Automatic Reduction of Combinatorial Filters

Jason M. O'Kane and Dylan A. Shell

***Abstract*— We consider the problem of filtering whilst maintaining as little information as possible to perform a given task. The literature includes several illustrations of how adroit choices for state descriptions may lead to concise — or even minimal — filters tailored to specific tasks. We introduce an efficient algorithm which is able to reproduce these hand-crafted solutions.**

Specifically, our 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. 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. In the experiments we describe, our algorithm produces optimal or near-optimal reduced filters for a variety of problem instances.

These reduced filters are of interest for several reasons, including their direct application on platforms with severely limited computational power and in systems that require communication over low-bandwidth noisy channels. Moreover, inspection of reduced filters may provide insights into the structure of a problem that can guide the design of the other elements of a robot system.

I. Introduction

A central question facing the designer of any autonomous robot is to determine how the robot should process and store information from 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. A recent line of research has considered solutions to this problem using *combinatorial filters*, which are carefully crafted to retain only the information that is essential to completing the robot's task [11], [17], [19]. The goal of this paper is to investigate algorithms for automatically constructing optimal combinatorial filters.

As a simple example, consider the problem depicted in Figure 1, which was introduced by Tovar, Cohen, and LaValle [16]. In this problem, two agents move through an annulus-shaped environment. The environment is subdivided 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 determine, at all times, whether the agents are in the same region. An obvious approach is to define a nine-element state space

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.

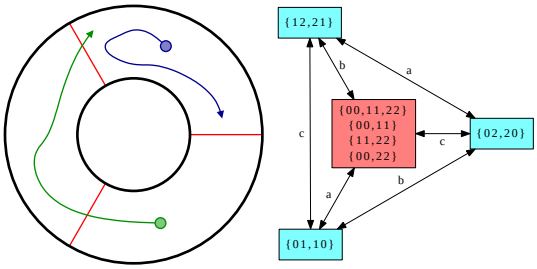


Fig. 1: [left] Two agents move amidst three beam sensors. [right] An optimal combinatorial filter, first discovered by Tovar, Cohen, and LaValle and reproduced by our algorithm, for tracking whether the agents are in the same region. The numbers 0, 1, and 2 denote the regions, and the letters a, b, and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

$X = \{0, 1, 2\} \times \{0, 1, 2\}$, 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 us to track which of $2^9 = 512$ 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 "agents are together" and three I-states representing "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 describe an algorithm that is able to automatically replicate this kind of filter reduction, which heretofore has required clever hand-crafting.

More generally, our results are applicable to any filtering task that can be described in terms of discrete transitions between finitely many information states (which need not necessarily represent sets of possible states as in the example above), triggered by finitely many events such as observations from sensors or actions executed by a robot. A filtering task in this context is determined by a partition or "coloring" of the I-states, indicating which I-states that filter must be able to tell apart.

We present an algorithm that accepts this kind of colored graph as input, and outputs a reduced graph that is provably equivalent to the input graph. The intuition of our approach is to identify pairs of same-colored I-states that must remain distinct to ensure that the reduced filter produces correct results, and to use a graph coloring subroutine to refine the original coloring, eliminating those conflicts. When no conflicts remain, we merge each pair I-states that have the same color to form the reduced filter.

We also show that the problem of determining whether a given filter can be reduced to a given size is NP-complete. As a result, no polynomial time algorithm exists for this problem, unless $\mathrm{P} = \mathrm{NP}$. This limitation is evident in our algorithm in the graph coloring subroutine which dominates its run time; our algorithm is efficient only when that graph coloring is approximate rather than optimal. We consider several quadratic-time approximate coloring algorithms which, in our experiments, produce optimal or near-optimal reduced filters.

We believe that these kinds of reduced combinatorial filters are of interest for several reasons.

1) First, reduced combinatorial filters, which require very small quantities of memory, can be directly useful on platforms that have, because of constraints on space, weight, or energy, severely limited computation power.
2) Second, for systems that require communication between multiple robots over a low-bandwidth noisy channel, using a reduced filter to maintain the information to be communicated can optimize the number of bits that must be transmitted.
3) Finally, inspection of reduced filters may reveal insights into the structure of the problem that we can exploit. For example, if the algorithm generates a reduced filter that does not utilize the output of one or more sensors, we can conclude that those sensors are unnecessary for completing the task, and revise the system's hardware design accordingly.

After reviewing related work in Section II, this paper makes several specific contributions:

- In Section III, we define the filter minimization problem in a precise, general way.
- In Section IV, we prove that the problem of finding an optimal reduction of a given filter is NP-hard.
- In Section V, we present an efficient algorithm for filter reduction that often generates optimal or near-optimal filters.
- In Section VI, we describe an implementation and a series of quantitative experiments that measure the performance of our algorithm.

The paper concludes with a discussion of future work in Section VII.

II. Related Work

The kinds of combinatorial filters and reduced I-states we study in this paper have a long history—see, for example, the sensorless manipulation work of Erdmann and Mason [6] and Goldberg [7] which uses transition graphs to represent the evolution of a robot's uncertainty—and were formalized in a general way by LaValle [10], [11].

A number of recent papers have presented combinatorial filters for such tasks as target tracking [19], mobile robot navigation [12], [17], and manipulation [9]. 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 paper 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 [4], [5]. 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 [13], [15] 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 [14] 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.

III. Definitions and Problem Formulation

We consider filters that have access to a stream of discrete *observations* from a finite *observation space* denoted Y. Each observation $y \in Y$ corresponds to a discrete unit of information that becomes available to the filter. These will generally be readings produced by sensors, but may also be *actions* that a robot executes, as observed by a passive filter on that robot. For the purposes of reduction in this paper, the difference between observations and actions is not important. To simplify the language we use the term "observations" exclusively.

Following the terminology introduced by LaValle [10], 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: An I-state graph $\mathbf{G}$ *is a edge-labelled directed graph supplemented with a starting vertex,* i.e., $\mathbf{G} \triangleq (V, E, l : E \to Y, v_0)$*, in which*

1) *the finite set V contains vertices which we call "I-states",*
2) *the set E 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 $v_0 \in V$.*

An additional and important requirement on l is that if $e_1 \triangleq (v, v_j)$ and $e_2 \triangleq (v, v_k)$ with $v_j \neq v_k$ then $l(e_1) \neq l(e_2)$. That is, no two edges originating from the same vertex have the same label. 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 $y_0 y_1 \cdots y_n$, one may trace these on $\mathbf{G}$ by starting at v_0 and following the edges labelled by each y_i, one after another. If all of the corresponding edges exist, then the resulting I-state is uniquely determined. However, because we do not require the image of l to be its entire codomain, it is possible that no edge labelled with $y_i \in Y$ exists from the current I-state. This occurs when constraints imposed by the structure of the underlying problem indicate that an observation cannot occur at a given I-state. 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 $\mathbf{G}$*, 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* $\mathbf{G}$ *by starting at its initial vertex.*

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

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

The interpretation is that observations are made and information retained, and ultimately a color is reported as the filter output. 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 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 paper is to reduce 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 $\mathbf{F_1} \triangleq (V, E, l : E \to Y, v_0, c_1)$*, and* $\mathbf{F_2} \triangleq (W, F, m : F \to Y, w_0, c_2)$ *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* $l \in \mathcal{L}$*, the I-states* v^l *and* w^l *reached by tracing* l *on* $\mathbf{F_1}$ *and* $\mathbf{F_2}$ *respectively are both defined, and we have* $c_1(v^l) = c_2(w^l)$*. We denote this equivalence relation with* $\mathbf{F_1} \overset{\mathcal{L}}{=} \mathbf{F_2}$.

Assuming we are given an initial filter $\mathbf{F}$ as the specification of a filtering task, we are concerned with other filters that are equivalent on the language induced by $\mathbf{F}$, *viz.* those filters $\mathbf{F}'$ where $\mathbf{F} \overset{\mathcal{L}(\mathbf{F})}{=\!=} \mathbf{F}'$. (Note that one needs some care since $\mathbf{F} \overset{\mathcal{L}(\mathbf{F})}{=\!=} \mathbf{F}' \not\Rightarrow \mathbf{F} \overset{\mathcal{L}(\mathbf{F}')}{=\!=} \mathbf{F}'$.) Comparing the cardinality of the vertex sets of $\mathbf{F}$ and $\mathbf{F}'$, one obtains a relative measure of the memory required by implementations of either filter. It is natural to consider the *filter minimization problem*:

Problem: Filter Minimization (FM**)**

Input: A filter $\mathbf{F}$.

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

IV. Hardness of Filter Minimization

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

Decision Problem: Filter Minimization (FM-DEC**)**

Input: A filter $\mathbf{F}$ and $k \in \mathbb{N}^+$.

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

The primary result of this section is that FM-DEC is an NP-complete problem, which directly implies that FM is NP-hard. We proceed by arguing that FM-DEC is in complexity class NP (Section IV-A) and by providing a polynomial time reduction from the problem of 3-coloring a graph —a known NP-complete problem— to FM-DEC (Section IV-B). Finally, Section IV-C briefly discusses the relationship between this problem and the closely related (but efficiently solvable) problem of minimizing deterministic finite automata.

A. 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 $\mathbf{F_1} \overset{\mathcal{L}(\mathbf{F_1})}{=\!=} \mathbf{F_2}$.

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.

B. Filter Minimization is NP-complete

The previous section showed that verifying the correctness of a reduced filter can be done efficiently. 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 $\mathbf{F_1} \overset{\mathcal{L}(\mathbf{F_1})}{=\!=} \mathbf{F_2}$*, or False otherwise.*

if $c_1(v_1) \neq c_2(v_2)$ **return** False
$Q \leftarrow (v_1, v_2)$ {Initialize queue with the start vertices.}
while Q is not empty **do**
  $(v_1, v_2) \leftarrow Q.\text{pop}()$
  for each edge $v_1 \xrightarrow{y} w_1$ in E_1 **do**
    if E_2 has an edge $v_2 \xrightarrow{y} w_2$ **then**
      if $c_1(w_1) \neq c_2(w_2)$ **then**
        return False {$\mathbf{F_2}$ produces an incorrect color.}
      else if (w_1, w_2) has not been enqueued before **then**
        $Q.\text{insert}((w_1, w_2))$
      end if
    else
      return False {$\mathbf{F_2}$ terminates where $\mathbf{F_1}$ does not.}
    end if
  end for
end while
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 [2]. 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 $v_0 \xrightarrow{y_i} v$ in E_2, by ensuring that (v_0, v) is in E_2 and $l : (v_0, v) \mapsto y_i$.
4) Create two additional vertices in V_2 named v_R and v_G. Let $c : v_R \mapsto 3$ and $c : v_G \mapsto 4$. The intuition of these vertex names is to suggest the colors "red" and "green."
5) For each edge (v_i, v_j) in E_1, create a new observation y_{ij} in Y and two new edges $v_i \xrightarrow{y_{ij}} v_R$ and $v_j \xrightarrow{y_{ij}} v_G$, *i.e.*, (v_i, v_R) and (v_j, v_G) are in E_2, and $l : (v_i, v_R) \mapsto y_{ij}$ and $l : (v_j, v_G) \mapsto y_{ij}$.
6) Set $k = 6$.

The intuition of 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 2 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 exists a reduced filter $\mathbf{F_3}$ with at most 6 vertices, such that $\mathbf{F_2} \overset{\mathcal{L}(\mathbf{F_2})}{=\!=} \mathbf{F_3}$. Let us consider each direction of this proposition in turn.

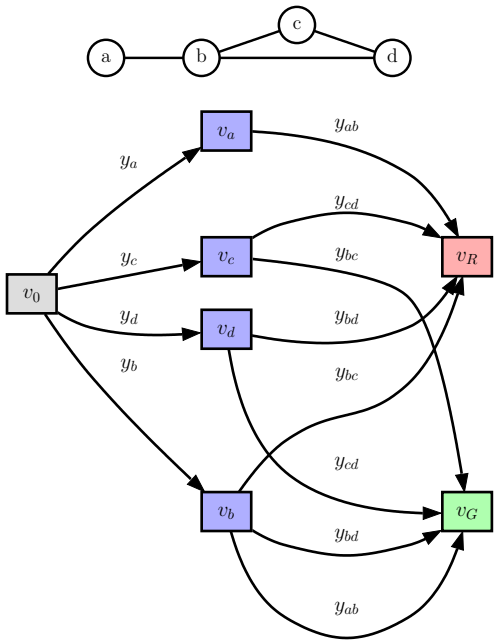


Fig. 2: [top] An example instance of GRAPH-3C. [bottom] The corresponding instance of FM-DEC. The vertices in the left column have color 1, the middle column has color 2, and the vertices in the right column have colors 3 and 4. Our constructed instances use $k = 6$.

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_R, v_G, and at most three vertices associated with the three distinct colors in c_1. Figure 3 illustrates this construction for the example introduced in Figure 2.

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.

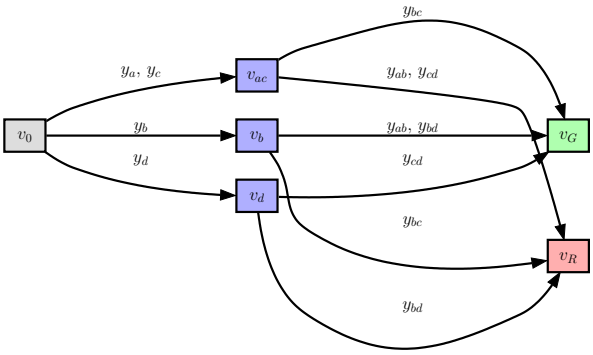
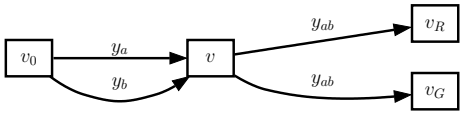


Fig. 3: A filter equivalent to the filter shown in Figure 2. Because the original graph is 3-colorable, the filter can be reduced to one with only six vertices.



Note that 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 $\mathbf{G_1}$. Therefore $\mathbf{F_3}$ 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: Use 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, $\mathbf{F_3}$ has edges $v_0 \xrightarrow{y_a} v_{c_1(v_a)}$ and $v_0 \xrightarrow{y_b} v_{c_1(v_a)}$. Therefore, the observation sequences $y_a y_{ab}$ and $y_b y_{ab}$ generate the same color in $\mathbf{F_3}$. 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. □

Finally we can assemble these partial results.

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

Proof: Combine Lemmas 2 and 3. □

Theorem 5: FM-DEC *is NP-complete.*

Proof: Combine Lemmas 1 and 4. □

Theorem 6: FM *is NP-hard.*

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

C. Relationship to DFA minimization

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

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 [8].

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. Perhaps somewhat surprisingly, the need to perform these merges in a globally optimal way leads to the hardness result expressed in Theorem 6.

V. Approximate Filter Minimization

In the previous section, we showed that, under widely-accepted complexity assumptions, the optimal filter minimization problem cannot be solved by any polynomial-time algorithm. In this section, we present an efficient technique for *approximate filter minimization*. That is, we describe an algorithm whose input is a filter $\mathbf{F_1}$, and whose output is another filter $\mathbf{F_2}$, for which $\mathbf{F_1} \overset{\mathcal{L}(\mathbf{F_1})}{=\!=} \mathbf{F_2}$ and $|V_1| \leq |V_2|$. In contrast to the optimal filter minimization problem discussed in Section IV, we do not require $\mathbf{F_2}$ to be the smallest filter with this property.

A. Algorithm Description

The intuition of 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

Algorithm 2 Approximate 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* $\mathbf{F_1} \overset{\mathcal{L}(\mathbf{F_1})}{=\!=} \mathbf{F_2}$.

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 approximate 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** $\mathbf{F_2}$

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 $\mathbf{F_2}$.

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.*

Definition 6: In a filter, the conflict graph for color k *is an undirected graph with vertex set* $\{v \in V \mid c(v) = k\}$ *and edge set* $\{(v, w) \in E \mid v$ *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, for which pseudocode appears as 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 V-C discusses a few options for how one might instantiate this black box, and the experiments in Section VI 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.

B. Correctness and Runtime

The next two lemmas confirm that Algorithm 2 terminates and returns a correct answer.

Lemma 7: 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 $n(\mathbf{F}) = 0$, then every vertex in $\mathbf{F}$ has a distinct color, so by definition there are no conflicts. Therefore, the algorithm must terminate on iteration $|V|^2$, if not before. □

Lemma 8: Algorithm 2 correctly produces a filter equivalent to $\mathbf{F_1}$.

Proof: Note that 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 $\mathbf{F_1} \overset{\mathcal{L}(\mathbf{F_1})}{=\!=} \mathbf{F_2}$. Therefore, Algorithm 2 is correct. □

To bound the run time 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 $\mathbf{F}$ 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, note that this a pessimistic bound: In practice, the algorithm generally uses far fewer than $|V|^2$ iterations of its outer loop.

C. 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 [2]. Therefore, Algorithm 2 can only be efficient if the subroutine we use to color the conflict graphs is only approximate.

A large family of approximate graph coloring algorithms have been proposed [1], [18], any of which would be suitable for our approach. Our implementation uses *sequential greedy coloring* [3] 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.

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

VI. Experimental Results

We have implemented Algorithms 1 and 2 in C++. This section presents results showing its 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.

We considered two generalizations of problem shown in Figure 1, in which we varied both the number of agents and the number of beam sensors subdividing the annulus. 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 4 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 5 shows two of the reduced filters for the annulus problem. Note that the equivalence of these 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 Figure 1 in which there are two agents, the number of beam

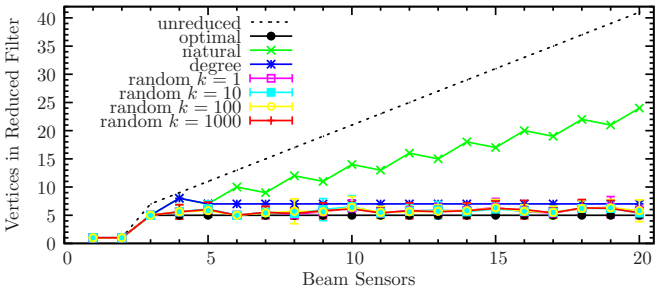


Fig. 4: Solution quality for automatically reduced filters for a single agent moving in an annulus amongst beam sensors. For randomized algorithms, the error bars show the standard deviation after 10 trials.

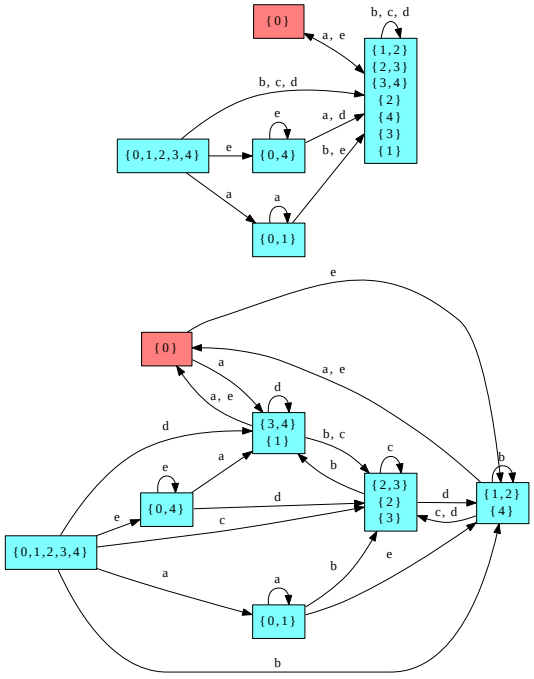


Fig. 5: [top] A 5-vertex filter for the 5-sector 1-agent donut problem, generated by Algorithm 2, using optimal colorings for the conflict graphs. [bottom] An equivalent filter using seven vertices for the same problem, generated using the best of 10 colorings of each conflict graph, each based on sequential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

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 6 shows results of this experiment, which used the same conditions as described above.

Third, we considered the L-shaped corridor problem introduced in Section 11.3.1 of LaValle's book [10]. 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 size of the reduced filters increases only linearly. Figure 7 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.

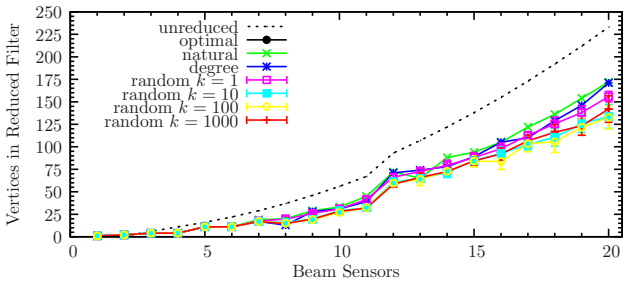


Fig. 6: Solution quality for automatically reduced filters for a two agents moving in an annulus amongst beam sensors. For randomized algorithms, the error bars show the standard deviation after 10 trials.

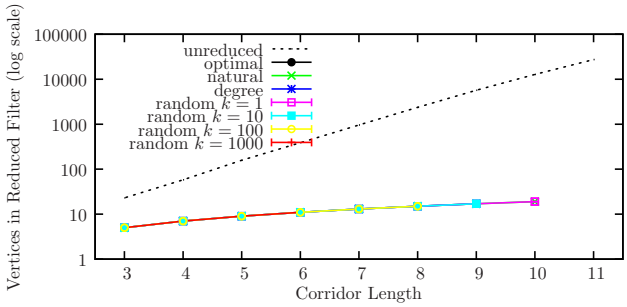


Fig. 7: Solution quality for automatically reduced L-corridor filters. For randomized algorithms, the error bars show the variance after 10 trials. Note the logarithmic scale on the vertical axis. The optimal curve is not visible because it coincides with the experimental data.

VII. Conclusions

In this paper, we proved that optimal minimization of combinatorial filters is NP-hard. We also presented an efficient algorithm to perform these minimizations that often produces optimal or near optimal reduced filters. There remain a number of interesting unanswered questions.

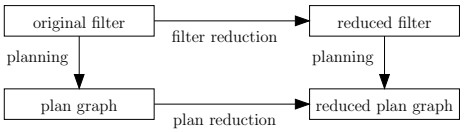
A. Sub-optimality bounds

For some problems, such as the families of filters referred to in Figures 4 and 7, 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.

B. Relationship to planning

In this paper, we focused exclusively on the passive problem of filtering the information available to the robot, without regard for the related planning problems. Indeed, if we consider information feedback plans $\pi : V \to U$ which map I-states to actions, then we can naturally extend the filters described here into *plan graphs* in which each vertex (that is, each I-state) is labelled with the action that should be executed at that I-state. This gives rise to a number of questions on plan reduction and its relationship to filter reduction.

The diagram below depicts one way to visualize the operations that might be performed on filters (top row) and plan graphs (bottom row):



The current paper has considered only filter reduction. An interesting unanswered question is to determine the conditions under which the diagram "commutes." Is it better to reduce and then plan, to plan and then reduce, or are these two options ultimately equivalent to each other?

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS-0953503.

References

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

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

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

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

[5] ——, "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.

[6] 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.

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

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

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

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

[11] ——, "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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.