2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Optimizing pre-scheduled, intermittently-observed MDPs

Patrick Zhong[1], Federico Rossi[2], and Dylan A. Shell[1]

Abstract— **A challenging category of robotics problems arises when sensing incurs substantial costs. This paper examines settings in which a robot wishes to limit its observations of state, for instance, motivated by specific considerations of energy management, stealth, or implicit coordination. We formulate the problem of planning under uncertainty when the robot's observations are intermittent but their timing is known via a pre-declared schedule. After having established the appropriate notion of an optimal policy for such settings, we tackle the problem of joint optimization of the cumulative execution cost and the number of state observations, both in expectation under discounts. To approach this multi-objective optimization problem, we introduce an algorithm that can identify the Pareto front for a class of schedules that are advantageous in the discounted setting. The algorithm proceeds in an accumulative fashion, prepending additions to a working set of schedules and then computing incremental changes to the value functions. Because full exhaustive construction becomes computationally prohibitive for moderate-sized problems, we propose a filtering approach to prune the working set. Empirical results demonstrate that this filtering is effective at reducing computation while incurring only negligible reduction in quality. In summarizing our findings, we provide a characterization of the run-time *vs* quality trade-off involved.**

I. Introduction

We examine planning and control problems where obtaining a reliable estimate of state can be costly or is generally undesirable. Unlike the existing body of work on information gathering and active perception [3], [1], [2], the core question is not *what* or *how* to sense, but rather *when* to do so. We consider a setting in which the timing of observations must be pre-planned: the robot is not merely permitted to decide online that it would be convenient to receive a sensor reading now, but must pre-schedule the observations. Far from being esoteric or abstruse, such problem instances arise naturally:

1) Sensor network–enabled navigation: Imagine a robot is navigating through a cave or subterranean cavern. Suppose that a mesh network of sensor motes is deployed within the space and the robot queries the network to obtain its pose by triangulating signal strengths (e.g., see [4]). To boost longevity, the network nodes conserve energy by entering cycles of waking and hibernation. The robot can triangulate its position only when the nodes aren't hibernating. Fortunately, one can specify, prior to deployment, the schedule by which nodes set the watchdog timer to trigger their (synchronized) waking alarm. Some schedules will be more informative than others; naturally, one wishes to understand the relationship between the energy cost of the network and navigational cost of the mobile robot.

2) Stealth amidst snooping adversaries: Consider an autonomous off-road vehicle moving in some GPS-denied environment. The vehicle is supported by aircraft that fly overhead at intervals, using their bird's-eye view to provide state information. In the presence of nefarious entities wishing to harass either the vehicle or the aircraft, it is important to find the right compromise between risk posed by more frequent flyovers *vs* the precision of off-road navigation.

3) Relative rendezvous: A pair of underwater gliders attempt to collect data in two parallel transects. They resurface occasionally to determine their poses relative to one another and to communicate, before diving again to make additional measurements. Treated as a single system with joint actions, what is an effective pre-determined ascent/descent schedule, exploiting knowledge of the environment and their task?

A. Contributions and Paper Outline

In the preceding, the robots all operate under uncertainty; they receive observations which are intermittent, but their sporadic occurrences have been scheduled beforehand. The three examples have the same core issue: task performance generally improves with additional information but, though obtaining a state estimate with high frequency diminishes uncertainty, the expense incurred (expressed as fuel/energy costs, or diminished stealth, or other factors) may outweigh those gains. The tension between these elements means that the question is how to find a suitable balance.

The perspective adopted in this paper is that appropriate compromises can be struck if informed by the Pareto frontier in the space of execution and observation costs. Obtaining an exact representation this frontier is challenging and, hence, we explore how to obtain an approximation in reasonable time. In Sections III and IV, the paper formulates the class of problem, making both notions of cost precise, as well as that of schedules. In Section V, we provide an algorithm that, starting from a collection of basic schedules, expands the set via a prepending operation along with an incremental update to Bellman-like value functions. The method includes several parameters which simplify its operation, including a filtering approach that trades resolution and completeness for running-time. These aspects are explored through case studies and experiments in Sections VI and VII of the paper.

II. Related work

Broadly speaking, choosing an observation schedule involves optimizing the perception process; this paper, thus, represents an sort of perceptual optimization, realized via

[1]Dept. of Comp. Sci. & Eng., Texas A&M University, College Station, TX 77843, USA. {patrickzhong | dshell}@tamu.edu

[2]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA. federico.rossi@jpl.nasa.gov

We acknowledge the support of Office of Naval Research Award #N00014-22-1-2476. Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

pre-planning. We explore how choosing when to sense is important, and are constrained to have an "open-loop declaration" of this fact. This constraint makes our work rather distinct from standard problems where most solutions use gathered information (or estimates) to alter perception online.

The present paper generalizes the work presented in [8]. In that work, the authors assume that state observations are sparse, but will appear with a strict periodicity: every κ timesteps, for a specific and given κ. The algorithm in the present paper searches the space of schedules, a space containing many options besides the strictly periodic ones. And, as will be seen, the algorithm's output can directly identify situations in which non-periodic schedules are superior.

The example scenarios earlier are all described as involving forms of multi-agent interaction. For instance: the aircraft executing the overhead flight generates an observation and communicates it with the ground vehicle. The schedule forms a sort of communication timetable. Existing work, albeit with a quite different focus, examines the question of what and when to communicate in planning settings, e.g., [9], [10].

III. Problem Formulation

To begin, we define the class of pre-scheduled MDPs and their solutions. We then give suitable costs associated with schedules, formalize the core problem, and subsequently turn to some certain types of regularity on schedules.

A. Preliminaries

Definition 1 (Schedules)**.** *A* schedule *is a function* $\mathcal{D} : \mathbb{N} \cup \{0\} \to \mathbb{N}$ *describing a sempiternal sequence. The schedule has* stride bounded by σ *if* $\mathcal{D}(n) \leq \sigma, \forall n \in \mathbb{N}$. *Schedule* $\mathcal{D}$ *is* tail recurrent *if* $\exists N_0 \in \mathbb{N}$, $\mathcal{D}(n+1) = \mathcal{D}(n), \forall n \geq N_0$.

We will write schedules simply as sequences of numbers, for example, consider $\mathcal{D} = (1, 3, 2, 1, 3, 2, 2, 2, 2, 2, \ldots) \equiv (1, 3, 2, 1, 3, \overline{2})$. To be compatible with the function definition, we will refer to elements with indexes starting from zero. Clearly, the previous example given is tail recurrent. To ease subsequent presentation, we use illustratory schedules with stride bounded by 9, writing them concisely: $13213\overline{2}$.

Definition 2 (PS-MDP)**.** *A* pre-scheduled Markov Decision Process *is a tuple* $\langle S, A, T, C_{\text{ex}}, G, \mathcal{D} \rangle$ *where*

- $S = \{s_1, s_2, \ldots, s_{|S|}\}$ *is the finite set of states;*
- $A = \{a_1, a_2, \ldots, a_{|A|}\}$ *is the finite set of actions;*
- $T : S \times A \times S \to [0, 1]$ *give the transition dynamics describing the stochastic state transitions of the system, assumed to be Markovian in the states, where:*
$$P(s^{t+1} = s' \mid s^t = s, a^t = a) = T(s', a, s), \ \forall t,$$
- $C_{\text{ex}} : S \times A \to \mathbb{R}$ *is the function prescribing the cost incurred for taking action* a *in state* s*;*
- $G \subseteq S$ *is a goal region;*
- $\mathcal{D}$ *is a schedule of state observations (or check-ins).*

We will require that when $s \in G$*, for every* $a \in A$*, both* $C_{\text{ex}}(s, a) = 0$ *and* $T(s, a, s) = 1$.

The first four elements are standard in the MDP literature [5], [7], [6]. In classic instances, the optimal control problem involves, at each point in time: (1) obtaining, via sensors, the robot's state, and (2) the decision of which action to execute, and then (3) the robot undergoing a transition satisfying the transition model. The PS-MDP differs from the MDP in the sparsity of observations: states are disclosed to the robot, but only at specific points in time. These times are those described by the schedule, $\mathcal{D}$, which provides an (infinite) sequence of natural numbers. At time $t = 0$, the system obtains its state. Then $\mathcal{D}(0)$ gives the number of time steps until the state information will be obtained next; after that occurs, $\mathcal{D}(1)$ is the offset to the next observation, and so forth. Thus, if $\mathcal{D} = 1111\cdots = (\overline{1})$ then the PS-MDP is a classic MDP, as, after each step, the state will be obtained in the very next step. When there is some k so that $\forall n, \mathcal{D}(n) = k$, or $\mathcal{D} = (k, k, k, \ldots) = (\overline{k})$, then the PS-MDP corresponds to the periodically state-observed MDP of [8], with the check-in period equal to k.

B. Solutions to PS-MDPs

Following the traditional performance measure for MDPs, it is natural to provide a discount factor $\gamma_{\text{ex}} \in (0, 1)$ and consider the *expected discounted cumulative cost* to reach a goal state. The expectation provides a concrete objective even though the process's evolution is stochastic, allowing aspects of uncertainty, e.g., the initial state may only be specified by a distribution $S_0 : S \to [0, 1]$. Also, the dynamics expressed by T are uncertain: given an action to perform, the outcome is only known up to a distribution before being executed. For standard MDPs, those actions come from a *policy*, a map $\pi : S \to A$, that gives an action for the realized state. An optimal policy is one minimizing the performance measure. The Markov property implies that the current state in an MDP suffices to determine the optimal action for each step.

A policy as a direct map from states to actions will not work for PS-MDPs because, in general, the current state is not known at every timestep. Indeed, for schedule $\mathcal{D}$, states are disclosed to the robot only at times: $(0, \mathcal{D}(0), \mathcal{D}(0) + \mathcal{D}(1), \ldots, \sum_{i=0}^{m-1} \mathcal{D}(i), \ldots)$. To relate timestep with observation occurrences, let $\lceil t \rfloor$ be the number of check-ins that have been received by the robot by time t, that is $\lceil t \rfloor := \min\{k : \mathbb{N} \mid t < \sum_{i=0}^{k-1} \mathcal{D}(i)\}$. A PS-MDP policy (or just policy), is a

$$\vec{\pi} : S \times \mathbb{N} \to A^{\infty}, \text{ s. t. } \vec{\pi}\big(s_t, \lceil t \rfloor\big) = \vec{a}^t \in A^{\mathcal{D}(\lceil t \rfloor - 1)}, \quad (1)$$

with $A^{\infty} := A \cup A^2 \cup A^3 \cup \cdots$, where $A^2 := A \times A$, $A^3 := A \times A \times A$, and $A^4 := A \times A \times A \times A$, etc. (The existence of such a policy for all PS-MDPs is proven in Section IV.)

For example, at $t = 0$, if schedule $\mathcal{D}(\lceil 0 \rfloor - 1) = \mathcal{D}(0) = 4$, then $\vec{\pi}(s_0, 0)$ must yield an ordered sequence of 4 actions. Those actions will be executed by the robot, and thereafter it will obtain the next state, s_4. And $\lceil 4 \rfloor = 2$, and the value $\mathcal{D}(1)$ gives the delay (in units of time) until the state will be known next, so $\vec{\pi}(s_4, 1)$ will give $\mathcal{D}(1)$ actions. Sequences of actions are generated as composite units, and executed without state feedback between them. The dimension of these composites is termed the *stride*. Considered in order, the strides must match the values in $\mathcal{D}$.

IV. The model

We consider a setting with two agents or parties: the *observation process* and the *actor*. Operationally, the observation process generates check-ins according to the schedule while the actor executes the policy. The observation process is interested in a notion of cost, distinct from C_{ex}, but related to the number of check-ins required. Note how $\mathcal{D}$ forms part of the PS-MDP's definition and, accordingly, the planning problem is solved *given* a schedule. It is sequential so, in game-like terms, the observation process selects the schedule first and then declares it; the actor seeks to minimize the cost C_{ex}, finding a policy subject to $\mathcal{D}$.

For the observation process's cost, consider the following:

$$\vec{C}_{\text{ck}}(s, \vec{a}) = \begin{cases} 0 & s \in G \\ 1 & s \notin G \end{cases} \text{ for all } s \in S, \vec{a} \in A^{\infty}. \qquad (2)$$

As $\vec{C}_{\text{ck}} : S \times A^{\infty} \to \mathbb{R}$, it considers sequences of actions—we term such a function a *macro cost*. The unit penalty incurred in (2) models the fact that a check-in is generated at the end of a whole sequence of actions. The longer the stride in a $\mathcal{D}$, the more that penalty will be amortized across time steps (or, equivalently, elementary actions).

Though we have stated that the actor's objective involves finding some cost minimizing $\vec{\pi}$, we must first show that the concept in (1) is indeed appropriate for PS-MDPs. This is point of Proposition 6.

A. Policies for PS-MDPs

First, we need the following two definitions:

Definition 3. *For transitions $T : S \times A \times S \to [0,1]$, the* macro transition model *is the function $\vec{T} : S \times A^{\infty} \times S \to [0,1]$ defined as*

$$\vec{T}(s', \vec{a}, s) = \vec{T}\left(s', (a_1, a_2, \ldots, a_{|\vec{a}|}), s\right) = \sum_{\substack{(s_1, \ldots, s_{|\vec{a}|}) \in S^{|\vec{a}|} \\ \text{where } s_0 = s \text{ and } s_{|\vec{a}|} = s'}} \prod_{i=1}^{|\vec{a}|} T(s_{i+1}, a_i, s_i).$$

This is, essentially, just convolving the basic transition dynamics; the number of times is determined directly from the length of the $\vec{a}$ argument. We can do the same thing to turn C_{ex} into a macro cost function:

Definition 4. *For discounting factor $\gamma_{\text{ex}} \in (0,1)$ and cost function $C_{\text{ex}} : S \times A \to \mathbb{R}$, the corresponding* macro execution cost *is $\vec{C}^{\gamma}_{\text{ex}} : S \times A^{\infty} \to \mathbb{R}$ defined as*

$$\vec{C}^{\gamma}_{\text{ex}}(s, \vec{a}) = \vec{C}^{\gamma}_{\text{ex}}\left(s, (a_1, a_2, \ldots, a_{|\vec{a}|})\right) = \sum_{k=1}^{|\vec{a}|} \gamma_{\text{ex}}^{(k-1)} \sum_{\substack{(s_1, \ldots, s_{|\vec{a}|}) \in S^{|\vec{a}|} \\ \text{where } s_1 = s}} C_{\text{ex}}(s_k, a_k) \prod_{i=1}^{|\vec{a}|} T(s_{i+1}, a_i, s_i).$$

Notice that the γ_{ex} ensures costs incurred later, owing to sequences of actions, are diminished correctly.

Macro costs allow for the following notion of PS-MDPs policy evaluation.

Definition 5. *The* evaluation of policy $\vec{\pi} : S \times \mathbb{N} \to A^{\infty}$ *from state s on* PS-MDP $\mathcal{M}$ *with respect to discount $\gamma \in (0,1)$ and macro cost function $\vec{C} : S \times A^{\infty} \to \mathbb{R}$ is*

$$V_{\vec{C},\gamma}(s, \vec{\pi}; \mathcal{D}) = \vec{C}(s, \vec{\pi}(s)) + \gamma \sum_{s' \in S} \vec{T}(s', \vec{\pi}(s), s) V_{\vec{C},\gamma}(s', \vec{\pi}; \mathcal{D}).$$

In evaluating from some initial state distribution, we will write $V_{\vec{C},\gamma}(S_0, \vec{\pi}; \mathcal{D})$, for the expected cost with state $s^0 \sim S_0$. (The preceding has been defined for generic macro costs to be used when applied with γ_{ck}, $\vec{C}_{\text{ck}}$ and also γ_{ex}, $\vec{C}^{\gamma}_{\text{ex}}$.) The following holds for macro costs in general too:

Proposition 6. *Given some* PS-MDP $\mathcal{M}$*, discount $\gamma \in (0,1)$, macro cost $\vec{C} : S \times A^{\infty} \to \mathbb{R}$, and initial state distribution S_0, there always exists a stationary and deterministic* PS-MDP *policy $\vec{\pi}^{\star} : S \times \mathbb{N} \to A^{\infty}$ such that:*

$$V_{\vec{C},\gamma}(S_0, \vec{\pi}^{\star}; \mathcal{D}) = \inf_{\vec{\pi}} V_{\vec{C},\gamma}(S_0, \vec{\pi}; \mathcal{D}).$$

Proof (Sketch). Construct a countably infinite MDP, associating a copy of S for each $\mathbb{N} \cup \{0\}$. Assuming the copies are indexed by r, the actions and costs are convolved using the construction in Definition 3 and 4, restricted to actions with $|\vec{a}| = \mathcal{D}(r)$, for each r. Then, the existence of a traditional optimal policy $\pi^{\star}$ on state-space $S \times \mathbb{N}$, producing composite actions with stride $\mathcal{D}(r)$ for states $(\cdot, r)$, is a suitable $\vec{\pi}^{\star}$. □

The preceding establishes that the formal objects involved do exist, have correct types and well-defined cost metrics.

B. Problem: Schedule Dominance

The notational heaviness is because we wish to express the evaluation of policies on different macro costs. For some PS-MDP $\mathcal{M} = \langle S, A, T, C_{\text{ex}}, G, \mathcal{D} \rangle$, execution discount γ_{ex}, check-in discount γ_{ck}, and initial distribution S_0, we say:

1) the *execution cost* of the pair $(\mathcal{D}, \vec{\pi})$ is the expected γ_{ex}-discounted cumulative cost to execute the policy:
$$\mathbf{E}(\mathcal{D}, \vec{\pi}) = V_{\vec{C}^{\gamma_{\text{ex}}}_{\text{ex}}, \gamma_{\text{ex}}}(S_0, \vec{\pi}; \mathcal{D});$$
2) the *check-in cost* of the pair $(\mathcal{D}, \vec{\pi})$ is the expected γ_{ck}-discounted number of check-ins until G is attained:
$$\mathbf{C}(\mathcal{D}, \vec{\pi}) = V_{\vec{C}_{\text{ck}}, \gamma_{\text{ck}}}(S_0, \vec{\pi}; \mathcal{D}).$$

For a given initial state distribution S_0, we will say some schedule $\mathcal{D}_2$ *dominates* $\mathcal{D}_1$ if $\forall \vec{\pi}_1 \exists \vec{\pi}_2$ s.t. $\mathbf{E}(\mathcal{D}_2, \vec{\pi}_2) < \mathbf{E}(\mathcal{D}_1, \vec{\pi}_1)$ and $\mathbf{C}(\mathcal{D}_2, \vec{\pi}_2) < \mathbf{C}(\mathcal{D}_1, \vec{\pi}_1)$. And also that $\mathcal{D}_1$ is *non-dominated* in some class of schedules $\mathscr{D}$, if there is no other $\mathcal{D}_2 \in \mathscr{D}$ that dominates $\mathcal{D}_1$. To help improve practicality, Section IV-D seeks to approximate this criterion via a set of scalarizations.

We can now state the general problem to solve:

General Problem. Let S, A, T, C_{ex}, G, and S_0 be given. For a class of schedules $\mathscr{D}$, compute the non-dominated schedules on PS-MDPs, in terms of $\mathbf{E}(\cdot,\cdot)$ and $\mathbf{C}(\cdot,\cdot)$.

Note how this contrasts with standard treatment of a traditional MDP, where the solution is only a policy, which does not depend upon the initial state distribution.

C. Subclasses of Schedules

If the problem is to be concrete, some specific and meaningful class $\mathscr{D}$ of schedules must be identified; for an implementation, some practical means is needed to describe infinite schedules. It would be ideal, moreover, if (1.) it was concise and convenient; (2.) howsoever the sequences are circumscribed, the restriction should 'wash out' in the limit to infinity; (3.) the choice afforded opportunities to exploit problem-specific constraints.

Accordingly, as already introduced in Definition 1, we consider schedules with two restrictions:

Bounded stride can be useful to capture structure of particular problem settings. For instance, in the rendezvous scenario, there is a maximum duration before the underwater gliders must surface, which provides a bound naturally.

Tail recurrence holds for any schedule employing the bar notation solely on the last element, for instance, the earlier example $\mathcal{D} = 13213\overline{2}$ is tail recurrent. Since the execution is discounted by γ, the effect of the tail diminishes quickly as prefixes are prepended.[1]

Now, using the preceding as a sort of 'regularity condition' on schedules, we can state the focus of our algorithm.

Computational Problem. Let S, A, T, C_{ex}, G, and S_0 be given. For the tail recurrent schedules with stride bounded by σ, compute the non-dominated schedules on PS-MDPs, in terms of $\mathbf{E}(\cdot,\cdot)$ and $\mathbf{C}(\cdot,\cdot)$.

D. Bounds on Schedules' Costs

If it were easy to enumerate every possible policy, a given schedule's costs could be computed and a Pareto front for that schedule alone constructed; dominance across schedules could be determined by comparison of those fronts. The question, thus, becomes one of determining, in a computationally efficient way, the Pareto front for each schedule.

We can approximate the front of a schedule in the form of a bounded region representing the best we *know* a schedule can do and the best the schedule can *hypothetically* do. For a given schedule, there is a policy $\vec{\pi}^{\star}_{\text{ex}}$ which minimizes the execution cost $\mathbf{E}$, and a policy $\vec{\pi}^{\star}_{\text{ck}}$ that minimizes the check-in cost $\mathbf{C}$. We can disregard any policy dominated by $\vec{\pi}^{\star}_{\text{ex}}$ or $\vec{\pi}^{\star}_{\text{ck}}$ — a region denoted in dark blue in Figure 1. As $\vec{\pi}^{\star}_{\text{ex}}$ is the optimal policy with respect to execution cost, there can be no policies to its left, and similarly there can be no policies below $\vec{\pi}^{\star}_{\text{ck}}$. Of interest is the box formed with $\vec{\pi}^{\star}_{\text{ex}}$ and $\vec{\pi}^{\star}_{\text{ck}}$ as its corners:

Definition 7. *For* PS-MDP $\langle S, A, T, C_{\text{ex}}, G, \mathcal{D}\rangle$ *and a set of known policies* $K = \{\vec{\pi}_1, \vec{\pi}_2, \ldots, \vec{\pi}_{|K|}\}$*, then*

1) *schedule* $\mathcal{D}$*'s* realizable front *is the set of all policies in* K *not dominated by any other policy in* K*;*
2) *schedule* $\mathcal{D}$*'s* optimistic front *is the set of best possible (but unknown) policies such that none are dominated by any other possible policy.*

One interpretation of the realizable front is an upper bound (in terms of cost) on the best case scenario. One is able to guarantee the existence of a policy that dominates any policies on or beyond the realizable front (and, when $K = \{\vec{\pi}^{\star}_{\text{ex}}, \vec{\pi}^{\star}_{\text{ck}}\}$, this is consistent with what was expressed above). The optimistic front corresponds to a lower bound on the best case scenario. Unlike the realizable front, we have no guarantee that a policy on the optimistic front will actually exist, but we do know that it is impossible to produce any better policy.

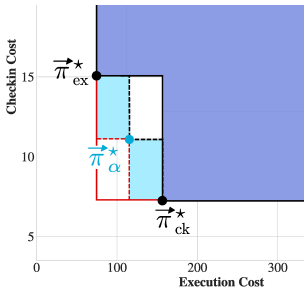


Fig. 1: Example showing characterization of a schedule with policy $\vec{\pi}$ as a point: $(\mathbf{E}(\mathcal{D}, \vec{\pi}), \mathbf{C}(\mathcal{D}, \vec{\pi}))$. Black lines mark the realizable fronts and red lines mark the optimistic fronts. Solid fronts are the specific bounds with $\vec{\pi}^{\star}_{\text{ex}}$ and $\vec{\pi}^{\star}_{\text{ck}}$, while dashed show the new fronts with the addition of an $\vec{\pi}^{\star}_{\alpha}$ policy, with the uncertainty region shown in light blue.

The region bounded by these two fronts represents uncertainty in knowledge of the schedule's true front. We incorporate this by considering a schedule $\mathcal{D}_1$ dominated by another schedule $\mathcal{D}_2$ if each point in $\mathcal{D}_1$'s optimistic front is dominated by some point in $\mathcal{D}_2$'s realizable front. The idea is that all optimal policies in $\mathcal{D}_1$, even hypothetical ones that may not actually exist, would be worse than a realizable policy in $\mathcal{D}_2$. This is safe treatment of dominance, as no schedule is incorrectly marked as dominated, but the price of uncertainty is in potentially maintaining an excess of non-dominated schedules.

E. Alpha Values

Accurately comparing one schedule to another requires that we reduce the uncertainty region in each schedule. Computing every possible policy collapses the region down to the true front, but would be exceedingly expensive to compute. Instead, we sample additional points in order to reduce (but not erase) the uncertainty. For a given $\alpha \in (0, 1)$, we can produce a policy minimizing a blend of $\mathbf{E}$ and $\mathbf{C}$, representing a linear scalarization of the problem. The intuition is that, while $\vec{\pi}^{\star}_{\text{ex}}$ minimizes along the horizontal and $\vec{\pi}^{\star}_{\text{ck}}$ minimizes along the vertical, the policy from an α-blend of costs will minimize along an intermediate line with the slope based on α; we denote the resulting policy by $\vec{\pi}^{\star}_{\alpha}$.

The benefit of generating a new policy $\vec{\pi}^{\star}_{\alpha}$ is two-fold. Firstly, we have a new known policy that we can provide for the schedule (and hence include in K), making it a new point in the realizable front. Secondly, since $\vec{\pi}^{\star}_{\alpha}$ is the optimal policy for its scalarization, it is impossible for another policy to have both lower check-in and lower execution cost, as that policy would then have a lower scalarized value. The new policy, therefore, tightens the lower bound by adding a point in the optimistic front. As demonstrated by the blue $\vec{\pi}^{\star}_{\alpha}$ point and the dashed fronts in Figure 1, computing new policies this way decreases the uncertainty region.

[1] Results presented subsequently will show this intuition to indeed be true.

V. Approach

A naïve approach to computing the set of non-dominated schedules would be to enumerate all schedules, computing each schedule's costs independently. Solving the problem in this way is prohibitively expensive due to exponential explosion. Hence, we next describe some insights to enable costs to be obtained much more efficiently.

A. Schedule Substructure

Firstly, while the number of schedules in the search space grows exponentially, they are clearly not independent. A schedule $(\mathcal{D}(0), \mathcal{D}(1), \mathcal{D}(2), \mathcal{D}(3), \dots)$ can be divided into two parts, a prefix $\mathcal{D}(0)$ and a suffix $(\mathcal{D}(1), \mathcal{D}(2), \mathcal{D}(3), \dots)$. As the suffix is itself also a schedule, we can define the costs of a schedule in terms of the costs of its suffix. We can also consider a PS-MDP policy $\vec{\pi} : S \times \mathbb{N} \to A^\infty$ as a series of policies $\pi_i : S \to A^{\mathcal{D}(i)}$, with a policy for each stride in the schedule. The Markov property implies that past policies in execution time depend on future policies but not vice versa, which we leverage by working backwards from the last policy to the first. Tail-recurrent schedules give us a natural starting point and base case—the recurrent tail $\overline{k}$, for which we can run standard value iteration (after employing Definitions 3–4) to convergence for the last policy. We next show that the policy for the stride immediately before, representing $\mathcal{D} = (\kappa, \overline{k})$, can be incrementally computed through a process dubbed *schedule extension*:

Proposition 8 (Schedule extension)**.** *Let* PS-MDP $\mathcal{M} = \langle S, A, T, C_{\text{ex}}, G, \mathcal{D} \rangle$ *be given with value function* $Q_{\vec{C},\gamma}$ *and policy evaluation* $V_{\vec{C},\gamma}$ *for policy* $\vec{\pi}$ *under discount* $\gamma \in (0, 1)$ *and macro cost function* $\vec{C}$*. Then value function* $Q'_{\vec{C},\gamma}$ *and policy evaluation* $V'_{\vec{C},\gamma}$ *for policy* $\vec{\pi}'$ *for the* $\mathcal{M}' = \langle S, A, T, C_{\text{ex}}, G, \mathcal{D}' \rangle$ *where* $\mathcal{D}' = (\kappa, \mathcal{D}(0), \mathcal{D}(1), \dots)$ *is:*

$$V'_{\vec{C},\gamma}(s, \vec{\pi}'; \mathcal{D}') = \vec{C}(s, \vec{\pi}'(s)) \quad + \gamma \sum_{s' \in S} \vec{T}(s', \vec{\pi}'(s), s) V_{\vec{C},\gamma}(s', \vec{\pi}; \mathcal{D}),$$

and

$$Q'_{\vec{C},\gamma}(s, \vec{a}; \mathcal{D}') = \vec{C}(s, \vec{a}) \quad + \gamma \sum_{s' \in S} \vec{T}(s', \vec{a}, s)\, Q_{\vec{C},\gamma}(s', \vec{\pi}(s', 0); \mathcal{D}),$$

where $\vec{\pi}'(s, n) = \vec{\pi}(s, n-1)$*, for* $n \geq 1$*, and* $\vec{\pi}'(s, 0)$ *is obtained from* $\min_{\vec{a} \in A^\kappa} Q'_{\vec{C},\gamma}(s, \vec{a}; \mathcal{D}')$.

Proof (Sketch). The computation here is reminiscent of value iteration. In fact, if $\mathcal{D} = (\overline{\kappa})$, the $Q'_{\vec{C},\gamma}$ update is equivalent to a single pass of value iteration, which makes sense considering $\mathcal{D}' = (\kappa, \mathcal{D}(0), \mathcal{D}(1), \dots) = (\kappa, \overline{\kappa}) = (\overline{\kappa}) = \mathcal{D}$. When this is not the case, however, the key difference is the new values computed, $Q'_{\vec{C},\gamma}$, are separate from the old values used, $Q_{\vec{C},\gamma}$. One can think of this separation in terms of parallel, stacked *layers* of value functions. When an agent takes the first step with stride $\mathcal{D}'(0) = \kappa$, it moves from a state s to a state s' and also moves vertically from the layer of $\mathcal{D}'$ to the lower layer of $\mathcal{D}$, where it repeats with shorter and shorter schedules. The value starting from s is simply the expected cost from moving, $\vec{C}(s, \vec{a})$, plus the value of the shorter schedule continuing from s', represented by the lower layer. The same argument applies for the policy evaluation. Indeed, the only difference between the two is that the value function selects the minimum cost action while the policy evaluation selects the policy action. □

The fact that schedules may, thus, be constructed from those of shorter length is the basis of the approach—the overall algorithm for which is presented in Algorithm 1. Indeed, every schedule in the search space can be recursively constructed from shorter ones, all the way down to the σ base cases of recurring tail schedules $\overline{\kappa}$. As shown in lines 4–9 of Algorithm 1, these base PS-MDPs can each be represented as a periodically state-observed Markov Decision Process (PSO-MDP) introduced in [8] and solved as ordinary MDPs, yielding a set of converged value functions for use as a base for schedule extensions.

Extending the length of a schedule by adding a prefix requires only a single pass; the key schedule extension procedure appears in Algorithm 2. The overlapping substructure allows us to apply a dynamic programming approach to constructing schedules, demonstrated in lines 13–18 of Algorithm 1, re-using values for already constructed schedules that make up each suffix rather than computing them anew.

B. Filtering

The recursive construction process markedly reduces the work done per schedule, but the overall complexity remains significant (exponential). As schedule lengths grow, a mechanism is needed to reduce the number of candidate schedules to extend. By filtering out schedules, fewer reach the final stage of Pareto front calculations, resulting in faster computation but potentially lower solution quality. As such, care must be taken in how schedules are filtered out.

One approach would be to pick only the best schedules at each stage, using the same criteria as the final Pareto front: the initial distribution's expected execution and check-in costs for each schedule. This approach, however, fails as a poor schedule with respect to the initial distribution may result in a good one once extended and vice versa. (We explore this as a case study later in Section VI-A.2.)

To mitigate this issue, we turn to a different evaluation criteria when filtering. Schedule suffixes do not start from the initial distribution and, as such, their policy values are not accurately represented when evaluated against the initial distribution; to alleviate this, we examine them against the other states as well. One such method is to use the uniform distribution, i.e., taking the average of the costs over every state. Rather than selecting schedules that work well starting in a limited number of states, we select schedules that work well on average. This filtering is done at the end of every extension stage, as seen in lines 19–26 of Algorithm 1.

We further tune the aggressiveness of the filtering with a *margin* parameter that also keeps schedules whose policy values are within a prescribed distance from the Pareto front,

rather than just the ones on the front. Another option is to use multiple independent distributions and be able to favor certain states like the greedy approach, while still having widespread support. Each distribution provides two additional dimensions in the Pareto front, resulting in less aggressive filtering. Taken to the extreme, one can select the collection of $|S|$ distributions with a Dirac delta distribution over each and every state, but in most cases not a single schedule will be dominated in the higher dimensional Pareto front and, as a result, none will be filtered out.

Algorithm 1 Algorithm with filtering.

PARETOFRONTSCHEDULES(
$\mathcal{M}$, the base MDP representing the environment
strides, the set of available check-in periods
length, the maximum length schedule to explore
$\mathbf{S_0}$, initial distribution of start states
filter, whether to filter out schedules during building
distribs, the set of distributions to use in filtering
margin, distance from front to keep during filtering
alphas, the alpha values for intermediary policy computation)

allSchedules $\leftarrow \{\}$ ▷ All candidate schedules
stages[0] $\leftarrow \{\}$ ▷ Sets for schedules of each length
▷ Construct initial $\overline{k}$ schedules
for all $k \in$ strides **do**
▷ Construct composite action process MDP with k-length macro-action sequences
$\mathcal{M}_k \leftarrow$ COMPOSITEMDP$(\mathcal{M}, k)$ ▷ See Composite Action Process in [8]
$\vec{C}_{\text{ck}}(s, \vec{a}) \leftarrow \vec{C}^{\gamma}_{\text{ex}}(s, \vec{a})$ with action costs set to 1 as per Eq. (2)
▷ Policy evaluations for $\vec{\pi}^{\star}_{\text{ex}}$ and $\vec{\pi}^{\star}_{\text{ck}}$: $V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot,\cdot\,;\mathcal{D})$ for execution cost and $V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot,\cdot\,;\mathcal{D})$ for check-in cost (see IV-B)
$V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}\,;\mathcal{D}) \leftarrow$ solve $\mathcal{M}_k$ w.r.t. $\vec{C}^{\gamma}_{\text{ex}}$ for $\vec{\pi}^{\star}_{\text{ex}}$
$V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}\,;\mathcal{D}) \leftarrow$ POLICYEVAL$(\vec{C}_{\text{ck}}, \mathcal{M}_k, \vec{\pi}^{\star}_{\text{ex}})$
$V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}\,;\mathcal{D}) \leftarrow$ solve $\mathcal{M}_k$ w.r.t. $\vec{C}_{\text{ck}}$ for $\vec{\pi}^{\star}_{\text{ck}}$
$V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}\,;\mathcal{D}) \leftarrow$ POLICYEVAL$(\vec{C}^{\gamma}_{\text{ex}}, \mathcal{M}_k, \vec{\pi}^{\star}_{\text{ck}})$
$\mathcal{D} \leftarrow$ **Schedule**(Strides = $[k]$, $V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot,\cdot\,;\mathcal{D})$, $V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot,\cdot\,;\mathcal{D})$)
stages[0].append($\mathcal{D}$)
allSchedules.append($\mathcal{D}$)
▷ Extending schedules
for $i \leftarrow 1$ to (length $-$ 1) **do**
stages[i] $\leftarrow \{\}$
for all $(\mathcal{D}, k) \in$ stages[i $-$ 1] $\times$ strides **do**
▷ Add k to head of schedule
$\mathcal{D}' \leftarrow$PREPENDSCHEDULE$(\mathcal{D}, k, \mathcal{M}_k, \vec{C}^{\gamma}_{\text{ex}}, \vec{C}_{\text{ck}}, \text{alphas})$
stages[i].append($\mathcal{D}'$)
allSchedules.append($\mathcal{D}'$)
▷ Filtering
if filtering **then**
for all $(\mathcal{D}, d) \in$ allSchedules $\times$ distribs
$\mathbf{E}(\mathcal{D},\cdot)_d = V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(d,\cdot\,;\mathcal{D})$ and $\mathbf{C}(\mathcal{D},\cdot)_d = V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(d,\cdot\,;\mathcal{D})$
$P_R \leftarrow$ PARETOFRONT(all realizable fronts) ▷ see IV-D on Pareto fronts
$P \leftarrow$ all $\mathcal{D} \in$ allSchedules with optimistic front not dominated by P_R
$M \leftarrow$ all $\mathcal{D} \in$ allSchedules with optimistic front within margin distance of P_R
allSchedules $\leftarrow$ P $\cup$ M
stages[i] $\leftarrow$ stages[i] $\cap$ allSchedules
for all $\mathcal{D} \in$ allSchedules **do**
$\mathbf{E}(\mathcal{D},\cdot) = V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(S_0,\cdot\,;\mathcal{D})$ and $\mathbf{C}(\mathcal{D},\cdot) = V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(S_0,\cdot\,;\mathcal{D})$
$P_R \leftarrow$ PARETOFRONT(all realizable fronts)
$P \leftarrow$ all $\mathcal{D} \in$ allSchedules with optimistic front not dominated by P_R
return P

VI. ALGORITHM CHARACTERIZATION

A. Case Studies

We provide two case studies to better illustrate the reasoning behind the algorithm.

1) Corridor grid with two cadences in sequence: Our main case study is a grid world featuring a series of walls in sequence, directly motivated by the example in [8, Sect. III-B] showing that a higher frequency of check-ins is not always better. The central idea is that columns of walls are placed with a certain cadence.

Algorithm 2 Schedule extension procedure.

PREPENDSCHEDULE(
$\mathcal{D}$, the **Schedule** to extend
k, the stride to insert at the schedule head
$\mathcal{M}_k$, the composite MDP representing k-length macro actions
$\vec{C}^{\gamma}_{\text{ex}}, \vec{C}_{\text{ck}}$, the macro execution and check-in cost functions, respectively
alphas, the alpha values for intermediary policy computation)

$\mathcal{D}' \leftarrow$ copy of $\mathcal{D}$ ▷ Below, EXTPOLICYEVAL means extend policy evaluation
$\mathcal{D}'$.Strides.insert(0, k) ▷ Insert k as first element
▷ Execution cost of $\vec{\pi}^{\star}_{\text{ex}}$ is value function on $\vec{C}^{\gamma}_{\text{ex}}$ while check-in cost is evaluation of $\vec{\pi}^{\star}_{\text{ex}}$ on $\vec{C}_{\text{ck}}$.
$V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}{}'\,;\mathcal{D}') \leftarrow$ EXTENDVALUEFUNCTION$(V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}\,;\mathcal{D}), \mathcal{M}_k)$
$V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}{}'\,;\mathcal{D}') \leftarrow$ EXTPOLICYEVAL$(V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}\,;\mathcal{D}), \mathcal{M}_k, \vec{\pi}^{\star}_{\text{ex}}{}')$
▷ Check-in cost of $\vec{\pi}^{\star}_{\text{ck}}$ is value function on $\vec{C}_{\text{ck}}$ while execution cost is evaluation of $\vec{\pi}^{\star}_{\text{ck}}$ on $\vec{C}^{\gamma}_{\text{ex}}$
$V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}{}'\,;\mathcal{D}') \leftarrow$ EXTENDVALUEFUNCTION$(V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}\,;\mathcal{D}), \mathcal{M}_k)$
$V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}{}'\,;\mathcal{D}') \leftarrow$ EXTPOLICYEVAL$(V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}\,;\mathcal{D}), \mathcal{M}_k, \vec{\pi}^{\star}_{\text{ck}}{}')$
for each $\alpha \in$ alphas **do**
$U_\alpha \leftarrow \alpha \times V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\text{ex}}{}'\,;\mathcal{D}') + (1-\alpha) \times V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\text{ck}}{}'\,;\mathcal{D}')$
$\vec{C}_\alpha \leftarrow \alpha \times \vec{C}^{\gamma}_{\text{ex}} + (1-\alpha) \times \vec{C}_{\text{ck}}$
$\vec{\pi}^{\star}_{\alpha}{}' \leftarrow$ policy from U_α, $\vec{C}_\alpha$, and $\mathcal{M}_k$
$V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\alpha}{}'\,;\mathcal{D}') \leftarrow$ EXTPOLICYEVAL$(V_{\vec{C}^{\gamma}_{\text{ex}},\gamma_{\text{ex}}}(\cdot, \vec{\pi}^{\star}_{\alpha}\,;\mathcal{D}), \mathcal{M}_k, \vec{\pi}^{\star}_{\alpha}{}')$
$V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\alpha}{}'\,;\mathcal{D}') \leftarrow$ EXTPOLICYEVAL$(V_{\vec{C}_{\text{ck}},\gamma_{\text{ck}}}(\cdot, \vec{\pi}^{\star}_{\alpha}\,;\mathcal{D}), \mathcal{M}_k, \vec{\pi}^{\star}_{\alpha}{}')$
return $\mathcal{D}'$

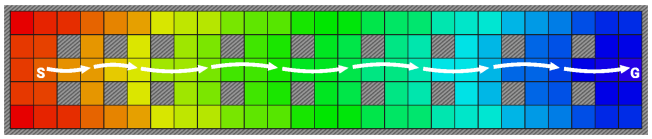


Fig. 2: The $\pi^\star$ policy for $22\overline{3}$ schedule in corridor grid world.

A cadence of 3 means that it takes 3 horizontal steps to reach one wall from another. The agent chooses between moving in a cardinal direction or waiting (a "no-op" action). Movement has a chance of causing an additional drift left or right of the movement direction, while no-ops incur no drifting. As a result, the agent prefers a check-in immediately before each wall column to verify it has not drifted vertically. The $\pi^\star$ policy, thus, has the agent wait for a check-in when it believes it has reached the preferred spot, rather than risk moving when the next check-in is still in the future. This behavior leads to an interesting phenomenon, where a stride that matches the environment cadence well will lead to fewer waiting delays than a more frequent check-in period.

One would expect that the optimal schedule would mirror the cadences: $22\overline{3}$. Figure 2 demonstrates how each stride moves to the spot before each column. Indeed, this schedule appears in the Pareto front shown in Figure 3, dominating both $\overline{2}$ and $\overline{3}$, and would be the optimal schedule if both check-in cost and execution cost were valued equally.

The Pareto fronts in Figure 3 correspond to the different stages of schedule extension. As schedules are extended, they form new fronts with existing schedules, progressing towards

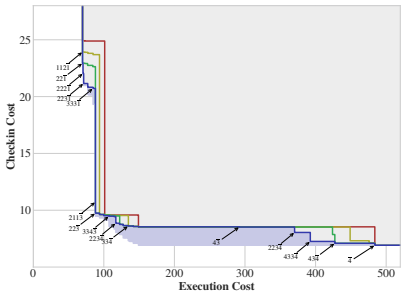
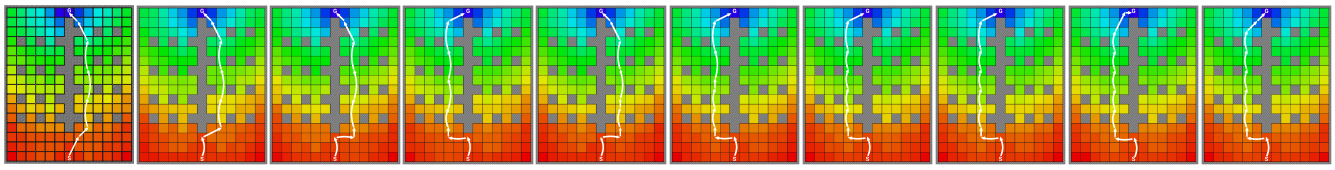


Fig. 3: Progression of the Pareto front (red, yellow, green, blue) as schedules get longer, up to length 4.



(a) $\overline{3}$ **(b)** $2\overline{3}$ **(c)** $22\overline{3}$ **(d)** $222\overline{3}$ **(e)** $2222\overline{3}$ **(f)** $22222\overline{3}$ **(g)** $2^6\overline{3}$ **(h)** $2^7\overline{3}$ **(i)** $2^8\overline{3}$ **(j)** $2^9\overline{3}$

Fig. 4: Optimal policies may switch between going left and going right when schedules are extended.

the final front for all schedules up to length 4. Note how most of the base schedules—$\overline{1}$, $\overline{2}$, and $\overline{3}$—never make it to the final front, yet appear as suffixes in those that do. They are sub-optimal candidates by themselves from the start state, but the prefixes bring the agent to an optimal state for the suffixes to continue, which is why it is important to evaluate the suffixes in states apart from the start when filtering.

The corridor world has a recurring goal reward of 10 000, movement cost of 1, zero no-op cost, collision cost of 300 000, drift probabilities of 5 % left and 5 % right of the intended direction, and a base discount factor of $\sqrt{0.99}$.

2) Splitter grid with two cadences in parallel: We further illustrate how schedule extension can result in non-trivial changes to the policy of a schedule, with the grid world in Figure 4. The world is constructed such that a $\overline{2}$ schedule would favor the west side due to the cadence 2 rows, while $\overline{3}$ would favor the east. Due to the center column, the agent cannot switch once it takes a side. At the start state, the choice is clear: go left if the stride is 2, go right if the stride is 3. As we start prepending $k = 2$ check-ins to the $\overline{3}$ schedule, however, we notice that the policy flips directions for certain schedules, with $222\overline{3}$ going leftwards instead. This occurs since the prefix of the schedule, 222, brings the agent to states on the left that are superior for the remainder of the schedule, $\overline{3}$, than if it had gone right. As we continue prepending more $k = 2$ check-ins, the flipping continues as the tail gets pushed farther north from the start state.

B. Time Complexity

The standard procedure for solving MDPs, value iteration, has time complexity $O(|S|^2|A|)$ per iteration but requires multiple iterations to converge. If we assume some I iterations to converge, the complexity for solving each base PS-MDP is $O(I|S|^2|A|^\sigma)$, as there are $|A|^\sigma$ actions in the composite MDP. Since our approach re-uses already computed schedules, it requires only the σ base convergences and $\sigma^n - \sigma$ extensions of a single pass per alpha. If we account for the proportion f of schedules filtered out, we have our final complexity of $O(|\alpha|(\sigma I + (1-f)\sigma^n)(|S|^2|A|^\sigma))$. In our experimental results, we have cases of f ranging from 0.12 (high margins) to 0.993 (low margins plus alphas).

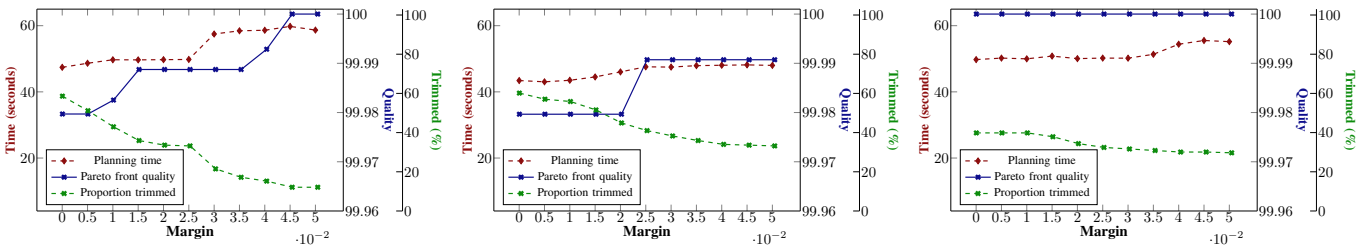
VII. Results

A. Margin and Distribution Effectiveness

Figure 5 shows the effect of the margin parameter on running time, quality, and filtering aggressiveness on the corridor case study. The quality metric is obtained by comparing the final Pareto front with the true front generated without filtering. A margin value of zero results in the most aggressive pruning, keeping only non-dominated schedules on each stage's Pareto front. As a result, it has the lowest running time and solution quality, both of which steadily increase with higher margin values as filtering decreases. Some distributions are better suited than others in certain aspects: Figure 5b shows how filtering with the initial distribution S_0 starts out with both a slightly higher solution quality and a lower running time than its uniform distribution counterpart in Figure 5a, but is much less effective in raising quality by increasing the margin. This is a result of the greedy pitfall: it succeeds at recognizing suffixes optimal from the start state, but fails to capture schedules with suffixes only optimal from a different state. It then requires a higher margin tolerance to avoid filtering the latter case out early. Figure 5c shows how using both distributions in parallel in a multi-dimensional Pareto front appears to balance both. The initial distribution ensures that the easy cases are captured, while the uniform distribution prevents the harder cases from being dropped—ultimately, it gives improved quality even without a margin.

B. Scaling

Figure 6 summarizes different problem sizes to demonstrates how filtering can significantly mitigate exponential growth of the search space. One can easily see how increasing schedule lengths rapidly becomes a problem with σ^n schedules: for $n = 9$ the unfiltered algorithm took 15 hours,



(a) Uniform distribution **(b)** Initial distribution **(c)** Both uniform and initial distributions

Fig. 5: Planning time and precision *vs* margin for check-in periods 1 to 4, up to schedule length 4 for different filtering distributions.

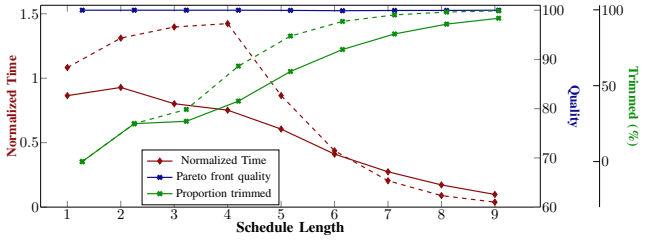


Fig. 6: Planning time and precision *vs* problem size, with alphas {0.2, 0.4, 0.6, 0.8} (dashed) and without (solid). Planning time normalized against running time without filtering.

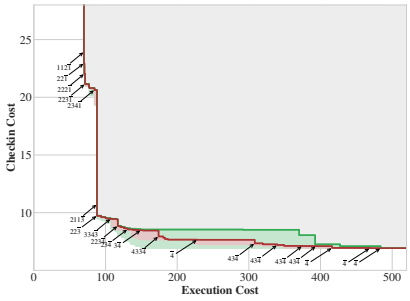


Fig. 7: Pareto fronts with alphas {0.1, 0.2, 0.3, 0.4, ..., 0.9} (red) and without (green) at length 4, with shaded regions between the optimistic fronts and the bold realizable fronts.

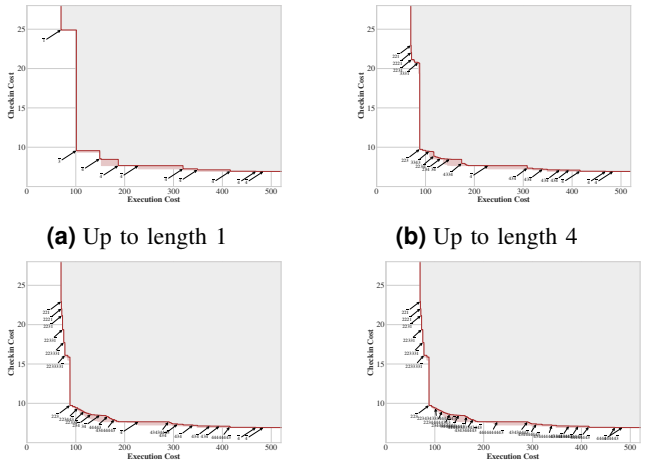
compared to 66 seconds for $n = 4$. As the problem size grows and filtering is able to trim out more schedules, we see that avoiding extending sub-optimal candidates improves the running time significantly without sacrificing precision.

We also see an interesting phenomenon appear in the introduction of alpha values. They increase running time for shorter schedules as one would expect, due to needing to compute the extra policies, but also allow filtering to be more effective at dismissing schedules from consideration, by reducing the region of uncertainty that each schedule occupies. This performance benefit is not enough to overcome the overhead from extra computation for shorter schedules, but when asking for longer schedules, the savings from increased trimming scales and reduces the computation time overall, bringing the filtered running time down from 90 minutes to 36 for $n = 9$. Their effect on Pareto fronts is also clearly visible in Figure 7. The green region is derived without alpha values—that is, with only $\overrightarrow{\pi}^{\star}_{\text{ex}}$ and $\overrightarrow{\pi}^{\star}_{\text{ck}}$ for each schedule. The red region uses 9 additional policies per schedule, resulting in much tighter realizable and optimistic fronts.

In Figure 8 we examine an even larger problem as we extend schedules to length 16. Large changes in the front are made at the start, but as schedules get longer, fewer non-dominated schedules are discovered. As schedule lengths surpass the size of the environment, longer schedules become beneficial only for the less and less likely cases of not reaching the goal earlier. In the figure we see that the longer non-dominated schedules become focused in a very small area. The filtering takes full advantage of this sparsity and diminishing returns. Otherwise, computing the entire roster of 4^{16} schedules would be quite impractical.

VIII. Conclusion

In tackling the problem of planning for a robot which relies on intermittent state observations from an external source, we examined methods for producing a pre-defined



(a) Up to length 1 **(b)** Up to length 4

(c) Up to length 8 **(d)** Up to length 16

Fig. 8: Progression of the candidate Pareto front (red) as schedules get longer, up to length 16. Filtering using initial distribution with no margin tolerance and 10 alphas (0.1, 0.2, ..., 0.9).

observation schedule suitable to both parties. The complication of execution and check-in costs being impacted by both the schedule and the policy to execute led us to formulate an approximate Pareto front, in which schedules are regions bounded by sub-fronts that can be improved through the use of alpha values. We introduced a dynamic programming algorithm that constructs schedules via accumulation through the use of a schedule extension procedure, computing incremental changes to value functions and policy evaluations. Further, we also proposed a filtering approach that prunes the working set to curb exponential growth. Our results demonstrated that this filtering scheme significantly reduces computation times for only negligible reductions in quality. Coupled with alpha values that tighten the bounds for more effective filtering and an overall performance gain for longer schedules, our algorithm allows for the computation of problem sizes that would not be feasible without filtering.

References

[1] Y. Aloimonos, *Active perception*. Mahwah, N.J.: LEA, Inc, 1993.
[2] R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.
[3] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *Autonomous Robots*, vol. 42, no. 2, pp. 177–196, 2018.
[4] M. A. Batalin, G. S. Sukhatme, and M. Hattig, "Mobile robot navigation using a sensor network," in *IEEE International Conference on Robotics and Automation (ICRA)*, Apr. 2004, pp. 636–641.
[5] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Belmont, M.A., U.S.A: Athena Scientific, 2019.
[6] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.
[7] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
[8] F. Rossi and D. A. Shell, "Planning under periodic observations: bounds and bounding-based solutions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2022.
[9] M. Roth, R. Simmons, and M. Veloso, "What to communicate? Execution-time decision in multi-agent POMDPs," in *Distributed Autonomous Robotic Systems (DARS) 7*. Springer, 2006, pp. 177–186.
[10] V. Unhelkar and J. Shah, "Contact: Deciding to communicate during time-critical collaborative tasks in unknown, deterministic domains," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.