$
  - $[t_i^{\text{start}}, t_i^{\text{end}}]$: time interval associated with $\mathcal{G}_i$
- a robot model $\langle \mathcal{S}, \mathcal{U}, f, \text{SIMULATE} \rangle$
- an initial state $s_{\text{init}} \in \mathcal{S}$

compute controls $[u_i]_1^{\ell-1}$ such that the dynamically-feasible trajectory $\zeta : \{1, \ldots, \ell\} \rightarrow \mathcal{S}$ obtained by starting at s and

applying $[u_i]_{i=1}^{\ell-1}$ in succession is collision-free and reaches every goal $\mathcal{G}_i$ within the time interval $[t_i^{\text{start}}, t_i^{\text{end}}]$.

Setting $t_i^{\text{end}} = \infty$ removes the upper bound. The special case of imposing only an upper bound t on the overall task duration can be obtained by setting each $t_i^{\text{end}} = t$.

III. Abstract Problem Solving

As mentioned, our approach uses an abstraction to facilitate temporal reasoning. We first describe our problem in the abstract setting and the solutions that we follow. The next section describes our overall approach that integrates temporal reasoning into sampling-based motion planning.

The problem abstraction used in this paper requires finding a low-cost open tour over a weighted graph that starts at a specified vertex and reaches each vertex within a specified time bound $[t_i^{\text{start}}, t_i^{\text{end}}]$. This problem is referred to as TSP with Time Windows (TSPTW) [19].

Definition 2 (TSPTW): Given

- a start vertex v_{start},
- goal vertices $V_{\text{goals}} = \{g_1, \ldots, g_k\}$ with corresponding time intervals $\{\langle g_1, t_1^{\text{start}}, t_1^{\text{end}}\rangle, \ldots, \langle g_k, t_k^{\text{start}}, t_k^{\text{end}}\rangle\}$, where $0 \leq t_i^{\text{start}} < t_i^{\text{end}} \leq \infty$,
- edges $E \subseteq V \times V$, where $V = \{v_{\text{start}}, g_1, \ldots, g_k\}$, and
- time durations $T = \{t_{(v',v'')} : (v', v'') \in E\}$

compute a path σ over the graph $G = (V, E, T)$ and start times $\langle t_1, \ldots, t_{|\sigma|}\rangle$ such that σ starts at v_{start} and visits each vertex in V_{goals} during the corresponding time interval. Among the valid paths (temporal plans), preference should be given to the ones that reduce the plan duration $t_{|\sigma|}$.

The goal vertices in the abstraction correspond to representative samples from the goal regions. As discussed in Section IV, the temporal planner will be called many times with different start vertices, times, and V_{goals} to reflect the goal regions that have yet to be reached from different branches of the motion tree. Our approach can be used with any temporal planner. We use a PDDL temporal planner and two refined TSPTW solvers (based on depth-first branch-and-bound and Monte-Carlo search).

1) Interface with Temporal PDDL Planner: The temporal planner gets each query as a text file (or a string) consisting of the start vertex, a set of pairs of connected waypoints (*edge wpX wpY*), and the traveltime between each pair of connected waypoints (*traveltime*). *Timed initial literals* are used to specify time windows. The *domain* includes *(move wpX wpY)* durative actions, whose duration is given by the *traveltime* function[1]. A fragment of the query is given below:

```
(at v0) (edge v0 v1) (edge v0 v2) (edge v1 v2)
(= (traveltime v0 v1) 0.8)
(= (traveltime v0 v2) 1.5)
(= (traveltime v1 v2) 0.7)
(located task1 v1) (located task2 v2)
(at 1.1 (tw_open task1)) (at 2.1 (not (tw_open task1)))
(at 2.3 (tw_open task2)) (at 3.3 (not (tw_open task2)))
```

Note that in this new framework, the temporal-planning problem only contains waypoints corresponding to locations of tasks, as opposed to the approach in [5] where the problem includes one waypoint for each node of the PRM [20] used to abstract the continuous space of the motions. This allows increased scalability by an order of magnitude.

The output of the temporal PDDL planner is the list of waypoints (tasks) to be visited in specific times, given how long it takes to move between each pair of waypoints, e.g.,

```
v1 v3 v5 v4 v2
0.0  1.26  3.22  12.55  21.11
```

Note that it might be the case that the vehicle, when moving from v_5 to v_4, takes less than $12.55 - 3.22$ time units. The temporal planner takes this into account and assumes the vehicle will wait at v4 in order to satisfy the time windows (wait actions do not need to be present in the plan).

2) Interface with Specialized TSPTW Solvers: The TSPTW solver provides a function interface using arrays to denote the start and end times and to output the computed tour. We have implemented two specialized solvers based on branch-and-bound and Monte-Carlo search.

a) Branch-and-Bound Search: The depth-first branch-and-bound (DFBnB) procedure [21] uses an upper bound U to prune the search. U can be obtained via a heuristic; the lower it is, the better the pruning, but in case no upper bound is known, it is safe to set U to ∞. The tour is maintained globally and updated during backtracking. Another global variable keeps track of the actual solution path. Temporal constraints are checked when extending the solution and possible waiting times on early arrivals are introduced. Sorting the successors according to increasing cost accelerates the search for finding an early solution.

b) Monte-Carlo Search: Monte-Carlo search has had many successes in games, planning, and optimization [22]. It uses results from rollouts to guide the search; a rollout is a path that descends the search tree making a random move at each level until reaching a leaf. As results can be strongly influenced by the choice of appropriate policy to bias the rollouts, we employ *nested rollout with policy adaptations* (NRPA) [23]. In this context, TSPTW is interpreted as a game to find legal moves by extending a partial tour. NRPA effectively learns valid and short tours [24] by optimizing over an objective function that combines the number of constraint violations with the total travel time. The nested search trades exploration and exploitation and the number of rollouts. The runtime is limited by $O(i^d)$ rollouts, where d is the recursion depth and i is the branching factor.

IV. Overall Method

Our approach has several components: (i) a discrete abstraction obtained via a roadmap that captures the connectivity of the free space; (ii) expansion and partition of a motion tree into equivalence classes based on the discrete abstraction and time information; and (iii) use of temporal planning over the discrete abstraction to guide sampling-based expansion of the equivalence classes. A schematic illustration of the main components and their interplay is shown in Fig. 3. Pseudocode is shown in Alg. 1.

[1]We use the same domain presented in [5].

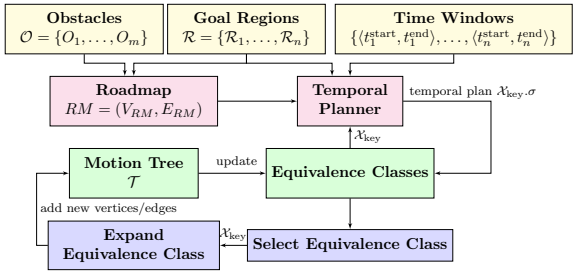


Fig. 3. Components of our approach and their interplay.

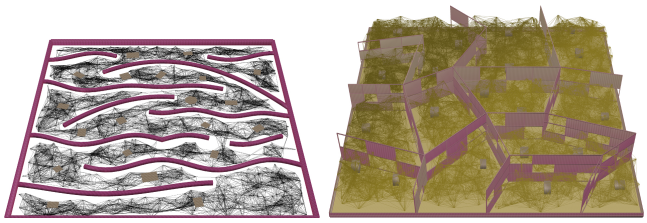


Fig. 4. Examples of roadmaps in 2D and 3D scenes.

A. Discrete Abstraction via a Roadmap

The discrete abstraction provides a simplified problem representation that ignores the dynamics. It is obtained by constructing a roadmap $RM = (V_{RM}, E_{RM})$ over the configuration space, where a configuration defines only the position and orientation. The objective is to construct a dense roadmap that connects the goals and provides many routes to reach the goals. First, a configuration $q_{\mathcal{G}_i}$ is added to RM for each goal $\mathcal{G}_i$ by sampling a point inside $\mathcal{G}_i$ and a random orientation. Leveraging PRM [20], the roadmap is further populated by generating n_{add} collision-free configurations and then attempting to connect each configuration to its n_{neighs} nearest neighbors. The path between two neighboring configurations q, q_{neigh} is defined via interpolation. The edge (q, q_{neigh}) is added to the roadmap when the path from q to q_{neigh} is not in collision. This process of adding and connecting configurations is repeated until all the goals belong to the same connected component in RM. In the experiments, we used $n_{\text{add}} = 1500$ and $n_{\text{neighs}} = \lceil \log_2 |V_{RM}| \rceil$. Fig. 4 shows some examples of the roadmaps that were created.

To facilitate temporal reasoning, each edge stores the expected time it would take the robot to travel along the associated path. Such time is estimated based on the path distance and expected velocity of the robot.

B. Motion-Tree Partition based on Discrete Abstraction and Temporal Plans

Starting at s_{init}, a motion tree $\mathcal{T}$ is expanded in the continuous state space $\mathcal{S}$ by adding new vertices. Each vertex $v \in \mathcal{T}$ has the fields $\{s, u, \text{parent}, t, \text{goals}, q\}$, which correspond to a state, control, parent, time duration, remaining goals, and nearest roadmap configuration, respectively. By construction, $v.s$ is a collision-free state. $\mathcal{T}$ is expanded from v by applying a control u to $v.s$ and simulating the robot motion for one time step dt. If not in collision, the new vertex v_{new}, where $v_{\text{new}}.s \leftarrow \textsc{simulate}(v.s, u, f, dt)$, is added to $\mathcal{T}$ with v as its parent (Alg. 1b).

Algorithm 1 Proposed approach to integrate temporal reasoning into sampling-based motion planning.

Input: world $\mathcal{W}$; obstacles $\mathcal{O}$; goals $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, $\mathcal{G}_i = \langle \mathcal{R}_i, [t_i^{\text{start}}, t_i^{\text{end}}]\rangle$; robot model $\langle \mathcal{S}, \mathcal{U}, f\rangle$; initial state s_{init}; time step dt; runtime limit $t_{\max}$
Output: collision-free and dynamically-feasible trajectory that reaches each goal within its time window; or `null` if no solution is found

$RM \leftarrow \textsc{ConstructRoadmap}(\mathcal{O}, \mathcal{G})$ *//§IV-A*
$\Xi \leftarrow \textsc{ShortestPaths}(RM, \mathcal{G})$
$\mathcal{T} \leftarrow \textsc{InitializeMotionTree}(s_{\text{init}})$
$\mathcal{X} \leftarrow \emptyset$; $\textsc{UpdateEquivalenceClasses}(\mathcal{X}, \textsc{RootVertex}(\mathcal{T}))$
while $\textsc{time}() < t_{\max}$ **do**
  $\mathcal{X}_{\text{key}} \leftarrow \textsc{SelectEquivalenceClass}(\mathcal{X})$ *//§IV-C1*
    ◇ *Expand* $\mathcal{X}_{\text{key}}$ *along temporal plan* $\mathcal{X}_{\text{key}}.\sigma$ *§IV-C2*
  $p \leftarrow \textsc{SelectTarget}(\mathcal{X}_{\text{key}}.\sigma)$
  $v \leftarrow \textsc{SelectVertex}(\mathcal{X}_{\text{key}}.\text{vertices}, p)$
  for several steps **do**
    $\langle v_{\text{new}}, \text{wait}\rangle \leftarrow \textsc{StepTowardTarget}(\mathcal{T}, \mathcal{X}, v, p, \texttt{false})$
    if $v_{\text{new}} = \texttt{null}$ **then break**
    if $v_{\text{new}}.\text{goals} = \emptyset$ **then return** $\zeta_{\mathcal{T}}(v_{\text{new}})$
    if $\neg\textsc{UpdateEquivalenceClasses}(\mathcal{X}, v_{\text{new}})$ **then**
      $\{\textsc{RemoveVertex}(\mathcal{T}, v_{\text{new}})$; **break**$\}$
    if $\neg$wait **and** $\textsc{NearTarget}(s_{\text{new}}, p)$ **then break**
    $v \leftarrow v_{\text{new}}$
return `null`

(a) $\textsc{UpdateEquivalenceClasses}(\mathcal{X}, v_{\text{new}})$ *//§IV-B*
$\mathcal{X}_{\text{new}} \leftarrow \textsc{FindEquivalenceClass}(\mathcal{X}, \langle v_{\text{new}}.q, v_{\text{new}}.\text{goals}, v_{\text{new}}.t\rangle)$
if $\mathcal{X}_{\text{new}} = \texttt{null}$ **then**
  $\mathcal{X}_{\text{new}} \leftarrow \textsc{NewEquivalenceClass}(\langle v_{\text{new}}.q, v_{\text{new}}.\text{goals}, v_{\text{new}}.t\rangle)$
  $\mathcal{A} \leftarrow \textsc{TemporalPlannerGraph}(RM, \Xi, \langle v_{\text{new}}.q, v_{\text{new}}.\text{goals}\rangle)$
  $\mathcal{X}_{\text{new}}.\sigma \leftarrow \textsc{TemporalPlanner}(\mathcal{A}, \langle v_{\text{new}}.q, v_{\text{new}}.\text{goals}, v_{\text{new}}.t\rangle)$
  if $\mathcal{X}_{\text{new}}.\sigma = \texttt{null}$ **then return** `false`
  $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathcal{X}_{\text{new}}\}$
$\mathcal{X}_{\text{new}}.\text{vertices} \leftarrow \mathcal{X}_{\text{new}}.\text{vertices} \cup \{v_{\text{new}}\}$
return `true`

(b) $\textsc{StepTowardTarget}(\mathcal{T}, \mathcal{X}, v, p, \text{wait})$ *//§IV-C2*
$u \leftarrow \textsc{controller}(v.s, p, \text{wait})$
$s_{\text{new}} \leftarrow \textsc{simulate}(v.s, u, f, dt)$
if $\textsc{collision}(s_{\text{new}})$ **then return** `null`
$v_{\text{new}}.[\text{parent}, s, t, \text{goals}, q] \leftarrow$
  $\textsc{AddNewVertex}(\mathcal{T}, v, s_{\text{new}}, v.t+dt, v.\text{goals}, \textsc{map}(RM, s_{\text{new}}))$
if $(\mathcal{G}_i \leftarrow \textsc{ReachedGoal}(\mathcal{G}, v_{\text{new}})) \neq \texttt{null}$ **then**
  wait $\leftarrow v_{\text{new}}.t < t_i^{\text{start}}$
  if $v_{\text{new}}.t \in [t_i^{\text{start}}, t_i^{\text{end}}]$ **then** $v_{\text{new}}.\text{goals} \leftarrow v.\text{goals} - \{\mathcal{G}_i\}$
return $\langle v_{\text{new}}, \text{wait}\rangle$

Let $\zeta_{\mathcal{T}}(v)$ denote the trajectory obtained as the sequence of states connecting the root of $\mathcal{T}$ to v. To facilitate temporal reasoning, v keeps track of $v.t$ and $v.\text{goals}$, which denote the time duration of $\zeta_{\mathcal{T}}(v)$ and the set of goals that have yet to be reached by $\zeta_{\mathcal{T}}(v)$. The vertex v reaches goal $\mathcal{G}_i = \langle \mathcal{R}_i, [t_i^{\text{start}}, t_i^{\text{end}}]\rangle$ if $v.s$ places the robot inside the region $\mathcal{R}_i$ and $v.t \in [t_i^{\text{start}}, t_i^{\text{end}}]$. When v reaches $\mathcal{G}_i$, $\mathcal{G}_i$ is removed from $v.\text{goals}$. Thus, $\zeta_{\mathcal{T}}(v)$ is a solution to the multi-goal planning problem with time windows if $v.\text{goals} = \emptyset$.

We leverage the discrete abstraction and temporal planning to effectively guide the expansion and partition $\mathcal{T}$ into equivalence classes. The premise is that vertices that provide the same discrete information should belong to the same equivalence class. The partition seeks to determine how to expand $\mathcal{T}$ from v. It requires searching the discrete abstraction to find a temporal plan with v as the start vertex, $v.t$ as the start time, and $v.\text{goals}$ as the set of unreached goals (with their time windows). For this reason, v is mapped to the closest configuration in the roadmap RM, denoted by $v.q$. We can now use the solvers described in Section III to find a temporal

plan over RM, denoted by $v.\sigma$.

Consider a new vertex $v_{\text{new}} \in \mathcal{T}$. We could possibly use the above procedure to find a temporal plan for v_{new}. However, $\mathcal{T}$ typically has tens of thousands of vertices and invoking a temporal planner from each vertex would be infeasible. To address this, we introduce the notion of equivalence classes. The vertex v_{new} belongs to the same equivalence class as v when they map to the same roadmap configuration, have the same remaining goals, and the temporal plan associated with v is compatible with the start time $v_{\text{new}}.t$. In other words, $v.\sigma$ does not violate any time constraints when the start time is changed from $v.t$ to $v_{\text{new}}.t$. We can now define the equivalence class $\mathcal{X}_v$ as

$$\mathcal{X}_v = \{v_{\text{new}} : v_{\text{new}} \in \mathcal{T} \wedge v.q = v_{\text{new}}.q \wedge \quad (5)$$

$$v.\text{goals} = v_{\text{new}}.\text{goals} \wedge \quad (6)$$

$$\text{CompatibleTemporalPlan}(\mathcal{G}, v.\sigma, v_{\text{new}}.t) = \texttt{true}\} \quad (7)$$

The first equivalence class is $\mathcal{X}_{v_{\text{init}}}$. When v_{new} is added to $\mathcal{T}$, a check is done to determine if v_{new} can be added to an existing equivalence class. If not, a new equivalence class $\mathcal{X}_{v_{\text{new}}}$ is created. Pseudocode is shown in Alg. 1a.

When creating $\mathcal{X}_{v_{\text{new}}}$, the temporal planner is invoked to compute $\mathcal{X}_{v_{\text{new}}}.\sigma$ using $v_{\text{new}}.q$ as the start vertex, $v_{\text{new}}.t$ as the start time, and $\langle q_1, \ldots, q_k \rangle = \langle q_{\mathcal{G}_i} : \mathcal{G}_i \in v_{\text{new}}.\text{goals} \rangle$ as the goal vertices. The time duration for any pair (q', q'') where $q', q'' \in \{v_{\text{new}}.q, q_1, \ldots, q_k\}$ is set to the time duration of the shortest path in RM from q' to q''. This is a much smaller graph than RM, and will get even smaller as $\mathcal{T}$ reaches new goals. To generate the query efficiently, the framework precomputes shortest-path distances using Dijkstra's shortest-path algorithm. This takes only a negligible fraction of the runtime in the problem settings considered here (several thousand nodes and edges and up to 20 goals).

C. Integrated Search

After constructing the roadmap RM, initializing the motion tree $\mathcal{T}$, and creating the first equivalence class $\mathcal{X}_{v_{\text{init}}}$, the overall approach selects an equivalence class $\mathcal{X}.v$ and expanding $\mathcal{T}$ from $\mathcal{X}.v$ along the temporal plan $\mathcal{X}.\sigma$. As $\mathcal{T}$ is expanded, new equivalence classes are created. These procedures are called repeatedly until a solution is found or the runtime limit is reached (Alg. 1:5–17).

1) Selecting an Equivalence Class based on Time Duration of the Temporal Plan: To promote effectiveness, priority is given to equivalence classes associated with short temporal plans. We take into account both the time duration when the temporal plan is supposed to be completed as well as the number of goals in the temporal plan when defining a weight for each equivalence class:

$$\text{weight}(\mathcal{X}.v) = \frac{\alpha^{\text{NrSelections}(\mathcal{X}.v)}}{\text{duration}(\mathcal{X}.\sigma) * 2^{|\mathcal{X}.\sigma|}}. \quad (8)$$

The equivalence class with maximum weight is then selected for expansion. Note the aggressiveness of the number of goals in $\mathcal{X}.\sigma$ as we would like to quickly generate plans that reach all the goals. We also introduce a penalty factor, $\alpha^{\text{NrSelections}(\mathcal{X}.v)}$ $(0 < \alpha < 1)$, based on the number of times $\mathcal{X}.v$ has been previously selected for expansion. Without it, the approach could become stuck trying to indefinitely expand from the same equivalence class, even though constraints imposed by dynamics and obstacles may make it impossible or difficult to do so. These multi-objective criteria promote rapid expansions along short temporal plans while allowing the approach to explore alternative plans.

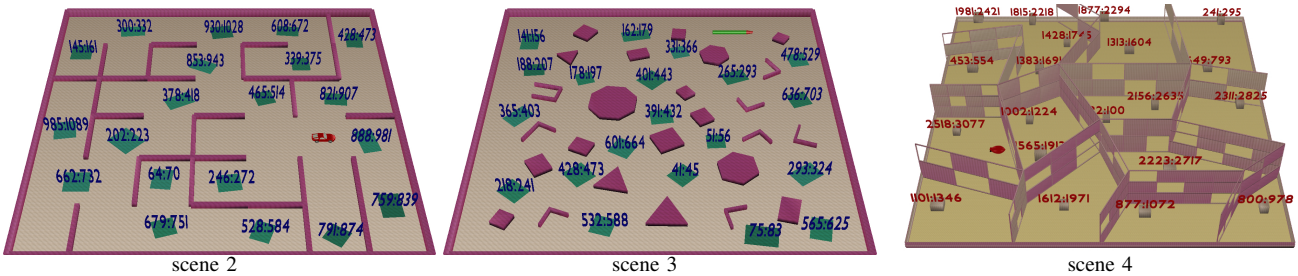
2) Expanding the Equivalence Class along the Temporal Plan: After selecting an equivalence class $\mathcal{X}.v$, the objective becomes to expand $\mathcal{T}$ so that it reaches the goals in the temporal plan $\mathcal{X}.\sigma$ in succession and within the specified time bounds (Alg. 1b). Let $\mathcal{G}_1$ be the first goal in $\mathcal{X}.\sigma$. To reach it, the approach attempts to expand $\mathcal{T}$ along the shortest path in the roadmap from $v.q$ to $q_{\mathcal{G}_i}$. Using the shortest path allows the approach to reduce the time traveled, thus improving the likelihood of reaching $\mathcal{G}_i$ in time. A proportional-derivative-integrative (PID) controller is used to expand $\mathcal{T}$ toward the roadmap configurations in succession. For a vehicle, the PID controller would turn the wheels toward the target and then move straight to it. The PID controller is run for several steps. Each intermediate state is added to $\mathcal{T}$. When the branch expansion reaches a goal $\mathcal{G}_i$ earlier, it waits there. The branch expansion stops when a collision is encountered. It also stops if the new vertex v_{new} is not compatible with an existing equivalence class and the temporal planner is unable to compute a temporal plan for $\mathcal{X}.v_{\text{new}}$. When the motion-tree expansion stops, the procedure updates the equivalence classes and their weights, and goes back to the core loop. In the next iteration, it could possibly select a different equivalence class since the weights might have changed. When the approach has difficulty expanding $\mathcal{T}$ along the shortest path from $v.q$ to $q_{\mathcal{G}_i}$, then attempts are made to expand $\mathcal{T}$ in some random direction. In this way, the approach explores the sorrounding areas, which could also lead to the creation of new equivalence classes. This interplay between temporal planning and sampling-based motion planning is the salient feature of our approach that allows it to effectively solve challenging problems, as the experiments demonstrate.

V. Experiments and Results

Experiments use complex scenes (Figs. 1 and 5) and robot models (snake, car, blimp) with nonlinear dynamics (Section II). Scalability and efficiency are evaluated by increasing the number of goals and tightening the time bounds.

1) Temporal Planners: We use *Random*, *Optimal*, *Branch-and-bound*, and *Monte-Carlo* as specialized TSPTW solvers and *Optic* [2] as the more generic temporal planner. *Random* iterates over all permutations until it finds a compatible temporal plan. *Optimal* does not stop early but updates the temporal plan when a better one is found. Since *Random* and *Optimal* iterate over $n!$ permutations, they are mainly used for small problem instances with up to $n = 8$ goals.

2) Benchmark Instances: We generate 30 instances for each combination of scene and number of goals. Each instance is generated by randomly placing the goals, ensuring that they do not collide with the obstacles. To make the problems harder, goals are not cluttered so that the vehicle has to travel for longer distances.



scene 2 scene 3 scene 4

Fig. 5. Scenes used in the experiments (see also Fig. 1).

Each goal is associated with a time bound. Generating the time bounds at random leads to many unsolvable instances. For this reason, we generated the time bounds by first generating a random location and a tour at random. We then computed the time t_i each goal $\mathcal{G}_i$ in the tour would be reached when traveling along the shortest paths in the roadmap graph, using the expected velocity to convert distances to times. A time window is then defined as $[(1-\epsilon)t_i, (1+\epsilon)t_i]$ for some parameter ϵ. In the experiments, the default value is $\epsilon = 0.2$. In Section V-5, its value is varied from $\inf$ to 1.0, 0.4, 0.2, 0.1, and 0.05.

For each combination of parameters, the planner is run over all instances. Results report mean runtime and plan duration after dropping runs that are below the first quartile or above the third quartile to avoid the influence of outliers. Runtime measures everything from reading the input until reporting a solution or reaching the runtime limit (set to 20s).

3) Runtime Results when Increasing the Number of Goals: Fig. 6 summarizes the results when varying the number of goals from 4 to 20. Results show the efficiency of the approach as it is able to solve these challenging problems within a few seconds. When coupled with *Random* and *Optimal*, the approach can only solve small problem instances with up to 8 goals. Using the specialized TSPTW planners makes it possible to solve larger problem instances with up to 20 goals. Because of the time bounds, the motion tree is partitioned not just in space and with respect to goals that have been reached but also with respect to time and temporal plans. As a result, more equivalence classes are created. Even though this increases the number of calls to the discrete planner, it allows the approach to selectively explore the state space along routes defined by temporal plans. As a result, in a few seconds we are able to solve challenging problems with up to 20 goals and tight time bounds, demonstrating the importance of integrating temporal reasoning into sampling-based motion planning.

4) Baseline Comparisons: As discussed in Section I, there are no other methods specifically designed to solve multi-goal motion planning with dynamics and time windows. For baseline comparisons, we used DROMOS [8], which was designed for multi-goal motion planning with dynamics but does not take time windows into account. To make it work here, we modified DROMOS to mark a goal $\mathcal{G}_i$ as reached only when a motion-tree branch reaches $\mathcal{G}_i$ within the time window associated with $\mathcal{G}_i$. To show the importance of coupling temporal planning with motion planning, we also created a decoupled version of our approach, which computes only one temporal plan σ in the beginning and never changes it. The decoupled planner then seeks to expand the motion tree from one goal to the next as defined by σ. We also used an RRT to reach the goals in succession as defined by σ.

Fig. 7 shows the results when comparing our approach to the modified DROMOS and sequential planners. As expected, these other planners have difficulty solving challenging problems, timing out in cases with 8 or more goals. As DROMOS does not use temporal information, it becomes increasingly difficult to reach each goal within its time window. For the decoupled planner and RRT, the issue is that it does not change the temporal plan. As a result, the motion-tree expansion becomes stuck when constraints imposed by the dynamics and obstacles make it difficult to follow the temporal plan. In contrast, the interplay between motion planning and temporal planning in our approach makes it possible to expand the motion tree along alternative temporal plans. In this way, our approach continually makes progress toward reaching each goal within its time window.

5) Runtime Results when Adjusting the Time Windows: The approach is also evaluated when adjusting the time windows by varying ϵ from $\inf$ to 1.0, 0.4, 0.2, 0.1, and 0.05, so the time window would be $[(1-\epsilon)t_i, (1+\epsilon)t_i]$. Results in Fig. 8 show that the approach is most effective when there are no bounds $\epsilon = \inf$ or when the bounds are loose. As ϵ is made smaller, it becomes increasingly challenging to find solution trajectories that satisfy the time constraints. Nevertheless, our approach is able to effectively find solutions even as the bounds are made tighter. This again is due to the use of temporal planning to effectively guide sampling-based motion planning.

6) Runtime Distribution: Fig. 9 shows the runtime distribution for various components of our approach. For smaller problem instances, the roadmap construction takes more time since the motion-tree expansion quickly finds solutions. As the number of goals increases, it becomes harder to find solutions so more time is spent in the motion tree expansion and the interplay with the temporal planner. As noted earlier, the number of equivalence classes could reach into the hundreds requiring that many calls to the temporal planner. One caveat is that the the number of goals on which the temporal planner is run becomes smaller and smaller as the tree expands and reaches the goals. Nevertheless, there is substantial work done by the temporal planner. The runtime distribution does not imply that temporal planning is the bottleneck. In fact, our approach is shifting the load from the motion-tree expansion, which is slow, to the temporal planner which can find increas-

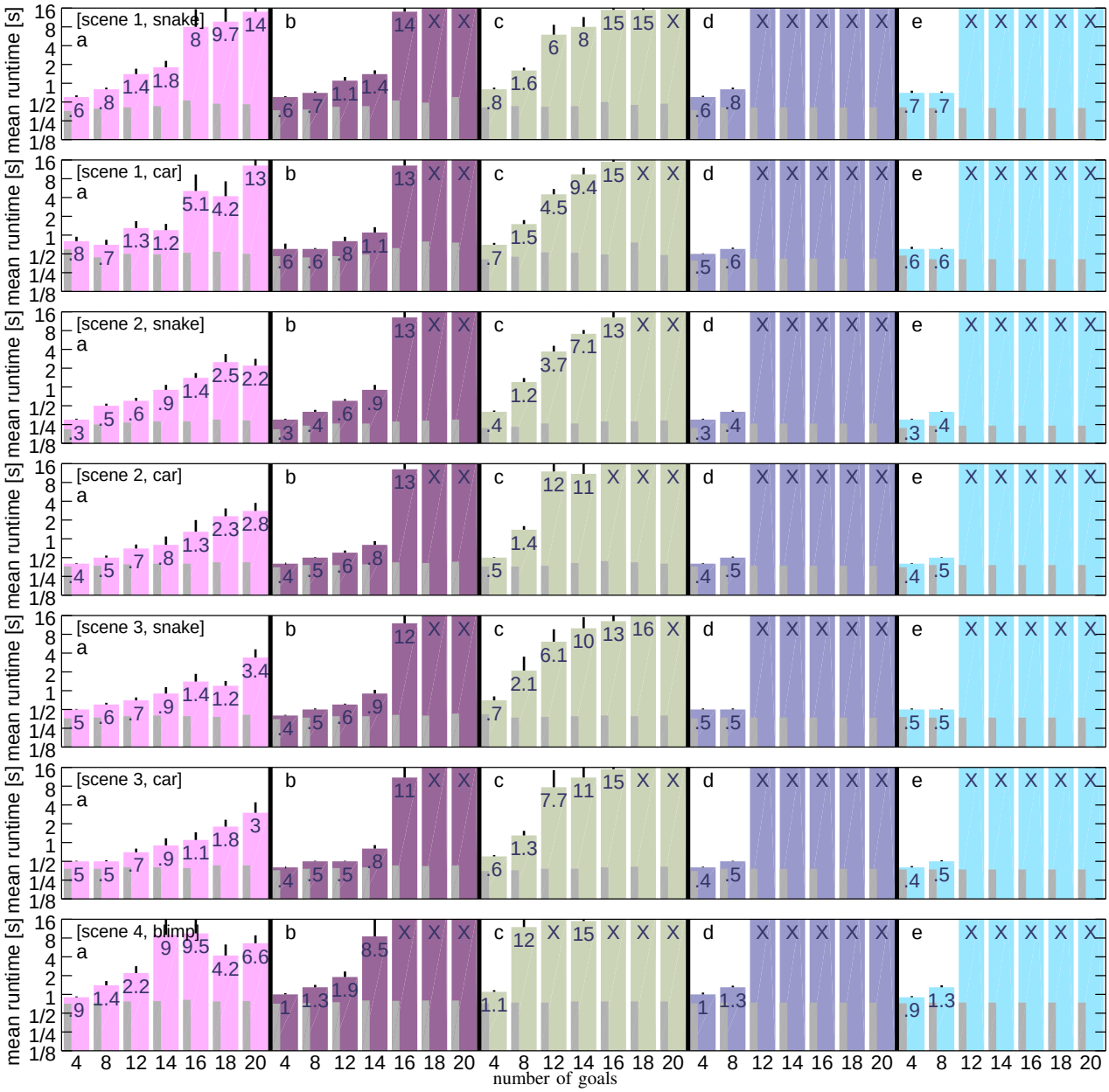


Fig. 6. Runtime results when varying the discrete planner in our approach as a function of the number of goals: (a) MC (b) BNB (c) Optic (d) Random (e) Optimal. Runtime includes everything from preprocessing to reporting that a solution is found. The thin bar on the left side indicates the preprocessing time (constructing the roadmap and running Dijkstras single-source shortest path algorithm from each of the goals).

ingly effective plans over the discrete abstraction to guide the motion-tree expansion.

7) Results on the Time Duration of Solution Trajectories: Fig. 10 shows that our approach generally finds good solution trajectories since it is guided by temporal plans that seek to reduce the time duration to reach all the goals.

VI. Discussion

Multi-goal motion planning with dynamics and time windows is relevant in many logistics applications. This work developed an approach that integrated temporal reasoning into sampling-based motion planning. Scalability and efficiency were shown by increasing the number of goals and tightening the time windows. The approach is also agnostic to the inner workings of the temporal planner so that it can be used in conjunction with any temporal planner. One direction for future research is to consider oversubscription planning when, due to temporal constraints, it is not possible to reach all the goals. The objective then is to maximize the number of reached goals. As we looked at time windows attached to goals, future research towards tighter integration of task and motion planning will exploit the much larger expressiveness of PDDL temporal planners in more complex scenarios.

References

[1] P. Eyerich, R. Mattmüller, and G. Röger, "Using the context-enhanced additive heuristic for temporal and numeric planning," *Towards Service Robots for Everyday Environments*, vol. 76, pp. 49–64, 2012.

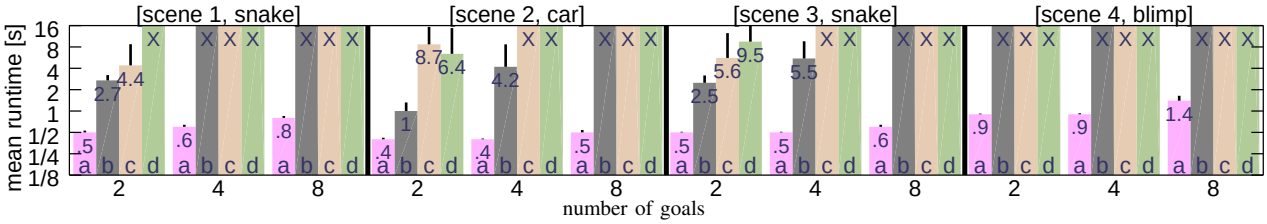


Fig. 7. Comparison of (a) our approach to baseline planners: (b) modified DROMOS, (c) decoupled version of our approach, and (d) SuccessionRRT, as described in Section V-4. For these results, our approach, its decoupled version, and SuccessionRRT used MC as the temporal planner. Modified DROMOS does not use any temporal planner. The p-values of the statistical tests when comparing our approach to the baseline planners were below 0.001.

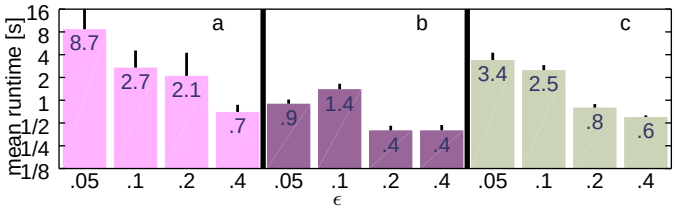


Fig. 8. Runtime results when adjusting the time window of each goal by varying ϵ in $[(1-\epsilon)t_i, (1+\epsilon)t_i]$. (left) Our approach with the *Monte-Carlo* temporal planner over scene 4 using the blimp model and 16 goals. (middle) Our approach with the *Branch-and-bound* temporal planner over scene 1 using the snake model and 14 goals. (right) Our approach with *Optic* temporal planner over scene 2 using the snake model and 12 goals.

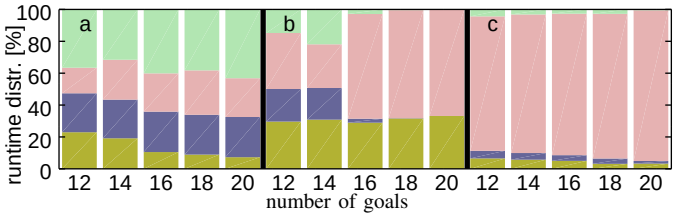


Fig. 9. Runtime distribution as a percentage of the total runtime for the various components of our approach (from bottom to top): (1) roadmap construction and precomputation of shortest paths from each goal, (2) collision checking and SIMULATE, (3) temporal planner, and (4) other. Results are shown for our approach used with (a) *Monte-Carlo*, (b) *Branch-and-bound*, and (c) *Optic* temporal planners in the case of [scene 1, snake].

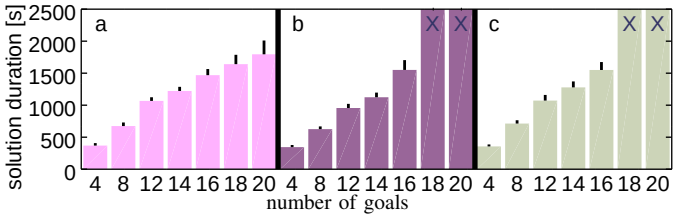


Fig. 10. Time duration of the solution, i.e., time to move along the solution trajectory. Results correspond to our approach when used with (a) *Monte-Carlo*, (b) *Branch-and-bound*, and (c) *Optic* for scene 1 and the car model.

[2] J. Benton, A. J. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *International Conference on Automated Planning and Scheduling*, vol. 77, 2012, pp. 2–10.

[3] M. Fox, D. Long, and D. Magazzeni, "Plan-based policies for efficient multiple battery load management," *Journal of Artificial Intelligence Research*, pp. 335–382, 2012.

[4] C. Piacentini, D. Magazzeni, D. Long, M. Fox, and C. Dent, "Solving realistic unit commitment problems using temporal planning: Challenges and solutions," in *International Conference on Automated Planning and Scheduling*, 2016, pp. 421–430.

[5] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "AUV mission control via temporal planning," in *IEEE International Conference on Robotics and Automation*, 2014, pp. 6535–6541.

[6] H. Keller, *Numerical Methods for Two-Point Boundary-Value Problems*. New York, NY: Dover, 1992.

[7] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT, 2005.

[8] E. Plaku, S. Rashidian, and S. Edelkamp, "Multi-group motion planning in virtual environments," *Computer Animation and Virtual Worlds*, 2016, in press, doi: 10.1002/cav.1688.

[9] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, pp. 61–124, 2003.

[10] E. Scala, P. Haslum, S. Thiébaux, and M. Ramírez, "Interval-based relaxation for general numeric planning," in *European Conference on Artificial Intelligence*, 2016, pp. 655–663.

[11] M. Cashmore, M. Fox, D. Long, and D. Magazzeni, "A compilation of the full PDDL+ language into SMT," in *International Conference on Automated Planning and Scheduling*, 2016, pp. 79–87.

[12] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal planning in uncertain environments with partial satisfaction guarantees," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 538–599, May 2016.

[13] J. McMahon and E. Plaku, "Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3726–3733.

[14] J. A. DeCastro and H. Kress-Gazit, "Guaranteeing reactive high-level behaviors for robots with complex dynamics," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2013, pp. 749–756.

[15] E. Plaku and D. Le, "Interactive search for action and motion planning with dynamics," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 28, pp. 849–869, 2016.

[16] O. Maler and D. Nickovic, "Monitoring Temporal Properties of Continuous Signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems: Joint International Conferences on Formal Modeling and Analysis of Timed Systmes*. Springer, 2004, pp. 152–166.

[17] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *IEEE Conference on Decision and Control*, 2014, pp. 81–87.

[18] L. Lindemann and D. V. Dimarogonas, "Robust motion planning employing signal temporal logic," in *American Control Conference*, 2017, pp. 2950–2955.

[19] T. S. Kumar, M. Cirillo, and S. Koenig, "On the traveling salesman problem with simple temporal constraints," in *Symposium on Abstraction, Reformulation, and Approximation*, 2013, pp. 73–79.

[20] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[21] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[22] M. H. Winands, Y. Björnsson, and J.-T. Saito, "Monte-carlo tree search solver," *Computers and Games*, vol. 5131, pp. 25–36, 2008.

[23] C. D. Rosin, "Nested rollout policy adaptation for monte carlo tree search," in *International Joint Conference on Artificial Intelligence*, 2011, pp. 649–654.

[24] S. Edelkamp, M. Gath, T. Cazenave, and F. Teytaud, "Algorithm and knowledge engineering for the TSPTW problem," in *IEEE Symposium on Computational Intelligence in Scheduling*, 2013, pp. 44–51.