Concise planning and filtering: hardness and algorithms

Jason M. O'Kane and Dylan A. Shell

Abstract—**Motivated by circumstances with severe computational resource limits (*e.g.*, settings with strong constraints on memory or communication), this paper addresses the problem of concisely representing and processing information for estimation and planning tasks. In this article, <u>conciseness</u> is a measure of explicit representational complexity: for filtering, we are concerned with maintaining as little state as possible to perform a given task; for the planning case, we wish to generate the plangraph (or policy-graph) with the fewest vertices that is correct and also complete. We present hardness results showing that both filtering and planning are NP-hard to perform in an optimally concise way, and that the related decision problems are NP-complete.**

We also describe algorithms for filter reduction and concise planning, for which these hardness results justify the potentially sub-optimal output. The filter-reduction algorithm accepts as input an arbitrary combinatorial filter, expressed as a transition graph, and outputs an equivalent filter that uses fewer information states to complete the same filtering task. The planning algorithm, using the filter-reduction algorithm as a subroutine, generates concise plans for planning problems that may involve both non-determinism and partial observability. Both algorithms are governed by parameters that encode tradeoffs between computational efficiency and solution quality. We describe implementation of both algorithms and present a series of experiments evaluating their effectiveness.

Note to Practitioners—**The reduced filters and plans explored in this paper are of practical interest in several contexts, including: *(i)* on robot platforms with severely limited computational power; *(ii)* communication over low-bandwidth noisy channels; *(iii)* a special instance of the previous case includes human-robot interaction settings where interfaces constrain information transfer; *(iv)* understanding the size and structure of concise plans or filters for a given problems provides insights into those problems (*e.g.*, to assess the value of a particular sensor by comparing the size of filters with or without it.)**

Index Terms—**Primary Topics: Robotics, Planning, Estimation Secondary Topics: Automata**

I. INTRODUCTION

THE DESIGNER of any autonomous robot system faces at least two central questions. First, the designer must decide how the robot should process and store information collected by its sensors. The answer to this question must account for the incompleteness and potential inaccuracy of that information, the computational capabilities of the robot, and the specific task that robot must complete. Second, the

Jason M. O'Kane (jokane@cse.sc.edu) is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, South Carolina, USA.

Dylan A. Shell (dshell@cse.tamu.edu) is with the Department of Computer Science and Engineering, Texas A&M University, College Station, Texas, USA.
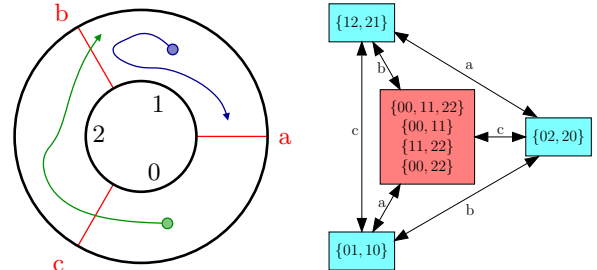

Fig. 1. [left] Two agents move amidst three beam sensors. [right] An optimal combinatorial filter, first discovered by Tovar, Cohen, and LaValle [30] and reproduced by our algorithm, for tracking whether the agents are in the same region. The numbers 0, 1, and 2 denote the regions, and the letters a, b, and c denote observations from each of the three beams.

designer must determine how the robot should select actions to generate task-oriented behavior, subject to the limited available information. In addition to sensor data, this may include the history of actions executed in the past and any *a priori* knowledge that might be available. These two problems—referred to in broad terms as *filtering* and *planning* respectively—are closely intertwined, since the best indicator of success in filtering is whether or not the information retained by the filtering process is adequate for decision-making [10].

This paper addresses the problem of concisely representing and processing information in planning and filtering tasks, examining the question of how to produce filters and plans with *minimal storage requirements*. That is, we are interested in maximally concise filters and plans subject to correctness for a given task (which entails either aggregating information, or selecting actions, in the respective cases). Specifically in filtering, one may be interested in constructing filters that retain information that is essential to completing a robot's task and strictly nothing further.

These questions are relevant when storage resources are at a premium, or when one is interested understanding of the nature of the requirements for a particular information processing task, perhaps, in order to compare objectively with another task. In the planning case, we are interested in producing a plan with minimal size, subject to correctness and completeness. Though the previous reasons also motivate consideration of conciseness for plans, a third motivation is that agents may communicate plans. Even in human to human interaction it is not altogether uncommon to prefer a shorter but adequate set of directions rather than a shorter but more complex route, especially when the communication channel is noisy.

The following two examples illustrate the basic problems addressed in this article.
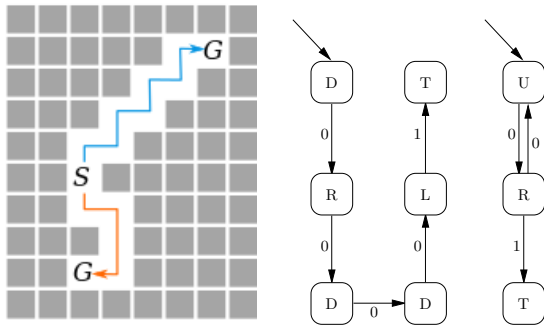
Fig. 2. [left] A planning problem in which a robot with a goal detector moves from S to G. [middle] A plan graph for this problem that minimizes execution time. [right] A plan graph for this problem of minimal size.

A. A concise filter

First, consider the filtering problem depicted in Figure 1, which was introduced by Tovar, Cohen, and LaValle [30]. In this problem, two agents move through an annulus-shaped environment. The environment is partitioned by three beam sensors that can detect when an agent crosses the beam, but cannot determine the identity of the agent nor the direction of the crossing. The goal is to design a filter that knows, at all times, whether the agents are in the same region (regardless of which region that might be) or in different regions.

An obvious approach to this problem is to define a nine-element state space $X = \{0, 1, 2\} \times \{0, 1, 2\}$, and to track the *nondeterministic information state (I-state)*—that is, the set of possible states—based on the beam crossings we observe. This approach requires one to track which of the $2^9 = 512$ distinct I-states is consistent with the observation history.

However, Tovar, Cohen, and LaValle observed that this problem can be solved using a filter with only four I-states: one I-state representing "the agents are together" and three I-states representing "the agents are separated by beam x" for each of the three beams. The right side of Figure 1 is graphical depiction of this filter. The vertices represent I-states and the directed edges show transitions that occur when a beam crossing is detected. In this paper, we address the problem of performing this kind of filter reduction algorithmically; heretofore this has required clever hand-crafting.

B. A concise plan

The value of compact expression is also evident for planning problems. Consider an idealized robot moving on a grid and in possession of a goal detection sensor, as shown in Figure 2. The robot's goal is to travel from its starting location (marked 'S') either of the two goals (marked 'G'). In this case, the plan with the smallest execution time travels directly to the lower goal. However this plan—informally, "Down, right, down, down, left, stop"—is more complex than the alternative that travels to the upper goal using a plan informally expressed as "Alternate up and right until reaching the goal." In this paper, we formalize this idea, and investigate methods for generating concise plans that solve a very general class of robotic planning problems.

C. Contributions

After a review of related research in Section II, this paper makes two new contributions.

For filters, we contribute to a recent line of research that considered solutions to this problem using *combinatorial filters* [21, 31, 33]. This existing work illustrates how adroit choices for state descriptions may lead to concise—or even minimal— filters that are tailored to retain only the information that is essential to completing the robot's specific task. With respect to filters, we may ask the following questions:

- How much space is required in order to perform some filtering task?
- What does the minimal-sized filter look like?
- How can we construct such minimal-sized filters?
- Can a filter be (losslessly) compressed or reduced?

We address these questions by introducing an algorithm that accepts as input an arbitrary combinatorial filter, expressed as a transition graph, and outputs an equivalent filter that uses fewer information states to complete the same filtering task. This algorithm is able to reproduce the hand-crafted solutions found in the literature. We also show that solving this problem optimally is NP-hard, and that the related decision problem is NP-complete. These hardness results justify the potentially sub-optimal output of our algorithm. Our results on concise filtering appear in Section III.

For plans, the underlying questions are:

- What is the most concise plan that solves a given planning problem?
- How can we find concise plans?

We present results on this front that dovetail with the filtering results summarized above: we prove that the problem of finding the *smallest* plan that solves a given problem is NP-hard. We introduce an algorithm that rapidly generates concise plans, albeit without any guarantee of optimality. Our approach involves two pieces: (*i*) reduction of a given plan to express it as concisely as possible, leveraging the connection between plans and filters; and (*ii*) an incremental search for plans that exploits structure in the solution space by reusing sub-parts of plans.

This study was motivated by circumstances with severe computational resource limits (*e.g.*, memory or communication constrained settings), but has implications more broadly. Perhaps its most significant contribution is an intellectual shift: it is the first work we are aware of that treats both filters and plan fragments as first-class objects, studying not merely those objects themselves, but rather theoretical properties of operations which can be automatically performed on them. Thus, the computational hardness results provide new insights into the nature of planning and filtering generally. The algorithms we introduce also form the basis of new tools for understanding the challenge of filtering and/or planning involving particular robot, sensor, and environment settings. Results on concise planning appear in Section IV.

Preliminary versions of this work appeared in two conference papers [24, 25]. This article integrates and extends the content of those two papers, including refinements, corrections, and clarifications throughout, along with additional

experimental evaluation.

II. RELATED WORK

The idea of understanding problems by examining the representational complexity needed to solve them can be traced back at least as far Kolmogorov's definition of the complexity of a string in terms of size of the smallest program that outputs that string [18]. Another family of well-known results considers the "power" of various sensors, such as abstract compasses [2] and pebbles [1], for exploration tasks. In that work, the power of a sensor is measured in terms of the amount of memory (finite, finite augmented with a single counter, *etc.*) required for an agent to explore its environment using that sensor.

More directly, the kinds of filters and plans we study in this paper have a long history—see, for example, the sensorless manipulation work of Erdmann and Mason [9] and Goldberg [11] which uses transition graphs to represent the evolution of a robot's uncertainty—and were formalized in a general way by LaValle [20, 21].

A. Combinatorial filtering

A number of recent papers have presented combinatorial filters for such tasks as target tracking [33], mobile robot navigation [22, 31], and manipulation [19]. However, that prior research relies upon careful human analysis of specific problem types and often seeks only to find feasible, rather than optimal, filters. To the best of our knowledge, this article is the first to address the question of automatic reduction of combinatorial filters.

The I-state graphs we consider are a special case of the nondeterministic graphs recently studied by Erdmann [7, 8]. That work is primarily concerned with topological conditions on the existence of plans to reach certain goals in such a graph, rather than with reducing the size of the graph itself.

Another thread of research has proposed systematic simplifications of geometric information spaces [23, 27] by approximating the I-states with simple geometric shapes. That work is more general than the present research because it does not require the observation and information spaces to be finite, but it relies on experimental data as evidence that the underlying tasks can still be completed, in contrast to the provable equivalence provided in this paper.

Roy, Gordon, and Thrun [26] also automate reduction of representational detail, but within a probabilistic planning setting using POMDPs. They apply dimensionality reduction techniques to reduce computational requirements needed to solve for policies. In contrast, the I-state model allows one to minimize state without requiring as rich a transition model, and enables the hardness result produced herein.

B. Plan graphs

The class of planning problems we consider in this paper is equivalent to the class of nondeterministic graphs that appears in Erdmann's recent topological conditions on the existence of plans that succeed in such graphs [8]. Such graphs, commonly represented as AND-OR graphs, have received attention by AI researchers employing heuristics to find a solution to reach a goal [3, 14]. The results we present here are orthogonal, in the sense that our results are algorithmic, and focused constructing on optimally concise, rather than merely extant, plans. Szczerba *et al.* [28] emphasized the importance of a variety of diverse path metrics and introduced an algorithm for computing paths with the fewest turns. In the special case where rotating in place is one action and linear navigation is another, their geometric path planner could produce (sensorless) plans with minimal expression complexity.

Likewise, the plan graphs we use here to represent the robot's strategy as a finite state machine have also been used in the context of POMDPs [13].

III. CONCISE COMBINATORIAL FILTERS

This section considers the problem of filter reduction, including some basic definitions (Section III-A), a hardness result (Section III-B), a heuristic algorithm (Section III-C), and some experimental results (Section III-D).

A. Definitions

We consider filters that process a sequence of discrete *observations* from a finite *observation space* which we denote by Y. Each observation $y \in Y$ corresponds to a distinct unit of information that becomes available to the filter; typically we think of these as readings produced from sensors, but a passive filter on a robot could potentially observe each *action* that the robot executes. In the context of filter reduction, the difference between observations and actions is irrelevant. Therefore, within this section, we use the term "observations" exclusively. (We defer planning problems, in which the difference between actions and observations is vital, and in which actions must be chosen rather than merely observed passively, to Section IV.)

Following the terminology championed by LaValle [20], we use the term *information state (I-state)* to refer to any representation of the information available to the system, derived from the history of observations it has received. Because the observation space is finite, we can describe the changes in the I-state using a transition graph.

Definition 1: *A passive I-state graph $\mathbf{G}$ is an edge-labelled directed multigraph with a start vertex. That is: $\mathbf{G} \triangleq (V, E, l : E \to Y, v_0)$, in which*

1) *the finite set V contains vertices which we call "I-states",*
2) *the set $E \subseteq V \times V$ consists of ordered pairs of vertices termed directed edges,*
3) *each edge is labelled with an observation via the function l, and*
4) *the starting I-state is identified as $v_0 \in V$.*

In addition, no two edges originating from the same vertex may have the same label. That is, if $e_1 \triangleq (v, v_j)$ and $e_2 \triangleq (v, v_k)$ with $v_j \neq v_k$ then $l(e_1) \neq l(e_2)$. For convenience, we write $v_1 \xrightarrow{y} v_2$ for an edge from v_1 to v_2 bearing label y.

Given a sequence of observations $y_0 y_1 \cdots y_n$, these may be traced on $\mathbf{G}$ by starting at v_0 and following the edges labelled

by each y_i, successively. When all the corresponding edges exist, the resulting I-state is uniquely determined. However, for some I-states, the I-state graph may lack outgoing edges for some observations. This can occur, for example, when the structure of the underlying problem indicates that an observation cannot occur at a given I-state. Definition 1 allows this because it does not require that the image of l be its entire codomain. For observation sequences under which this occurs, the resulting I-state is undefined.

Because we are primarily interested in the behavior of I-state graphs for observation sequences whose resulting I-states are well-defined, we define the language of strings for which this is the case.

Definition 2: *The* language induced by an I-state graph **G**, *denoted as* $\mathcal{L}(\mathbf{G}) \subseteq Y^\star$, *is the set of all sequences of observations (e.g.,* $y_0 y_1 \cdots y_n$*) for which valid transitions may be traced on* **G** *by starting at its initial vertex.*

We can now consider the kinds of tasks that one may wish to use an I-state graph to perform.

Definition 3: *A filter is an I-state graph supplemented with a coloring of its vertices. That is,* $\mathbf{F} \triangleq (V, E, l, v_0, c : V \to \mathbb{N}^+)$, *in which* (V, E, l, v_0) *is an I-state graph, and the function* c *assigns a natural number to each I-state.*

The interpretation is that observations are made and information retained, and the color of each visited I-state is reported as the filter output after each observation. The coloring describes the task performed by filter and, thus, represents the degree of fineness to which information is required. For example, in a planning problem in which the goal is to reach some I-state in a given class of goal I-states, one might form a "goal detection filter" by choosing c to assign color 1 to every goal I-state and color 2 to every non-goal I-state. By using more than two colors, arbitrarily complex filtering tasks can be defined.

Our goal in this section is to reduce size of these kinds of filters without impacting their correctness at completing a given task. This requires a precise notion of equivalence between two filters.

Definition 4: *Two filters* $\mathbf{F_1} \triangleq (V, E, l, v_0, c_1)$, *and* $\mathbf{F_2} \triangleq (W, F, m, w_0, c_2)$ *with a common observation space* Y *are said to be* equivalent with respect to a language $\mathcal{L} \subseteq Y^\star$ *if, for every observation sequence* $s \in \mathcal{L}$, *the I-states* v^s *and* w^s *reached by tracing* s *on* $\mathbf{F_1}$ *and* $\mathbf{F_2}$ *respectively are both defined, and we have* $c_1(v^s) = c_2(w^s)$. *We denote this equivalence relation with* $\mathbf{F_1} \stackrel{\mathcal{L}}{=} \mathbf{F_2}$.

The intuition is that if, for any string in a given language, both $\mathbf{F_1}$ and $\mathbf{F_2}$ produce well-defined and identical outputs, then $\mathbf{F_1}$ and $\mathbf{F_2}$ are equivalent.

Given an initial filter **F** as the specification of a filtering task, we are concerned with other filters that are equivalent on the language induced by **F**, that is, those filters $\mathbf{F}'$ where $\mathbf{F} \xrightarrow{\mathcal{L}(\mathbf{F})} \mathbf{F}'$. (Note that the reverse is not necessarily true, since $\mathbf{F} \xrightarrow{\mathcal{L}(\mathbf{F})} \mathbf{F}' \not\Rightarrow \mathbf{F} \xrightarrow{\mathcal{L}(\mathbf{F}')} \mathbf{F}'$.) Comparing the cardinality of the vertex sets of **F** and $\mathbf{F}'$, one obtains a relative measure of the

memory required by implementations of either filter. It is thus natural to consider the *filter minimization problem*:

Problem: Filter Minimization (FM)

Input: A filter **F**.

Output: A filter $\mathbf{F}^\star$ such that $\mathbf{F} \xrightarrow{\mathcal{L}(\mathbf{F})} \mathbf{F}^\star$ and the number of I-states in $\mathbf{F}^\star$ is minimal.

The balance of this section is concerned with algorithmic solutions to this problem.

B. Hardness of filter minimization

This section presents a hardness result for the filter minimization problem FM. Following the usual technique, we first convert FM to a decision problem:

Decision Problem: Filter Minimization (FM-DEC)

Input: A filter **F** and a desired size $k \in \mathbb{N}^+$.

Output: *True* if there exists a filter $\mathbf{F}'$ with at most k I-states, such that $\mathbf{F} \xrightarrow{\mathcal{L}(\mathbf{F})} \mathbf{F}'$; *False* otherwise.

We show in this section is that FM-DEC is an NP-complete problem, which directly implies that FM is NP-hard. The structure of the proof is first to argue that FM-DEC is in complexity class NP and then to provide a polynomial time reduction from the problem of 3-coloring a graph—a known NP-complete problem—to FM-DEC. Finally, we briefly discuss the relationship between this problem and the closely related (but efficiently solvable) problem of minimizing deterministic finite automata.

1) Filter minimization is in NP: To prove that FM-DEC is in NP, it suffices to show that, *given* the reduced filter $\mathbf{F}'$, we can verify its correctness in polynomial time. The first condition on $\mathbf{F}'$—that it has at most k vertices—is trivial to confirm. It remains to show how we can, given two filters $\mathbf{F_1}$ and $\mathbf{F_2}$, efficiently determine whether $\mathbf{F_1} \xrightarrow{\mathcal{L}(\mathbf{F_1})} \mathbf{F_2}$.

Algorithm 1 shows a method to perform this test. The intuition is to imagine both $\mathbf{F_1}$ and $\mathbf{F_2}$ working in parallel to filter some observation sequence from $\mathcal{L}(\mathbf{F_1})$. The algorithm uses a forward search to generate and examine each pair of vertices $(v_1, v_2) \in V_1 \times V_2$ that can be reached during any such simultaneous execution. If $\mathbf{F_2}$ produces correct colors for every possible observation at every reachable state pair, then the filters must be equivalent.

Note that the outer loop of Algorithm 1 (lines 3–16) executes at most $|V_1||V_2|$ iterations, that the inner loop (lines 5–15) executes at most $|Y|$ iterations, and that the remaining operations can be completed in constant time. Therefore, Algorithm 1 runs in $O(|V_1||V_2||Y|)$ time. The existence and polynomial run time of this algorithm lead directly to the following result.

Lemma 1: Filter minimization is in complexity class NP.

2) Filter minimization is NP-complete: Next we prove that, unless $\mathrm{P} = \mathrm{NP}$, determining whether such a reduced filter exists is a computationally intractable problem. We proceed by reduction from a standard graph coloring problem:

Algorithm 1 Filter Equivalence Test

Input:

> Two filters $\mathbf{F_1} \triangleq (V_1, E_1, l_1 : E_1 \to Y, v_1, c_1)$
> and $\mathbf{F_2} \triangleq (V_2, E_2, l_2 : E_2 \to Y, v_2, c_2)$

Output:

> True *if* $\mathbf{F_1} \xLeftrightarrow{\mathcal{L}(\mathbf{F_1})} \mathbf{F_2}$, *or* False *otherwise.*

1: **if** $c_1(v_1) \neq c_2(v_2)$ **return** False
2:   Q.insert $((v_1, v_2))$ {Initialize queue with the start vertices.}
3: **while** Q is not empty **do**
4:     $(v_1, v_2) \leftarrow Q$.pop()
5:     **for** each edge $v_1 \xrightarrow{y} w_1$ in E_1 **do**
6:       **if** E_2 has an edge $v_2 \xrightarrow{y} w_2$ **then**
7:         **if** $c_1(w_1) \neq c_2(w_2)$ **then**
8:           **return** False {$\mathbf{F_2}$ produces an incorrect color.}
9:         **else if** (w_1, w_2) has not been enqueued before **then**
10:           Q.insert $((w_1, w_2))$
11:         **end if**
12:       **else**
13:         **return** False {$\mathbf{F_2}$ terminates where $\mathbf{F_1}$ does not.}
14:       **end if**
15:     **end for**
16: **end while**
17: **return** True {No discrepancies for any reachable state pair.}

> **Decision Problem: Graph 3-Coloring (**GRAPH-3C**)**
>
>   *Input:* An undirected graph $\mathbf{G}$.
>   *Output: True* if there exists coloring of $\mathbf{G}$ using at most 3 colors, such that no pair of adjacent vertices shares the same color; *False* otherwise.

This problem is known to be NP-complete [5]. Therefore, it suffices for us to show a polynomial time reduction from GRAPH-3C to FM-DEC.

Given an undirected graph $\mathbf{G_1} \triangleq (V_1, E_1)$ as an instance of GRAPH-3C, we construct an instance of FM-DEC with filter $\mathbf{F_2} \triangleq (V_2, E_2, l, v_0, c)$ and size bound k as follows:

1) Create a start vertex in V_2 called v_0. Define $c : v_0 \mapsto 1$.
2) Create additional vertices in V_2, one for each vertex in V_1. For each such vertex v, assign $c : v \mapsto 2$.
3) For each vertex $v_i \in V_1$, create a new observation y_i and a new edge $v_0 \xrightarrow{y_i} v_i$ in E_2, by ensuring that (v_0, v_i) is in E_2 and $l : (v_0, v_i) \mapsto y_i$.
4) Create two additional vertices in V_2 named v_+ and v_-. Let $c : v_+ \mapsto 3$ and $c : v_- \mapsto 4$.
5) For each edge (v_i, v_j) in E_1, create a new observation y_{ij} in Y and two new edges $v_i \xrightarrow{y_{ij}} v_+$ and $v_j \xrightarrow{y_{ij}} v_-$, *i.e.*, (v_i, v_+) and (v_j, v_-) are in E_2, and $l : (v_i, v_+) \mapsto y_{ij}$ and $l : (v_j, v_-) \mapsto y_{ij}$. (Since edges in E_1 are undirected, an arbitrary direction may be selected in applying this rule.)
6) Set $k = 6$.

The intuition behind this construction is to "embed" the original graph $\mathbf{G_1}$ into filter $\mathbf{F_2}$ in such a way that vertices in V_2 are forced to remain separate in any reduced filter equivalent to $\mathbf{F_2}$. Figure 3 shows an example of this construction.

The algorithm to perform this construction clearly runs in time linear in the size of $\mathbf{G_1}$. Therefore, it remains for us to show that $\mathbf{G_1}$ is 3-colorable if and only if there
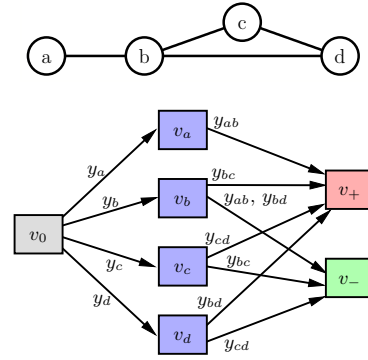

Fig. 3. [top] An example instance of GRAPH-3C. [bottom] The corresponding instance of FM-DEC. The vertices in the left column have color 1, the middle column has color 2, and the vertices in the right column have colors 3 and 4. Our constructed instances use $k = 6$.
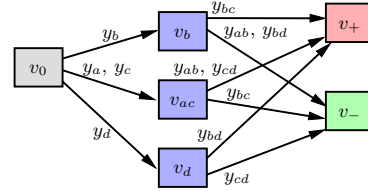

Fig. 4. A filter equivalent to the filter shown in Figure 3, formed by a "merge" of v_a with v_c. Because the original graph is 3-colorable, the filter can be reduced to one with only six vertices.

exists a reduced filter $\mathbf{F_3}$ with at most 6 vertices, such that $\mathbf{F_2} \xLeftrightarrow{\mathcal{L}(\mathbf{F_2})} \mathbf{F_3}$. Let us consider each direction of this proposition in turn.

Lemma 2: For any instance $\mathbf{G_1}$ of GRAPH-3C for which the correct output is "True," then the correct output of the filter minimization problem instance $\mathbf{F_2}$ described above is also "True."

Proof: We must show that if $\mathbf{G_1}$ is 3-colorable, then $\mathbf{F_2}$ can be reduced to an equivalent filter of at most 6 vertices. Let $c_1 : V_1 \to \mathbb{N}^+$ denote a 3-coloring of $\mathbf{G_1}$. To construct filter $\mathbf{F_3}$ with the required properties, start from $\mathbf{F_2}$ as described above, and perform vertex identification operations[1] on all pairs of vertices v_a and v_b for which (i) both are generated in step 2 above, and (ii) $c_1(v_a) = c_1(v_b)$. Note that the resulting graph has at most 6 vertices: v_0, v_+, v_-, and at most three vertices associated with the three distinct colors in c_1. Figure 4 illustrates this construction for the example introduced in Figure 3.

To show that $\mathbf{F_3}$ is a legitimate filter, we must confirm that none of its new vertices have more than one outgoing edge for any observation. Suppose such a vertex v exists with two distinct outgoing edges for observation y_{ab}. Then this vertex must also have incoming edges from v_0 for observations y_a and y_b. The resulting situation is depicted below.

---

[1] A vertex identification operation modifies a graph by replacing multiple vertices into single new vertex, redirecting the incoming and outgoing edges of the original vertices to the new vertex.
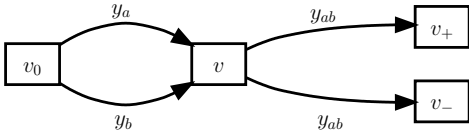
Because observations y_a and y_b both lead to v, we know that $c_1(v_a) = c_1(v_b)$. However, the existence of edges labeled with observation y_{ab} implies that an edge exists in E_1 between v_a and v_b. Since v_a and v_b are connected by an edge but have the same color, we have a contradiction to the supposition that c_1 is a proper 3-coloring of $\mathbf{G_1}$. This contradiction implies that $\mathbf{F_3}$ is a legitimate filter. Finally, it is straightforward to see that $\mathbf{F_3}$ is equivalent to $\mathbf{F_2}$ by examining each of the finitely many observation strings in $\mathcal{L}(\mathbf{F_3})$. ∎

Lemma 3: For any instance $\mathbf{G_1}$ of GRAPH-3C *for which the correct output is "False," then the correct output for the* FM-DEC *instance $\mathbf{F_2}$ described above is also "False."*

Proof: By contrapositive. Suppose there exists a six-vertex filter $\mathbf{F_3} \triangleq (V_3, E_3, m, v_0', d)$ that is equivalent to $\mathbf{F_2}$. We must show that there exists a 3-coloring of $\mathbf{G_1}$.

First, note that the start vertex of $\mathbf{F_3}$ must have color 1 (*i.e.,* $d(v_0') = 1$), since $\mathbf{F_2}$ generates color 1 on an empty observation string. Note also that $\mathbf{F_3}$ must also have one vertex of color 3 and one vertex of color 4 that are reached by observation sequences of length 2. Therefore the other three vertices are reached by observation sequences of length 1. Denote these three vertices $v_1, v_2,$ and v_3.

For each vertex v_a in V_1, note that an edge $v_0 \xrightarrow{y_a} v_j$, must exist in $\mathbf{F_3}$ since the filter is equivalent to $\mathbf{F_2}$. Let c_1 denote the vertex-labeling of $\mathbf{G_1}$ constructed by assigning $c_1(v_i) = i$. Since $v_1, v_2,$ and v_3 are the only candidates for v_j, this labeling uses only three colors.

We still must argue that this labeling is a proper coloring of $\mathbf{G_1}$. Suppose not, and let $(v_a, v_b) \in E_1$ denote an edge for which $c_1(v_a) = c_1(v_b)$. By construction, $\mathbf{F_2}$ has edges $v_0 \xrightarrow{y_a} v_a$ and $v_0 \xrightarrow{y_b} v_b$. Therefore, the observation sequences $y_a y_{ab}$ and $y_b y_{ab}$ generate the same color in $\mathbf{F_3}$, where the transition goes to $v_{c_1(v_a)} = v_{c_1(v_b)}$ on receiving either y_a or y_b. However, the construction of $\mathbf{F_2}$ dictates that these two observation sequences generate different colors, namely a 3 and a 4. This contradiction implies that the labeling c_1 is indeed a 3-coloring of $\mathbf{G_1}$, completing the proof. ∎

Partial results can be assembled thusly:

Lemma 4: FM-DEC *is NP-hard.*

Proof: Combine Lemmas 2 and 3. ∎

Theorem 1: FM-DEC *is NP-complete.*

Proof: Combine Lemmas 1 and 4. ∎

Theorem 2: FM *is NP-hard.*

Proof: This is a direct consequence of Lemma 4. ∎

3) Relationship to DFA minimization: Notice that FM-DEC has some surface-level similarity to the problem of minimizing a deterministic finite automaton (DFA):
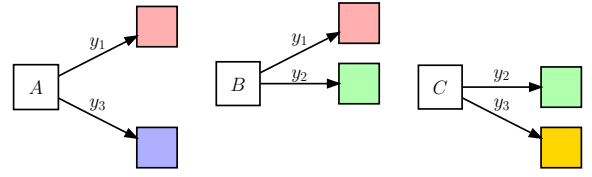

Fig. 5. A graph fragment illustrating a difficult case for filter reduction. Node B can merge with either of node A or node C, but not both. For DFA reduction, this case cannot occur because every node has an outgoing edge for each symbol in the input alphabet.

Decision Problem: DFA Minimization (DFA-DEC**)**

Input: A DFA $\mathbf{M}$.

Output: True if there exists a DFA $\mathbf{M}'$ with at most k states, such that $\mathcal{L}(\mathbf{M}) = \mathcal{L}(\mathbf{M}')$; *False* otherwise.

In both cases, the input is a graph that describes transitions that occur in response to a finite alphabet of input symbols, and the goal is to determine whether the input graph can be reduced to a given size. However, DFA-DEC is efficiently solvable using a straightforward partition refinement algorithm [15].

This apparent discrepancy is explained by the fact that, in contrast to DFA-DEC, we do not require the reduced filter to produce identical results for every observation string in $Y^\star$, but only on those observation strings in $\mathcal{L}(\mathbf{F})$. In practice, this means that the reduced filter may generate colors for observations strings that are not in the language induced by the original graph, which allows I-states to be "merged" even when their outgoing edges differ. Figure 5 shows a small example. Perhaps somewhat surprisingly, the need to perform these merges in a globally optimal way leads to the hardness result expressed in Theorem 2. This result is somewhat analogous to the known hardness of nondeterministic finite automaton (NFA) reduction [17], but is distinct due to the differing notions of equivalence between NFAs and filters.

C. An efficient heuristic algorithm

In the previous section, we showed that, under widely-accepted complexity assumptions, neither FM nor FM-DEC can be solved by any polynomial-time algorithm. In this section, we present an efficient algorithm whose input is a filter $\mathbf{F_1}$, and whose output is another filter $\mathbf{F_2}$, for which $\mathbf{F_1} \xhookrightarrow{\mathcal{L}(\mathbf{F_1})} \mathbf{F_2}$ and $|V_1| \leq |V_2|$. However, in contrast to the FM problem discussed in Section III-B, we do not require $\mathbf{F_2}$ to be the smallest filter with this property.

The idea behind the algorithm is to imagine merging each group of same-colored vertices in $\mathbf{F_1}$ into a single vertex in $\mathbf{F_2}$. If, for each color that appears in $\mathbf{F_1}$, all of the outgoing edges for each observation go to vertices of the same color, then this operation forms a well-defined filter—there is no ambiguity about the correct destination in $\mathbf{F_2}$ for each edge in $\mathbf{F_1}$. In contrast, if any color that appears in $\mathbf{F_1}$ has two outgoing edges labeled with the same observation but with different destination colors, then it is not clear which edges should be included in the new filter. Our algorithm works by iteratively refining the coloring of $\mathbf{F_1}$ until all of these conflicts are eliminated, after which it merges all of the same-colored vertices to form $\mathbf{F_2}$.

Algorithm 2 Heuristic Filter Minimization

Input:

A filter $\mathbf{F_1} \triangleq (V, E, l : E \to Y, v_0, c_1)$.

Output:

A filter $\mathbf{F_2}$, such that $\mathbf{F_1} \xmapsto{\mathcal{L}(\mathbf{F_1})} \mathbf{F_2}$.

1: **while** $\mathbf{F_1}$ has a conflicted color k **do**
2:     Compute the conflict graph for color k in $\mathbf{F_1}$.
3:     Color the conflict graph using an efficient (but sub-optimal) graph coloring algorithm.
4:     Refine the coloring of $\mathbf{F_1}$, replacing k with this coloring.
5: **end while**
6: Form $\mathbf{F_2}$ by performing vertex identifications on any pair of same-colored vertices in $\mathbf{F_1}$.
7: Color each vertex of $\mathbf{F_2}$ using the unique original color of its constituent vertices.
8: **return** $\mathbf{F_2}$

More formally, we use the notion of *conflict* between the two vertices:

Definition 5: *In a filter* $\mathbf{F} \triangleq (V, E, l : E \to Y, v_0, c)$, *two vertices* $v \in V$, $w \in V$ *are in* conflict *if* $c(v) = c(w)$ *and there exists an observation* y *and edges* $v \xrightarrow{y} v'$ *and* $w \xrightarrow{y} w'$ *such that* $c(v') \neq c(w')$. *A color* k *is called* conflicted *if at least one pair of vertices assigned to that color are in conflict.*

Note that there is some similarity between the idea of a conflict between in a filter and the idea of distinguishable states in a DFA. In the context of filters, however, the conflict relation does not form an equivalence relation, in contrast to the Myhill-Nerode equivalence relation that arises for regular languages [16].

Definition 6: *In a filter, the* conflict graph for color k *is an undirected graph with the vertex set* $\{v \in V \mid c(v) = k\}$ *and edge set* $\{(v, w) \in E \mid v \text{ conflicts with } w\}$.

The key observation is that, for any conflicted color k, if we find a coloring of its conflict graph (using new, unique colors that are not in the image of c) and modify the original c to use those new colors in replacement of k, then none of the new colors that replace k will be in conflict with any other vertices.

Our algorithm (see Algorithm 2) uses a series of these conflict graph colorings to refine the coloring of $\mathbf{F_1}$ until it has no conflicts. We intentionally leave the algorithm for coloring the conflict graph as an unspecified "black box." Section III-C2 discusses a few options for how one might instantiate this black box, and the experiments in Section III-D evaluate their performance. When there are no remaining conflicts, the final filter is formed by merging each subset of I-states that share the same color.

1) Correctness and analysis: The next two lemmas confirm that Algorithm 2 terminates and returns a filter equivalent to its input.

Lemma 5: Algorithm 2 terminates after at most $|V|^2$ *iterations of its loop.*

Proof: Let $n(\mathbf{F}) \triangleq \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} [c(v_i) = c(v_j)]$, in which $[\cdot]$ denotes the indicator function whose value is 1 when its argument is true and 0 when its argument is false. That is, $n(\mathbf{F})$ denotes the number of same-colored vertex pairs in $\mathbf{F}$. Observe that $n(\mathbf{F_1})$ decreases by at least 1 with each iteration of the loop in Algorithm 2. Moreover, if $n(\mathbf{F_1}) = 0$, then every vertex in $\mathbf{F_1}$ has a distinct color, so by definition there are no conflicts. Therefore, the algorithm terminates after at most $|V|^2$ iterations. ∎

Lemma 6: Algorithm 2 correctly produces a filter equivalent to $\mathbf{F_1}$.

Proof: Each edge in $\mathbf{F_1}$ corresponds to an edge in $\mathbf{F_2}$ with the same source color, destination color, and observation label. As a result, every observation sequence in $\mathcal{L}(\mathbf{F_1})$ generates the same color in both $\mathbf{F_1}$ and $\mathbf{F_2}$, which implies that $\mathbf{F_1} \xmapsto{\mathcal{L}(\mathbf{F_1})} \mathbf{F_2}$. Therefore, Algorithm 2 is correct. ∎

To bound the runtime of the algorithm, let $f(n)$ denote an upper bound on the time used to color a conflict graph of size n, which depends on the graph coloring technique we select. To compute the conflict graph requires $|Y|$ time to check for each conflict, and $|V|^2|Y|$ to build the entire graph. To apply the conflict graph's coloring back to $\mathbf{F_1}$ is a trivial $|V|$-time operation. Finally, forming and coloring $\mathbf{F_2}$ is also straightforward $|V||Y|$-time computation. Hence, Algorithm 2 runs in time $O(|V|^4|Y|f(|V|))$. However, this a pessimistic bound: In practice, the algorithm generally uses far fewer than $|V|^2$ iterations of its outer loop.

2) Conflict graph coloring: So far we have not specified any technique to use for coloring the conflict graphs in Algorithm 2. First, note that all known algorithms for performing this coloring optimally—that is, using the fewest colors possible—take time exponential in the number of vertices [5]. Therefore, Algorithm 2 can only be efficient if the subroutine we use to color the conflict graphs might produce a sub-optimal coloring.

A large family of sub-optimal graph coloring algorithms have been proposed [4, 32], any of which would be suitable for our approach. Our implementation uses *sequential greedy coloring* [6] because of its simplicity, ease of implementation, and solution quality. The intuition of this approach is to select some order for the vertices and to assign colors to them in that order, using the first available color for each. The quality of the solutions generated by this approach depends strongly on how the vertices are ordered. We considered several options:

- A *natural ordering*, in which we make no special attempt to order the vertices, and allow them to retain whatever arbitrary ordering is determined by the details of the implementation.
- *Ordering by degree*, in which the vertices are ordered by their degree in the conflict graph, starting with the highest degree vertex.
- *Random ordering*, in which we use a pseudo-random number generator to shuffle the vertices, so that all permutations are equally likely.
- *Iterated random ordering*, in which we repeat the coloring several times using different random permutations, and retain only the best result.
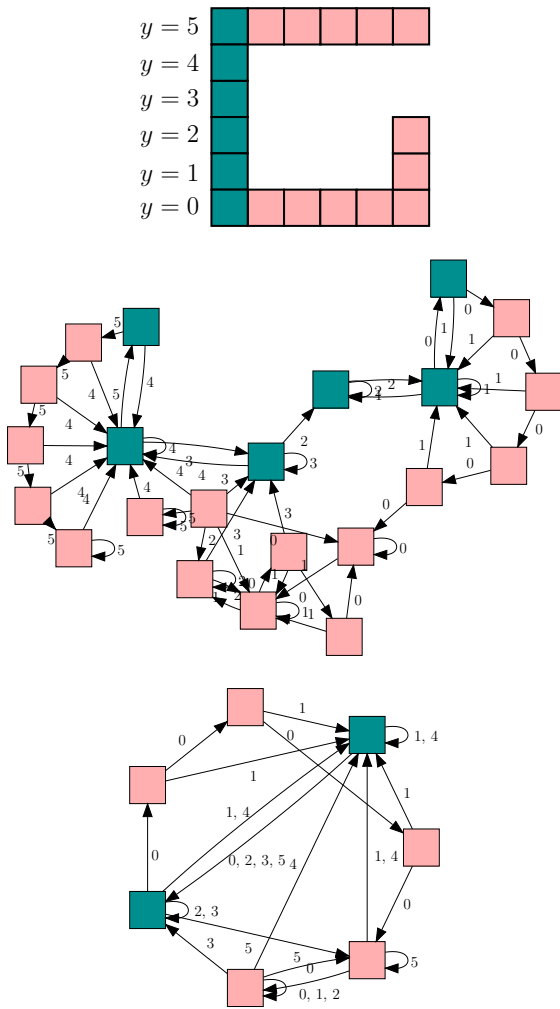
Fig. 6. [top] A grid environment for a robot that can detect its current row. [middle] An unreduced filter for the task of tracking whether the robot is in the leftmost row. [bottom] The reduced filter produced by our algorithm.

For comparison purposes, we also implemented an optimal coloring algorithm that works by exhaustively enumerating partitions of the vertices.

D. Experimental results

We have implemented Algorithms 1 and 2 in C++. This section presents results showing their effectiveness on several example problems. All of the executions described in this section were performed on a GNU/Linux computer, using a single core of a quad-core 2.5GHz processor. We terminated each run as a failure after 10 minutes of CPU time.

1) Grid example: Figure 6 shows an example, in which a robot moves in a grid one step at a time in a grid of 18 cells. At each step, the robot receives observations indicating its current row. The filtering task is, starting from a known starting state, to output 0 if the robot is in the leftmost column and a 1 otherwise. Our algorithm reduced the naïve version of this filter, which has 18 states, down to an equivalent, optimally reduced, filter of only 7 states. Our implementation took approximately 15ms to solve this problem.
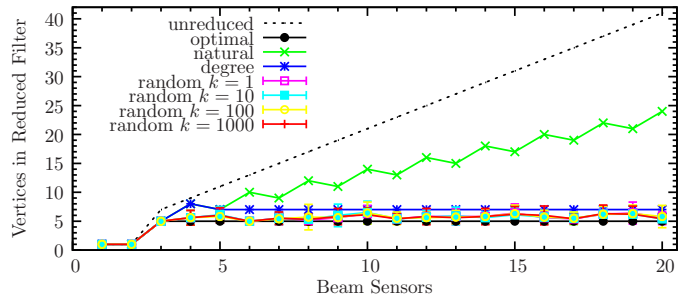

Fig. 7. Solution quality for automatically reduced filters for a single agent moving in an annulus amongst beam sensors. For randomized algorithms, the error bars show the standard deviation after 10 trials.

For comparison, we also implemented an alternative reduction algorithm that treats the input filter as a DFA—replacing each "missing" edge with a transition to a special "dead state" with its own unique color—and applies a standard table-filling minimization algorithm to the resulting DFA. This approach produces a correct filter. However, for this problem, resulting filter is identical to the input—every state is distinguishable from each other, because of the differences in observations that might be obtained.

2) Quantitative evaluation: We considered two generalizations of the annulus world problem shown in Figure 1, in which we varied both the number of agents and the number of beam sensors subdividing the environment. First, we formed a family of problems in which one agent moves in the annulus amidst varying numbers of beam sensors and the goal is to recognize when the agent is in a given target region ("region 0"). This problem is noteworthy because the optimal filter has a constant size of five vertices, regardless of the number of regions.

We constructed input filters based on nondeterministic I-states for all instances with between 1 and 20 regions, and executed Algorithm 2 to reduce those filters. For conflict graph coloring, we used an exponential-time optimal algorithm, along with sequential greedy coloring using (i) the implementation's natural ordering, (ii) ordering by degree, and (iii) random ordering with the number of random colorings k of each conflict graph set to $k = 1$, 10, 100, and 1000. For the randomized algorithms, we performed 10 trials and computed the mean and standard deviation of the reduced filter size. Figure 7 shows the results of this experiment. Notice that, except for the naïve natural ordering, all of the approximate coloring algorithms achieved results at or near the globally optimal solution with 5 vertices. All of these runs completed within the allotted time.

Figure 8 shows two of the reduced filters for the annulus problem. Note that the equivalence of these two to one another illustrates the utility of Algorithm 1 as it is challenging to assess their equivalence visually. We speculate that this difficulty stems from the fact that the language over which they are equivalent is a subset of $Y^\star$, which is only implicit here. Notice how the structure of the five region, single agent annulus implies that, for example, no observation sequence contains the subsequence "ad."

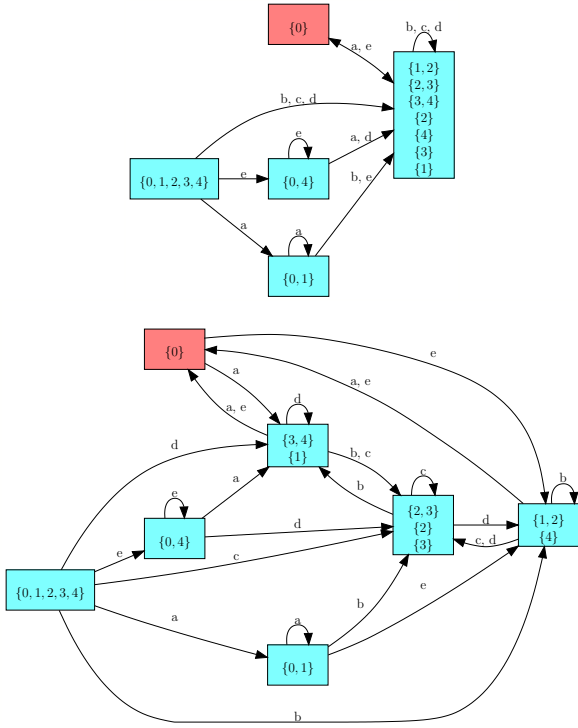Second, we considered a variation of the problem from

Fig. 8. [top] A 5-vertex filter for the 5-sector 1-agent annulus problem, generated by Algorithm 2, using optimal colorings for the conflict graphs. [bottom] An equivalent filter using seven vertices for the same problem, generated using the best of 10 colorings of each conflict graph, each based on sequential coloring with a random ordering of the vertices.
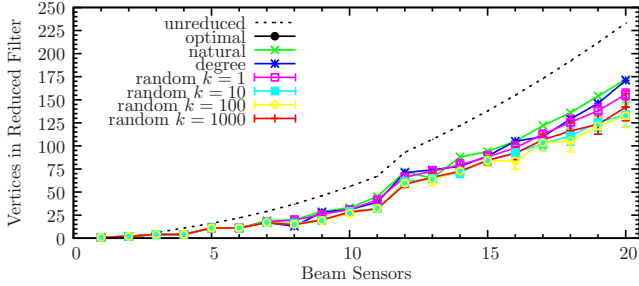

Fig. 9. Solution quality for automatically reduced filters for a two agents moving in an annulus amongst beam sensors. For randomized algorithms, the error bars show the standard deviation after 10 trials.

Figure 1 in which there are two agents, the number of beam sensors varies between 1 and 20, and the filter's goal is to know when the agents are in the same region without regard for which of region they share. Figure 9 shows results of this experiment, using the same conditions as described above.

Third, we considered the L-shaped corridor problem introduced in Section 11.3.1 of LaValle's book [20]. This scenario features a single robot moving in an L-shaped grid using actions up, down, left, and right. Each of these actions moves the robot in the requested direction by either one or two steps, but stops prematurely should the robot reach the environment boundary before completing its motion. The robot has no sensors. This problem is noteworthy because, whereas the number of I-states in the unreduced filter increases exponentially as a function of the length of the corridor, the
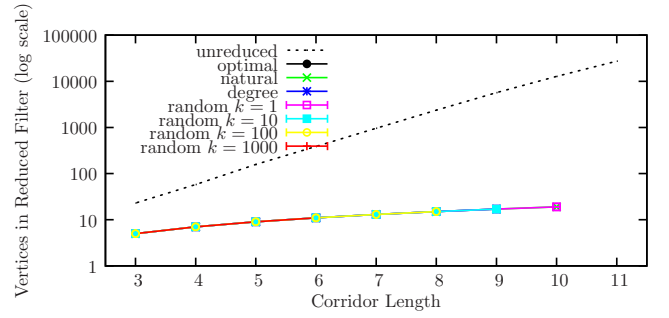

Fig. 10. Solution quality for automatically reduced L-corridor filters. For randomized algorithms, the error bars show the variance after 10 trials. Note the logarithmic scale on the vertical axis. The optimal curve is not visible because it coincides with the experimental data.

size of the reduced filters increases only linearly. Figure 10 shows the results of this experiment. In this case, our algorithm was able to generate the optimal reduced filter in every run for which it finished within the allotted time.

IV. CONCISE PLANS

Next, we move beyond passive filtering to consider planning problems, in which an agent actively selects actions, rather than being merely a passive observer. This section follows the same pattern as Section III, starting with a problem statement (Section IV-A) and proceeding to a hardness result (Section IV-B) and a heuristic algorithm (Section IV-C) before concluding with an experimental evaluation of that algorithm (Section IV-D).

A. Definitions

A robot interacts with its environment by executing *actions*, selected from a finite *action space* U. We assume that the action space contains a special *termination action* u_T, which signals that the robot has completed its execution. In response to each action, the robot receives an *observation*, as before, selected from a finite *observation space* Y, from its sensors.

To model planning problems properly, we must extend Definition 1 to active information state graphs that account for both actions and observations.

Definition 7: *An active I-state graph* $\mathbf{I} \triangleq (V_u \cup V_y, E_u \cup E_y, u : E_u \to U, y : E_y \to Y, v_0)$ *is an edge-labeled bipartite directed multigraph, with a start vertex, in which*

1) *the finite vertex set* $V_u \cup V_y$, *of which each member is called an I-state, can be partitioned into a set of* action vertices V_u *and a set of* observation vertices V_y,
2) *the edge set can be partitioned into a set of* action edges $E_u \subseteq V_u \times V_y$ *and a set of* observation edges $E_y \subseteq V_y \times V_u$,
3) *each action edge* e *is labeled with an action* $u(e)$,
4) *each observation edge* e *is labeled with an observation* $y(e)$, *and*
5) *the starting vertex* $v_0 \in V_u$ *is an action node.*

No pair of distinct edges (neither action edges nor observation edges) may share both a source vertex and a label.

Given an active I-state graph, we can trace the evolution of the robot's I-state by following the appropriately labeled

edge each time the robot executes an action or receives an observation. As in the filtering case, this formulation does not require that each action vertex have an out-edge for each action in the action space; those missing actions are considered "illegal" from those I-states. Likewise, an observation node need not have out-edges for each observation in the observation space, which—just as in Section III—can occur if the underlying structure of the problem dictates that certain observations cannot occur from a given I-state.

Note that, because we are interested in plans that succeed even in the worst case, we do not attach any probability models to the observations; any observation for which an observation edge exists is considered possible, and all such observations are treated equally by our algorithms. We discuss the potential for probabilistic extensions in Section V.

For a given active I-state graph, we can define a family of distinct planning problems by varying the initial and goal states.

Definition 8: *A* planning problem *is a 2-tuple* $\mathcal{P} = (\mathbf{I}, V_g)$, *in which* $\mathbf{I} \triangleq (V_u \cup V_y, E_u \cup E_y, v_0)$ *is an active I-state graph and* $V_g \subseteq V_u$ *is called the* goal region.

The objective is to generate a strategy that, when executed starting from v_0, will terminate at some I-state in V_g regardless of the observations received along the way. Such strategies, which operate in discrete time and with finite memory, are naturally expressed as transition graphs.

Definition 9: *A* plan graph $\mathbf{P} \triangleq (V_p, E_p, u : V_p \to U, y : E_p \to Y)$ *is a directed graph along with two labellings in which*

1) *one vertex* $v_0 \in V_p$ *is designated as a* start plan vertex.
2) *each vertex* $v \in V_p$ *is labeled with an action* $u(v) \in U$,
3) *each edge* $e \in E_p$ *is labeled with an observation* $y(e) \in Y$, *and*
4) *no pair of distinct edges share both a source vertex and a label.*

To execute the plan described by such a graph the robot should, starting from v_0, execute the action $u(v_0)$, and then follow the plan graph edge corresponding to the observation received. This process repeats until one of two stopping conditions is met:

1) The plan attempts to execute an action that is not allowed at the robot's current I-state (indicated by the absence of the corresponding edge in the I-state graph), or the plan lacks an edge labeled with the robot's observation outgoing from its current vertex. In either case, the result of the plan for that execution is a *failure*.
2) The plan executes action the termination action u_T. In this case, the plan's execution is a *success* if the current I-state is a member of the goal region, or a *failure* otherwise.

Notice that, given a plan graph that successfully terminates at the goal, the robot does not need to store the original I-state graph; the plan graph alone contains sufficient information for the robot's execution.

We are interested in plan graphs that succeed in the worst case for a given planning problem.

Definition 10: *A plan graph* $\mathbf{P}$ *solves a planning problem* $\mathcal{P} = (\mathbf{I}, V_g)$ *if there exists an integer* k *such that every execution of* $\mathbf{P}$ *successfully terminates in* V_g *after at most* k *steps.*

Finally, notice that the size of a plan graph is a direct indicator of the plan's conciseness. This motivates the core problem addressed in this section:

Problem: Concise Planning (CP)

Input: A planning problem $\mathcal{P}$.
Output: A plan graph $\mathbf{P}$ that solves $\mathcal{P}$, such that the number of vertices in $\mathbf{P}$ is minimal.

Through the rest of this section, we consider algorithms for this problem.

B. Hardness of concise planning

In this section, we prove that the concise planning problem introduced in Section IV-A is NP-hard. In keeping with the usual practice in complexity theory, our approach starts from the related decision problem:

Decision Problem: Concise Planning (CP-DEC)

Input: A planning problem $\mathcal{P}$ and an integer k.
Output: *True* if there exists a plan graph $\mathbf{P}$ of at most k vertices that solves the $\mathcal{P}$; *False* otherwise.

We show that CP-DEC is NP-complete, which directly implies that CP is NP-hard. To accomplish this, we first show that CP-DEC is in class NP (Lemma 7), and then show, via reduction from a graph coloring problem, that CP-DEC is NP-hard. Though the general approach of these proofs follows the pattern established for filter minimization problems in Section III-B, there are enough meaningful differences in the precise details to justify presenting them separately. In particular, the requirement that a plan terminate prohibits the active case from directly inheriting hardness properties from the passive version.

Lemma 7: CP-DEC *is in complexity class NP.*

Proof: To show that CP-DEC is in NP, it is sufficient to find a polynomial-time algorithm that determines, given a planning problem $\mathcal{P}$, an integer k, and a plan graph $\mathbf{P}$, whether (i) $\mathbf{P}$ has at most k nodes and (ii) $\mathbf{P}$ solves the planning problem. The former condition requires a simple count of the vertices. A technique to check the latter condition appears as Algorithm 3.

(The intuition behind the algorithm is to enumerate all reachable I-state/plan node pairs via a forward search, and to return *True* only if the set of reachable pairs is exhausted without finding any failures, incorrect terminations, or potential infinite cycles.)

It is straightforward to see that, for each iteration of the outer while loop, the algorithm does work bounded by the number of observations. The outer while loop can perform no more than one iteration per unique pair of plan and I-state graph vertices, and therefore the whole algorithm has time-complexity in $O(|Y||V_u||V_p| + |Y||V_y||V_p|)$. Because this

Algorithm 3 Verify Plan Correctness

Input:
>   *A problem $\mathcal{P} = (\mathbf{I}, V_{\mathrm{g}})$ and a plan graph $\mathbf{P}$.*

Output:
>   *True if $\mathbf{P}$ solves $\mathcal{P}$; False otherwise.*

1: $Q \leftarrow$ empty queue
2: $Q.\text{insert}((v_0, v_0(\mathbf{P})))$
3: **while** Q is not empty **do**
4:     $(v_i, v_p) \leftarrow Q.\text{pop}()$
5:     **if** (v_i, v_p) is its own ancestor **then**
6:        **return** False {Plan may never terminate.}
7:     **end if**
8:     **if** $u(v_p) = u_{\mathrm{T}}$ **then**
9:        **if** $v_i \notin V_{\mathrm{g}}$ **then**
10:        **return** False {Plan terminates outside of goal.}
11:        **end if**
12:     **else**
13:        **for** each out edge $v_i \xrightarrow{y} v_i'$ of v_p **do**
14:          **if** $\mathbf{P}$ has an edge $v_p \xrightarrow{y} v_p'$ and (v_i', v_p') has not be inserted into Q yet **then**
15:            $Q.\text{insert}((v_i', v_p'))$
16:          **else**
17:            **return** False
{Plan is not prepared for observation.}
18:          **end if**
19:        **end for**
20:     **end if**
21: **end while**
22: **return** True {No incorrect terminations, failures, or infinite cycles.}
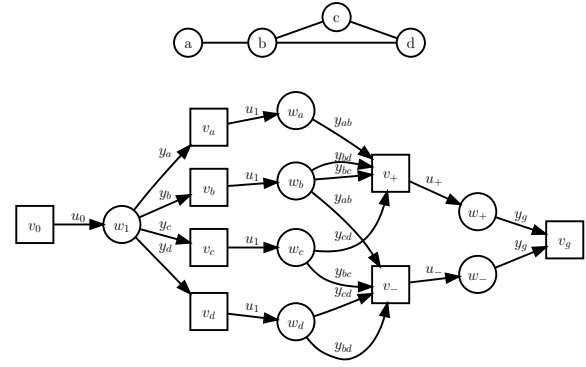

Fig. 11. [top] An example instance of 3-coloring. [bottom] The I-state graph constructed from that instance.
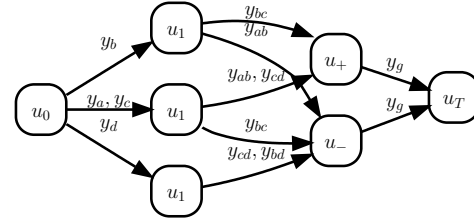

Fig. 12. A plan that solves the planning problem in Figure 11. Because the original graph is 3-colorable, the problem can be solved by plan with only 7 vertices. (Note that the text interior to vertices are actions, labelled via $u(\cdot)$.)

algorithm exists and executes in polynomial time, we have the desired result. ∎

To show that CP-DEC is NP-complete, the proof follows the same general pattern as the corresponding proof for FM-DEC in Section III-B. Again we reduce from GRAPH-3C.

Given an instance of GRAPH-3C, that is, an undirected graph $\mathbf{G} \triangleq (V, E)$, we construct an instance $(\mathbf{I}, V_{\mathrm{g}}), k$ of CP-DEC with the following elements in $\mathbf{I}$:

1) A starting action node v_0.
2) An observation node w_1 and an edge $v_0 \xrightarrow{u_0} w_1$ connecting it to v_0, labeled with an initial action u_0.
3) For each vertex a of $\mathbf{G}$, an action node v_a, and observation node w_a, and edges $w_1 \xrightarrow{y_a} v_a$ and $v_a \xrightarrow{u_1} w_a$.
4) Two action nodes v_+ and v_- and two observation nodes w_+ and w_-, along with action edges $v_+ \xrightarrow{u_+} w_+$ and $v_- \xrightarrow{u_-} w_-$.
5) For each edge (a, b) of $\mathbf{G}$, two observation edges $w_a \xrightarrow{y_{ab}} v_+$ and $w_b \xrightarrow{y_{ab}} v_-$.
6) An action node v_g, and two observation edges $w_+ \xrightarrow{y_g} v_g$ and $w_- \xrightarrow{y_g} v_g$.

We complete the CP-DEC instance by choosing v_0 and $\{v_g\}$ for the start node and goal region respectively, and setting $k = 7$.

Figure 11 shows an example of this construction. The

intuition is that only two action sequences allow the robot to successfully reach the goal, namely $(u_0, u_1, u_+, u_{\mathrm{T}})$ and $(u_0, u_1, u_-, u_{\mathrm{T}})$. Moreover, in any given execution, only one of these two choices will succeed, depending on the observation received after executing u_0. The construction forces any successful plan graph to remember enough about the observations to know whether u_+ or u_- is the correct choice.

The time required to perform this construction is linear in the size of $\mathbf{G}$. We now argue that the constructed planning problem is equivalent to the original graph, in the sense that the graph has 3-coloring if and only if the planning problem admits a solution of at most 7 vertices.

Lemma 8: For any instance $\mathbf{G} = (V, E)$ of GRAPH-3C for which the correct output is "True," the correct output of the CP-DEC instance described above is also "True."

Proof: Let $c : V \to \{1, 2, 3\}$ denote a 3-coloring of $\mathbf{G}$. Let $\mathbf{P}$ denote the plan graph of exactly 7 vertices with the following elements:

1) A start vertex v_0 labeled with action u_0.
2) Three vertices v_1, v_2, and v_3 (one for each of the colors of $\mathbf{G}$), all labeled with action u_1.
3) Three vertices v_+ and v_-, and v_g, labeled with action u_+, u_-, and u_{T} respectively, along with edges $v_+ \xrightarrow{y_g} v_g$ and $v_- \xrightarrow{y_g} v_g$.
4) For each vertex a of $\mathbf{G}$, an edge $v_0 \xrightarrow{y_a} v_{c(a)}$.
5) For each edge (a, b) of $\mathbf{G}$, edges $v_{c(a)} \xrightarrow{y_{ab}} u_+$ and $v_{c(b)} \xrightarrow{y_{ab}} u_-$.

Figure 12 illustrates this construction for the example introduced in Figure 11.

To show that $\mathbf{P}$ is indeed a plan graph, we must confirm that none of its vertices has multiple outgoing edges labeled with the same observation. The only vertices at which this could occur are v_0, v_1, and v_2. Suppose such a vertex v exists, with two distinct outgoing edges for observation y_{ab}. Because v_+ and v_- are the only two possible targets for edges outgoing from v, these two edges must connect those two vertices.

By construction, v must also have incoming edges from v_0 for observations y_a and y_b. The resulting situation is depicted below (labels inside the vertices gives the associated action).
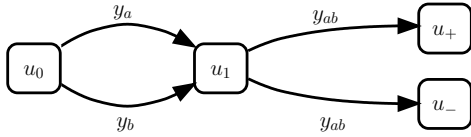
Note that because observations y_a and y_b both lead to v, we know that in the coloring of $\mathbf{G}$, we have $c(v_a) = c(v_b)$. However, the existence of edges labeled with observation y_{ab} implies that $\mathbf{G}$ has an edge between v_a and v_b. Since v_a and v_b are adjacent in $\mathbf{G}$ but have the same color in c, we have a contradiction to the supposition that c is a proper 3-coloring of $\mathbf{G_1}$. Therefore $\mathbf{P}$ is a legitimate plan graph.

Finally, it is straightforward to see that $\mathbf{P}$ correctly solves the planning problem by examining each of the finitely many possible execution traces. ∎

Lemma 9: For any instance $\mathbf{G}$ of GRAPH-3C *for which the correct output is "False," the correct output of the* CP-DEC *instance described above is also "False."*

Proof: Prove by contrapositive. Suppose there exists a seven-node plan graph $\mathbf{P}$ that solves this planning problem, in order to show that there exists a 3-coloring of the original $\mathbf{G}$.

First, note that any correct plan for this problem must contain at least one distinct node labeled with each of u_0, u_+, u_-, and u_T. Moreover, because each of these actions is executed at most once in any correct plan, we can assume, without loss of generality, that each of these actions is the label for *exactly* one vertex in $\mathbf{P}$. Therefore, there are at most three vertices of $\mathbf{P}$ labeled with u_1. Denote these vertices v_1, v_2, and v_3. Let v_0 denote the $\mathbf{P}$ vertex associated with u_0, which must be the start vertex of $\mathbf{P}$.

For each vertex a in $\mathbf{G}$, note that there must exist in $\mathbf{P}$ a unique edge $v_0 \xrightarrow{y_a} v_j$ to some v_j, where vertex v_j is labeled with u_1. Let $c : V \to \{1, 2, 3\}$ denote the vertex-labeling of $\mathbf{G}$ that maps each vertex a to the index of the target vertex of this associated edge. Since v_1, v_2, and v_3 are the only candidates for v_j, this labeling uses only three colors.

Let us prove by contradiction that c is a proper coloring of $\mathbf{G}$. Suppose not, and let (a, b) denote an edge of $\mathbf{G}$ with $c(a) = c(b)$. By construction, $\mathbf{P}$ has edges $v_0 \xrightarrow{y_{ab}} v_{c(a)}$ and $v_0 \xrightarrow{y_{ab}} v_{c(b)}$. The target vertices of these two edges are identical. However, notice that in a correct plan, the observation sequence $y_a y_{ab}$ must lead to the plan node labeled u_+, while the observation sequence $y_b y_{ab}$ must lead to the plan node labeled u_-. In $\mathbf{P}$ these observation sequences lead to same plan node and, thus, $\mathbf{P}$ is cannot be a correct solution to the planning problem. This contradiction demonstrates that c is a proper 3-coloring of $\mathbf{G}$. ∎

Algorithm 4 TRYSUBPLAN
Input:
    A plan graph $\mathbf{P}$ *and an action node* v.
1: Compute metadata for $\mathbf{P}$.
2: **for** $i \in \{1, 2\}$ **do**
3:    $s_i(v).\text{insert}(\mathbf{P})$
4:    **if** $s_i(v)$ holds more than k_i plans **then**
5:       **remove** worst plan, according to H_i, from $s_i(v)$
6:    **end if**
7: **end for**
8: **if** $\mathbf{P}$ remains in any $s_i(v)$ **and** each out-neighbor of v holds at least one plan **then**
9:    $Q.\text{push}(v)$
10: **end if**

Algorithm 5 PLANCONCISELY
Input:
    A problem $\mathcal{P} = (\mathbf{I}, V_g)$.
Output:
    A plan graph $\mathbf{P}$ *that solves* $\mathcal{P}$.
1: $Q \leftarrow$ empty set of observation nodes
2: $\mathbf{P}_T \leftarrow$ single vertex labeled u_T
3: **for** each action node $v \in V_g$ **do**
4:    TRYSUBPLAN$(v, \mathbf{P}_T)$
5: **end for**
6: **while** Q is not empty **do**
7:    $w \leftarrow Q.\text{pop}()$
8:    **for** each in-neighbor $v \in V_u$ of w **do**
9:       **build** candidate plans starting at v through w.
10:       **for** each candidate plan $\mathbf{P}$ and each $v' \in V_u$ **do**
11:          TRYSUBPLAN$(v', \mathbf{P})$
12:       **end for**
13:    **end for**
14: **end while**
15: **return** smallest reduced plan stored at v_0, if any.

The partial results presented above lead directly to our main hardness results.

Lemma 10: CP-DEC *is NP-hard.*

Proof: Combine Lemmas 8 and 9. ∎

Theorem 3: CP-DEC *is NP-complete.*

Proof: Combine Lemmas 7 and 10. ∎

Theorem 4: CP *is NP-hard.*

Proof: This is a direct consequence of Lemma 10. ∎

C. A heuristic algorithm for concise planning

The previous section proved that, unless $P = NP$, no efficient algorithm can optimally solve the concise planning problem CP. Therefore, we turn our attention now to a planning algorithm that generates plans that are correct, in the sense of reaching the goal even in the worst case, but cannot be guaranteed to be optimally concise.

The idea of the algorithm is to use the structure of the I-state graph to generate a series of candidate plans, each of which
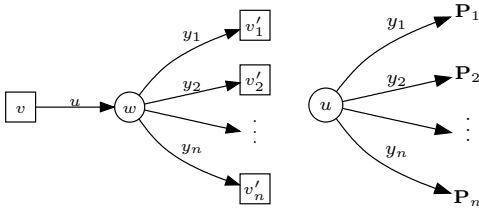
Fig. 13. [left] A fragment of an I-state graph, for which a new plan can be constructed, as long as all of $v_1, v_2, \ldots, v_n$ have at least one associated plan. [right] The constructed plan copies and grafts the existing plans into a tree rooted at a new node with action u.

can successfully reach the goal from at least one I-state. This process starts with a trivial "Terminate immediately" plan, which is correct from within the goal region. From there, the algorithm maintains a queue of observation nodes for which all of the out-neighbors have at least one associated plan, and repeatedly constructs new plans that pass though each successive observation node extracted from that set. The plans generated in this way, all of which have the form of a rooted tree with leaves labeled u_T, each undergo a plan reduction step, which mutates it a given plan into a smaller plan (ideally, optimally concise or nearly so) equivalent to the original.

As the algorithm proceeds, it maintains, for each action node, a small finite collection of sub-plans that reach the goal from that node. Our algorithm prioritizes the plans to retain based on heuristic evaluations of both the local conciseness and global applicability of each sub-plan.

The algorithm terminates when its queue is exhausted, at which time it returns the most concise plan associated with the start vertex of the I-state graph. Pseudocode for this approach appears as Algorithms 4 and 5. The following subsections explain and clarify the details.

1) Candidate plan construction: Beyond the initial trivial plan, Algorithm 5 constructs additional plans in the following way. It identifies action nodes v for which (i) there exists an action edge $v \xrightarrow{u} w$, and (ii) all of the out-neighbors $v_1', \ldots, v_n'$ of w have at least one associated plan. In this situation, we can form a new plan graph that reaches the goal from v as follows: start with a vertex that executes action u, and attach plan out-edges for each of the out-edges $w \xrightarrow{y} v'$ of w in the I-state graph. Each of these edges connects to a copy of a plan associated with v' which, by construction, reaches the goal from there. See Figure 13.

To efficiently locate portions of the I-state graph at which this construction is possible, the algorithm maintains, for each observation node w, a set of "incomplete" (that is, planless) out-neighbors. It inserts w into the global Q each time a new plan is stored at one of its out neighbors, provided that w's incomplete list is empty. In this way, the algorithm ensures that every plan it generates is complete and correct for the v at which it is generated.

2) Plan evaluation heuristics: As mentioned above, Algorithm 5 maintains a bounded size collection of "promising" plans for each action node. Specifically, at action node v, we store *two* plan sets $s_1(v)$ and $s_2(v)$, each of which holds at most k_1 or k_2 plans respectively, in which the k_i values are

tunable algorithm parameters.

Below, we introduce heuristic functions H_1 (which measures the *local* conciseness of a plan) and H_2 (which measures the *global* reusability of a plan). As the algorithm proceeds, $s_1(v)$ always contains the k_1 or fewer plans that maximize H_1, across all generated plans that reach the goal from v. The set $s_2(v)$ likewise stores the k_2 best plans that succeed form v, according to H_2. The next sections introduce H_1 and H_2.

a) Local heuristic: Reduced plan size: Notice that each of the plans constructed as described in Section IV-C1 will have the form of a rooted tree but that, in most cases, concise plans have cycles. In fact, there is no reason to suspect that these trees will be concise plans. Recall, for example, Figure 2. As a result, as part of the "compute metadata step" in Algorithm 4 (line 1), we use a *plan reduction* algorithm whose input is the original rooted tree plan graph $\mathbf{P}$, and the intended output is the smallest plan graph $r(\mathbf{P})$ that produces identical behavior. This problem is equivalent to the filter reduction problem from Section III, with the trivial difference that in Section III, we refer to vertex labels as abstract "colors" instead of actions. We employ Algorithm 2 to reduce plans.

The size of these reduced plans represents a local, greedy measurement of the usefulness of a plan. Therefore, we define the local heuristic as

$$H_1(\mathbf{P}) = -\operatorname{size}(r(\mathbf{P}))$$

Notice that, at the conclusion of the algorithm, the smallest plan in the set $s_1(v_0)$ represents the most concise start-to-goal plan we have found. As a result, this plan becomes the final output of the algorithm.

b) Global heuristic: Reuse potential: Unfortunately, the local heuristic introduced the previous section is insufficient alone, because it cannot account for the idea of choosing actions at one action node expressly because those plan nodes can be re-used in other portions of the I-state graph. This notion of the reuse of plan fragments motivates our second, global heuristic.

The idea is to compute the *outcome function*

$$\mathcal{O}_{\mathbf{P}} : V_\mathrm{u} \to 2^{V_\mathrm{u}}$$

of a plan $\mathbf{P}$. This function considers the potential results from executing $\mathbf{P}$ starting at each action node v, mapping each to the set of action nodes at which that plan might terminate, or to the empty set if the plan might fail when executed from v. This function can be computed by a forward search of reachable action node/plan node pairs, in a manner very similar to Algorithm 3.

The outcome function shows, from a global perspective, how much potential for reuse a plan possesses. For H_2 we use a straightforward measure of reusability based on the average movement across the I-state graph that a plan can achieve, summed across all starting vertices. Specifically, we define

$$H_2(\mathbf{P}) = \sum_{v \in \{V_\mathrm{u} | \mathcal{O}_{r(\mathbf{P})}(v) \neq \emptyset\}} \left(\frac{1}{\mathcal{O}_{r(\mathbf{P})}(v)} \sum_{v' \in \mathcal{O}_{r(\mathbf{P})}(v)} d(v, v') \right),$$

in which $d(v, v')$ denotes the number of edges in the shortest directed path connecting v to v' in $\mathbf{I}$.

3) Algorithm summary: This completes the overall picture of Algorithm 5. To summarize, it tracks a set of observation nodes through which new complete plans can be constructed. As long as this set is not empty, it removes an arbitrary observation node, w. It then constructs new plans that pass through w starting from each of its in-neighbors, using all combinations of plans stored in both the s_1 and s_2 sets of the resulting action nodes. For each such plan $\mathbf{P}$, we compute the heuristic functions H_1 and H_2, and insert $\mathbf{P}$ into the s_1 and/or s_2 sets at *every* action node for which $\mathbf{P}$ successfully reaches the goal and improves upon the existing plans and Q is updated appropriately. This process continues until Q is exhausted, at which point the best start-to-goal reduced plan is returned.

D. Experimental results

We implemented Algorithm 5 to test its efficiency and the conciseness of the plans it produces for both manipulation (Section IV-D1) and navigation (Section IV-D2) domains. Our implementation uses C++ and all of the executions used a single core of a 2.5GHz quad-core processor.

1) Manipulation: In the spirit of the established techniques for (nearly-)sensorless manipulation [12, 29], we executed the algorithm on a family of problems in which the goal is to orient a polygonal shape using a series of squeezes from a parallel-jaw gripper. Given a description of the convex hull of an object we followed the steps (introduced by Goldberg [11]) for solving such problems: (i) we computed the diameter function for the polygon, (ii) we identified minima in this function, giving the stable orientations that occur after a squeeze operation, and (iii) we computed the so-called squeeze function that maps a pre-squeeze orientation into the post-squeeze orientation. For these problems we considered small sets of actions of the form "rotate gripper by x and squeeze." This is sufficient to construct an I-state graph in which each I-state corresponds to a set of stable orientations consistent with the history of actions and observations.

The left part of Figure 14 gives an example using one of the objects we evaluated: the "fourgon" shape. The figure provides intuition for how the local minima in the plot represent stable orientations after a squeeze operation is performed by the frictionless parallel-jaw gripper. Figure 14 shows the form of the I-state graph generated for the problem of orienting the fourgon using rotations of only $5°$ and $65°$, and squeeze operations; the resulting plan produced by the algorithm is also shown. Note how, although the geometry of the object is simple, determining a concise open-loop plan given those actions remains far from obvious.

Next, we incorporated sensing information. Specifically, we specified a set of binary sensors, each determining whether the distance between jaws of the gripper exceed some threshold or not. Figure 15 extends the preceding example by adding a single diameter threshold sensor with distance 10.5cm. The I-state graph observation edges (in blue) are now labeled with the output from the threshold sensor. The resulting plan exploits this information and is smaller than the sensorless plan (which is consistent in plans for orienting other objects too).
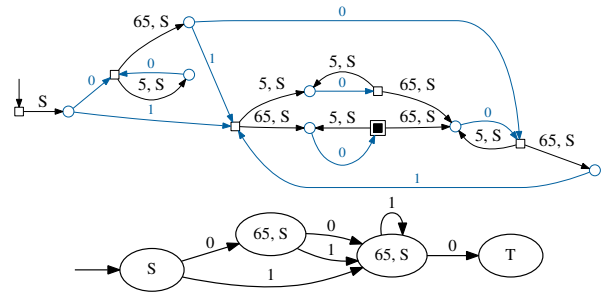

Fig. 15. [top] The I-state graph for the problem of orienting the Fourgon polygon with a binary sensor measuring whether the jaws of the gripper are more than $x = 10.5$cm apart or not. Observation arcs are labeled '0' or '1'; the latter is returned when the diameter of object in its stable orientation exceeds the distance threshold, and '0' is returned otherwise. (The remainder of the graph follows the format of Fig 14.) [bottom] The most concise plan found.
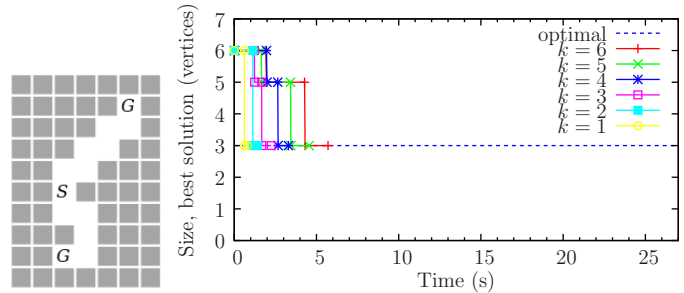

Fig. 16. An example with multiple goals. The same (optimal) solution is found with all values of parameter $k = k_1 = k_2$, which bounds the number of subplans stored at each action node.

We note that additional sensing may make online execution of the plan faster and the most concise plan shorter, but it grows the space of potential plans, increasing the work required for planning offline. As was eloquently pointed out by Goldberg [11], if the task can be solved without sensors, then the addition of sensors gives the ability to determine (through estimation) state sooner, potentially requiring fewer actions and thereby shortening plan execution. Our results suggest that sensors often also lead to plans that are more concise to represent.

2) Navigation: Second, we considered a simplified navigation domain in which a robot moves within a grid of discrete cells using actions up, down, left, and right, each of which reliably moves a single cell in the desired direction unless an obstacle impedes that motion. The robot's observation space is $Y = \{00, 01, 10, 11\}$, in which the first bit indicates whether the robot bumped an obstacle on its
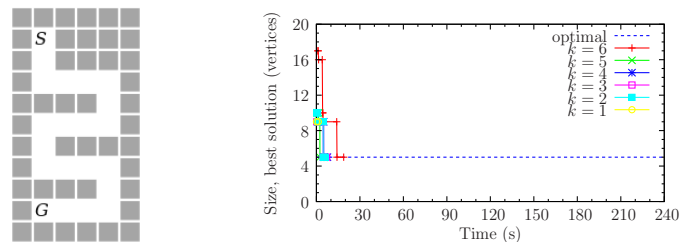

Fig. 17. A switch-back pattern affords opportunity for plan compression.
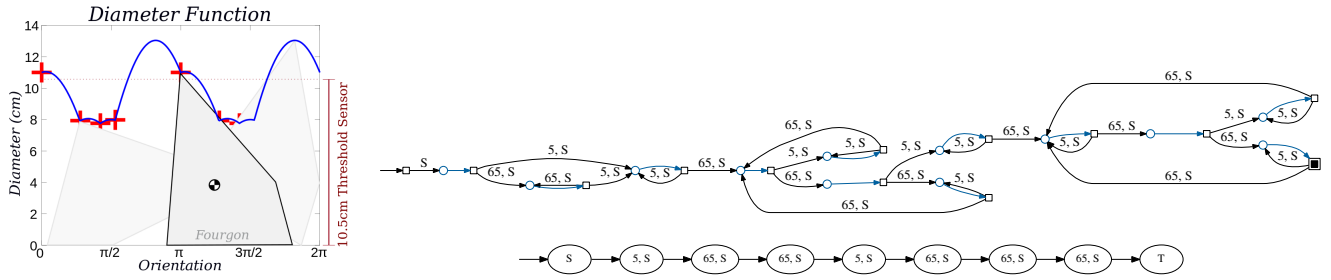
Fig. 14. [left] The diameter function computed for the Fourgon figure (shown inset in three orientations). Local minima in the plot represent stable orientations for the object when squeezed by a parallel-jaw gripper. [right, top] The I-state graph for the problem of orienting the Fourgon polygon without sensors, constructed automatically using classical techniques. The black arcs leading from square vertices are where an action must be selected, the blue arrows represent transitions based on a observation (but are empty in this example); the shaded vertex is the goal. The 'S' symbol denotes a squeeze action by the gripper; '5' and '65' denote rotations of the gripper by those many degrees, respectively. [right, bottom] An open-loop plan found believed to be the most concise solution, found using Algorithm 5.
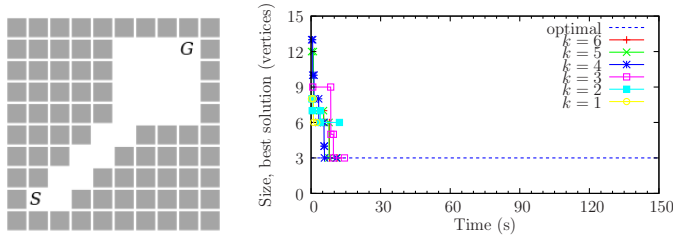

Fig. 18. An example in which many concise plans can navigate through the open field, but only one can navigate the narrow corridor.

		Fig. 16		Fig. 17		Fig. 18	
k_1	k_2	time	size	time	size	time	size
1	1	0.4	3.0	2.0	7.8	2.3	6.5
1	5	3.3	3.0	12.8	5.8	22.5	6.3
5	1	1.4	3.0	5.6	4.0	4.3	3.0
5	5	5.0	3.0	16.0	5.0	13.8	3.0

Fig. 19. Assessing the sensitivity to the parameters k_1 and k_2.

previous move, and the second bit indicates whether the robot has reached its goal or not.

This family of problems is interesting because many instances admit very concise plans. In particular, small plans tend to exist for instances in which short sequences of actions, terminated by appropriate observations, can be repeated to make progress toward the goal. We constructed several grid navigation problems, shown in Figures 16–18, to evaluate Algorithm 5. The figures show the performance of the method with different limits on the number of plans associated with each action node. This is indicated with the value of the k_i parameter used on Line 4 of Algorithm 4. For simplicity, we used the same value, denoted simply k, for both k_1 and k_2 for each execution of our algorithm on this problem

Figure 16 is the same environment as Figure 2, but the I-space differs because the earlier example had no bump detector. We varied this k from 1 to 6. The plot shows that an optimal 3 state solution is found for each value of k. Figure 17 is an environment with the potential for comparatively long sequences to be simplified in a concise plan. This example has a hierarchical structure to the repetitions: there are corridor navigation pieces (repeated `left`/`right` actions until a bump) and repetitive sequences composed of corridor navigation subplans. The shortest solution involves 5 states, and was found for values of k greater than one. Figure 18 is an instance where many conceivable concise subplans can be generated in the open field in the upper right, but only a single plan (the most concise one, with 3 states) is concise and navigates the corridor too. This solution is found only when $k \geq 5$.

The plots of "most concise plan so far" vs. running-time

show the same behavior. A larger k causes slower progress, because the search phase keeps more subplans thereby increasing the search space, but it also increases the likelihood of finding the highest quality solution. (At the extreme, $k = \infty$ is a search of the complete plan space). The lowest value of k which results in the optimal plan gives an informal idea of the comparative complexity of the problems, suggesting that Figure 16 is easiest and Figure 18 most difficult of the three. We hypothesize that this result occurs because of the open field in the top left of Figure 18, through which many distinct plans successfully travel. In a counterintuitive contrast to the narrow corridor problem faced by typical motion planning algorithms, it appears that concise planning problems are more difficult in the presence of wide corridors.

3) Sensitivity to parameters: To measure the sensitivity of the algorithm to its parameters k_1 and k_2, we chose relatively small ($k_i = 1$) and large values ($k_i = 5$) and performed four tests, spanning the four distinct assignments of these values to k_1 and k_2. We solved each of the problems depicted in Figures 16–18 and performed ten trials for each combination of planning problem and parameter values. For each trial, we measured the algorithm's run time and the size of the resulting plan. Figure 19 shows the results.

V. CONCLUSION

This article considered algorithmic problems of conciseness for both filtering and planning tasks. We proved that both optimal minimization of combinatorial filters and generation of optimally concise plans are both NP-hard problems. As always, there remain a number of interesting unanswered questions.

Most directly, though we have shown that the optimal versions of these problems are NP-hard, we leave to future

work the question of whether they can be approximated, in the sense of bounded approximation ratios, efficiently.

In the case of filtering, for some problems, such as the families of filters referred to in Figures 7 and 10, we can determine the size of the optimal reduced filter by manual inspection. We observed, for all such problems, that executing Algorithm 2 with optimal conflict graph coloring did indeed produce a globally optimal filter. An interesting conjecture is to determine whether Algorithm 2 always produces optimal reduced filters when the conflict graphs are colored optimally. More generally, we may be able to place bounds on the quality of solutions produced by Algorithm 2 in terms of the approximation ratio of the underlying conflict graph coloring algorithm.

Finally, all of our results may be extended by including probabilities in the models. For planning, for example, one might relax the requirement of worst case correctness and, after assigning a probability distribution over the out-edges of each observation node, instead form plans whose success probability is less than unity. An interesting question is to understand the impact of increasing the required success probability on plan conciseness.

References

[1] M. A. Bender, A. Fernández, D. Ron, A. Sahai, and S. Vadhan, "The power of a pebble: exploring and mapping directed graphs," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1998.

[2] M. Blum and D. Kozen, "On the power of the compass (or, why mazes are easier to search than graphs)," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1978, pp. 132–142.

[3] B. Bonet and H. Geffner, "An algorithm better than AO*?" in *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI'97)*, 2005, pp. 1343–1347.

[4] D. Brélaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, pp. 251–256, 1979.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[6] D. de Werra, "Heuristics for graph coloring," *Computing*, pp. 191–208, 1990.

[7] M. Erdmann, "On the topology of discrete strategies," *International Journal of Robotics Research*, vol. 29, no. 7, pp. 855–896, 2010.

[8] ——, "On the topology of discrete planning with uncertainty. in advances in applied and computational topology," in *Proc. Symposia in Applied Mathematics*, A. Zomorodian, Ed., vol. 70. American Mathematical Society, 2012.

[9] M. Erdmann and M. T. Mason, "An exploration of sensorless manipulation," *IEEE Transactions on Robotics and Automation*, vol. 4, no. 4, pp. 369–379, Aug. 1988.

[10] M. A. Erdmann, "Understanding action and sensing by designing action-based sensors," *International Journal of Robotics Research*, vol. 14, no. 5, pp. 483–509, 1995.

[11] K. Y. Goldberg, "Orienting polygonal parts without sensors," *Algorithmica*, vol. 10, pp. 201–225, 1993.

[12] K. Goldberg and M. T. Mason, "Bayesian grasping," in *Proc. International Conference on Robotics and Automation*, 1990, pp. 1264–1269.

[13] E. A. Hansen, "Solving POMDPs by searching in policy space," in *Proc. Conference on Uncertainty in Artificial Intelligence*, 1998.

[14] E. A. Hansen and S. Zilberstein, "LAO*: A heuristic search algorithm that finds solutions with loops," *Artificial Intelligence*, vol. 29, no. 1-2, pp. 35–62, 2001.

[15] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Addison-Wesley, 2006.

[16] J. E. Hopcroft, M. R, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Pearson, 2006.

[17] T. Jiang and B. Ravikumar, "Minimal NFA problems are hard," *SIAM Journal on Computing*, vol. 22, no. 6, pp. 1117–1141, 1993.

[18] A. N. Kolmogorov, "On tables of random numbers," *Theoretical Computer Science*, vol. 207, pp. 387–395, 1998, reprinted from Sankhyā: The Indian Journal of Statistics, Series A. Vol. 25, Part 4 (1963).

[19] S. Kristek and D. Shell, "Orienting deformable polygonal parts without sensors," in *Proc. International Conference on Intelligent Robots and Systems*, 2012.

[20] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[21] ——, "Sensing and filtering: A fresh perspective based on preimages and information spaces," *Foundations and Trends in Robotics*, vol. 1, no. 4, pp. 253–372, 2010.

[22] R. Lopez-Padilla, R. Murrieta-Cid, and S. M. LaValle, "Optimal gap navigation for a disc robot," in *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2012.

[23] J. M. O'Kane, "Decentralized tracking of indistinguishable targets using low-resolution sensors," in *Proc. International Conference on Robotics and Automation*, 2011.

[24] J. M. O'Kane and D. A. Shell, "Automatic reduction of combinatorial filters," in *Proc. IEEE International Conference on Robotics and Automation*, 2013.

[25] ——, "Finding concise plans: Hardness and algorithms," in *Proc. IEEE International Conference on Intelligent Robots and Systems*, 2014.

[26] N. Roy, G. Gordon, and S. Thrun, "Finding Approximate POMDP solutions Through Belief Compression," *Journal of Artificial Intelligence Research*, vol. 23, pp. 1–40, 2005.

[27] Y. Song and J. M. O'Kane, "Comparison of constrained geometric approximation strategies for planar information states," in *Proc. International Conference on Robotics and Automation*, 2012.

[28] R. Szczerba, D. Chen, and K. Klenk, "Minimum turns/shortest path problems: a framed-subspace approach," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 1997, pp. 398–403.

[29] R. H. Taylor, M. T. Mason, and K. Y. Goldberg, "Sensor-based manipulation planning as a game with nature," in *International Symposium on Robotics Research*, R. C. Bolles and B. Roth, Eds. Cambridge MA: MIT Press, 1988.

[30] B. Tovar, F. Cohen, and S. M. LaValle, "Sensor beams, obstacles, and possible paths," in *Algorithmic Foundations of Robotics, VIII*, G. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. Berlin: Springer-Verlag, 2009.

[31] B. Tovar, R. Murrieta-Cid, and S. M. LaValle, "Distance-optimal navigation in an unknown environment without sensing distances," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 506–518, June 2007.

[32] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–86, 1967.

[33] J. Yu and S. M. LaValle, "Shadow information spaces: Combinatorial filters for tracking targets," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 440–456, 2012.

Jason M. O'Kane Dr. O'Kane is Associate Professor and Director of the Center for Computational Robotics in the Department of Computer Science and Engineering at the University of South Carolina. He holds the Ph.D. (2007) and M.S. (2005) degrees from the University of Illinois and the B.S. (2001) degree from Taylor University (Upland, Indiana, USA), all in Computer Science. His research interests span sensor-based algorithmic robotics and related areas, including planning under uncertainty, artificial intelligence, computational geometry, sensor networks, and motion planning.

Dylan A. Shell Dr. Shell is an Associate Professor of Computer Science at Texas A&M University. He holds a Ph.D. in Computer Science from the University of Southern California. He has published papers on multi-robot task allocation, robotics for emergency scenarios, and biologically inspired multiple robot systems.