On nondeterminism in combinatorial filters

Yulin Zhang and Dylan A. Shell

Abstract— **The problem of combinatorial filter reduction arises from resource optimization in robots; it is one specific way in which automation can help to achieve minimalism, to build better robots. This paper contributes a new definition of filter minimization that is broader than its antecedents, allowing filters (input, output, or both) to be nondeterministic. This changes the problem considerably. Nondeterministic filters may re-use states to obtain more 'behavior' per vertex. We show that the gap in size can be significant (larger than polynomial), suggesting such cases will generally be more challenging than deterministic problems. Indeed, this is supported by the core complexity result established in this paper: producing nondeterministic minimizers is PSPACE-hard. The hardness separation for minimization existing between deterministic filter and automata, thus, fails to hold for the nondeterministic case.**

I. Introduction

With increasingly complex robots, one naturally turns to computational tools to help automate design processes. This leads directly to the practical question of how to reduce a robot's resource footprint. Minimizing resources causes one to reason about their necessity, which furnishes more fundamental insights about the underlying information requirements of particular robot tasks [1]. This paper focuses on minimizing state in *combinatorial filters* [2], discrete variants of the probabilistic estimators so widely used in robotics [3]. While their minimization problem is easy to formulate (to wit: reduce the number of states while preserving input–output behavior), it is computationally hard to solve.

Combinatorial filter reduction was first introduced as an open question by Tovar et al. [4, pg. 12]. They introduced the scenario in Fig. 1a to exemplify the problem: two agents wander in a circular world, and three sensor beams (producing symbols 'a', 'b', and 'c', resp.) partition the environment into sector-shaped regions (labeled 0, 1, 2). The beams detect if an agent crosses the dividing line but senses neither the agent's identity nor direction of motion. With the agents starting in some known configuration, the task is, given a sequence of sensor readings (i.e., a string of a's, b's, c's), to determine whether the pair are in the same sector or not. This problem may be solved via a filter, a finite transition system akin to a Moore machine transducer whose vertices bear an output (or color). Starting at the initial state, one traces the input string forward to produce a sequence of colors that represent estimates. When every string gives a solitary tracing, the filter is *deterministic*. One wonders: what is the smallest filter for tracking the co-location of our two agents? Only 4 states are required. See Fig. 1d; the minimal filter is deterministic. Other than human nous (how Tovar & friends did it), one may produce a minimal instance by starting with the filter obtained by directly transcribing of the problem, and applying a reduction algorithm.

Yulin Zhang is with the Dept. of Computer Science, The University of Texas, Austin, TX, USA. yulin@cs.utexas.edu. Dylan A. Shell is with Dept. of Computer Science & Engineering, Texas A&M University, College Station, TX, USA dshell@tamu.edu.

This work was supported by the National Science Foundation through awards IIS-1453652, IIS-1849249, and IIS-2034097.

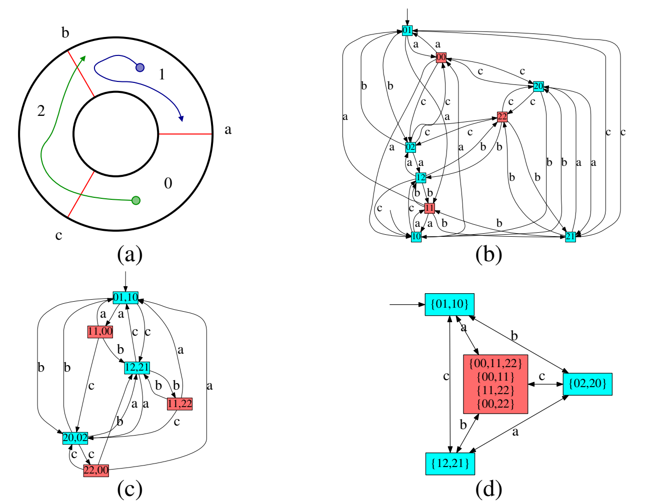


Fig. 1: Combinatorial filter minimization, as originally motivated by [4]: (a) A pair of agents move in a circular environment with three beam sensors, partitioned into regions, indexed by numbers 0, 1 and 2. Letters 'a', 'b', and 'c' denote observations from each of the three beams. (b) A nondeterministic filter to estimate whether these two agents are in the same region (red) or not (cyan). (c) The deterministic version of the same. (d) A *minimal* deterministic filter.

Most prior work on combinatorial filters, including all research on filter reduction until now [5]–[9], concerns deterministic filters. The present paper, in its first part, presents a compelling practical case for the utility of filter minimization methods that accommodate nondeterminism. The second part of this paper examines the hardness of the minimization problem for filters with nondeterministic inputs, including finding both deterministic and nondeterministic minimizers for nondeterministic input filters. We show that, under commonly held computational complexity assumptions, these problems are harder than the deterministic case. In what follows, we leverage hardness results from automata theory to establish these facts, which has the important added benefit of leading to a broader and clearer understanding of the relationship between filter and automata minimization.

II. The value of nondeterminism in minimizing combinatorial filters

Existing research on combinatorial filter reduction [5]–[9] only deals with deterministic input filters and deterministic output filters (or minimizers). To understand the implications of this, let's return to Fig. 1 in some detail. To arrive at the 4-state minimizer, we begin with the diagram in Fig. 1a. Using the assumption of continuous motion and beginning at a state representing the initial agent configuration, we trace all possible events forward, coloring the conditions encountered appropriately (red for together; cyan otherwise). The result, Fig. 1b, is not deterministic. To apply a minimization algorithm, the filter must be converted to an equivalent one that is. The process of determinizing produces a filter (Fig. 1c) that can then be fed into a minimization method to yield Fig. 1d. This procedure goes from 9 states, to 6, before reaching 4.

But now consider the nondeterministic 5-state filter in Fig. 2a. To find a minimal filter, it can be determinized (via a power set construction [10]) to track the $2^4 = 16$ distinct information states shown in Fig. 2b. Once minimized, it gives the deterministic filter in Fig. 2c. The growth in the number of vertices, caused by the need to determinize for the minimization algorithm, indicates trouble. Not only does the set increase exponentially, but this much larger object becomes the input for an exponential cost algorithm (as the problem is NP-hard [5]). Double trouble.

To by-pass this expansion, one requires filter reduction methods that are able to consume nondeterministic filters directly as input. Looking again at Figs.. 2a and 2c, the dramatic compression that cancels the extreme expansion raises some questions. Do large deterministic instances arising from small nondeterministic ones really induce hard minimization problems? Or are they instead structured in some special (sparse or low-density) form, conserving underlying information? Computational complexity provides clues: e.g., in characterizing the space requirements of direct nondeterministic filter to deterministic minimizer computation.

If nondeterminism can be of added value as input to a minimization algorithm, what about as its output? In finite automata minimization, the smallest nondeterministic automata can be smaller than any deterministic one. Typical examples exploit the fact that accepting a string in the nondeterministic automaton requires that *some* tracing arrive at an accepting state. For filters, analogous instances fail owing to their differing semantics (stated formally in the next section). The analogous fact, however, does hold. A small example suffices to show this: the deterministic input filter given in Fig. 3a has 19 states, but can be reduced to a deterministic minimizer with size 14. This minimizer has a single color selected for each of the 5 leaf states which have a choice, and then merges identically colored

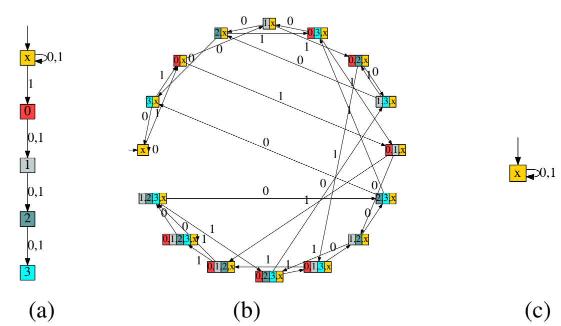


Fig. 2: (a) A nondeterministic filter. (b) A deterministic form obtained via the power set construction. (c) A minimal filter.

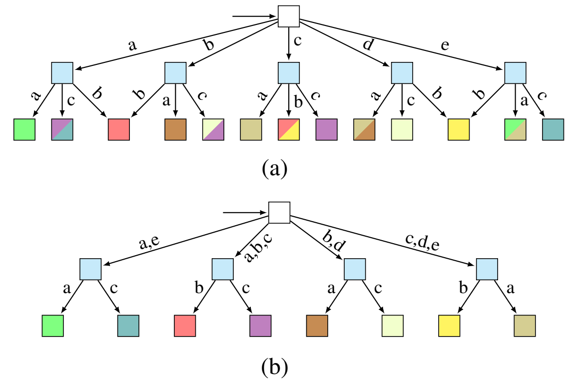


Fig. 3: (a) A 19-state deterministic filter that has no deterministic minimizer with fewer than 14 states. (b) A minimal nondeterministic 13-state minimizer for the filter above.

leaves. However, the filter can be shrunk still further: Fig. 3b gives a nondeterministic minimizer possessing only 13 states. Nondeterminism, then, provides extra freedom that can be exploited to further reduce filter size.

To summarize: nondeterminism may be practical importance for two reasons: (i) nondeterminism in the input allows minimization to proceed directly on models of certain problems, potentially saving on expensive intermediate steps; (ii) permitting nondeterminism in the filters produced as output can deliver greater compression.

III. Combinatorial filters and their minimization

We first give our model of combinatorial filters:

Definition 1 (procrustean filter [10])**.** A *procrustean filter*, *p-filter* or *filter* for short, is a tuple (V, V_0, Y, τ, C, c) where V is the set of states, V_0 is the set of initial states, Y is the set of of observations, $\tau : V \times V \to 2^Y$ is the transition function, C is the set of outputs (or colors), and $c : V \to 2^C \setminus \{\varnothing\}$ is the output function.

We write the states, initial states and observations for a filter $\mathscr{F}$ as $V(\mathscr{F})$, $V_0(\mathscr{F})$ and $Y(\mathscr{F})$. A filter $\mathscr{F} = (V, V_0, Y, \tau, C, c)$ is *deterministic*, if $|V_0| = 1$ and for every $v_1, v_2, v_3 \in V$ with $v_2 \neq v_3$, $\tau(v_1, v_2) \cap \tau(v_1, v_3) = \varnothing$. Otherwise, we say $\mathscr{F}$ is *nondeterministic*. A filter can also be viewed as a graph with states being its vertices, and transitions being directed edges.

In a filter $\mathscr{F} = (V, V_0, Y, \tau, C, c)$, an observation sequence (or a string) $s = y_1 y_2 \ldots y_n \in Y^*$ *reaches* a state w from state v, if there exists a sequence of states $w_0, w_1, \ldots, w_n$ in $\mathscr{F}$, such that $w_0 = v, w_n = w$, and $\forall i \in \{1, 2, \ldots, n\}$, $y_i \in \tau(w_{i-1}, w_i)$. In a filter $\mathscr{F}$, for every state v, if there exists a string that reaches v from some initial state, then we say $\mathscr{F}$ is *trim*. Any filter that is not trim can be made so by removing the states that are not reached by any string from the initial states. We consider filters that are trim, w.l.o.g.

We collect the set of all states reached by s from some initial state $v_0 \in V_0$, and denote it as $\mathcal{V}_{\mathscr{F}}(s)$. Specifically, for the empty string ϵ, we have $\mathcal{V}_{\mathscr{F}}(\epsilon) = V_0$. If no states are reached by some string from any initial state, then we say that string *crashes* on $\mathscr{F}$. The set of strings that do not crash on $\mathscr{F}$ is called the *interaction language* of $\mathscr{F}$, and is written as $\mathcal{L}^I(\mathscr{F}) = \{s \in Y^* \mid \mathcal{V}_{\mathscr{F}}(s) \neq \varnothing\}$. The *output of string* s on filter $\mathscr{F}$ is the set of outputs (or colors) of all states reached by s from some initial state, and is written as $\mathcal{C}(\mathscr{F}, s) = \cup_{v \in \mathcal{V}_{\mathscr{F}}(s)} c(v)$.

In minimizing a filter, we are interested in reduced filters that simulate the given filter in terms of outputs on its strings:

Definition 2 (output simulating)**.** Let $\mathscr{F}$ and $\mathscr{F}'$ be two filters, then $\mathscr{F}'$ *output simulates* $\mathscr{F}$ if the following properties hold: (i) language inclusion: $\mathcal{L}^I(\mathscr{F}) \subseteq \mathcal{L}^I(\mathscr{F}')$; (ii) output consistency: $\forall s \in \mathcal{L}^I(\mathscr{F})$, $\mathcal{C}(\mathscr{F}', s) \subseteq \mathcal{C}(\mathscr{F}, s)$.

This requires that $\mathscr{F}'$ be capable of processing all the inputs which $\mathscr{F}$ can, and produce outputs that $\mathscr{F}$ could. The input set is no smaller; the set of outputs no larger.

We wish to find minimal filters:

Problem: Filter Minimization (PFM)
Input: A filter $\mathscr{F}$.
Output: A filter $\mathscr{F}^\dagger$ with fewest states, such that $\mathscr{F}^\dagger$ output simulates $\mathscr{F}$.

This is a generalization of its deterministic version in the work [8], [9], which dealt only with deterministic input and deterministic minimizer. We use 'PF' to denote the fact that both the input and output of this problem can be general nondeterministic p-filters, and use 'M' for minimization. Additionally, we designate the problem of producing a *deterministic minimizer* for a nondeterministic input filter 'PFDM', a four-letter word where 'D' stands for deterministic.

IV. Background and preliminaries

We make use of some known facts from automata theory.

A finite automaton (NFA) is a tuple $(Q, Q_0, \Sigma, \delta, A)$, where Q, Q_0, Σ, δ, A are the states, initial states, alphabet (observations), transition function, and accepting states. Both filters and NFAs are similar, both being transition structures. Different from a filter, an NFA $\mathscr{A}$ has accepting states not outputs (colors). For automata, we are interested in the strings that reach some accepting states, which we term the accepting language $\mathcal{L}^A(\mathscr{A})$. An NFA with a singleton Q_0 and deterministic transition structure is also called a DFA.

Here are some results from automata theory:

Lemma 3 ([11])**.** Given two NFAs $\mathscr{A}$ and $\mathscr{B}$, it is PSPACE-complete to check if $\mathcal{L}^A(\mathscr{A}) = \mathcal{L}^A(\mathscr{B})$.

Lemma 4 ([12])**.** For a given NFA $\mathscr{A} = (Q, Q_0, \Sigma, \delta, A)$, it is PSPACE-complete to check whether $\mathcal{L}^A(\mathscr{A}) = \Sigma^*$.

Lemma 5 ([13])**.** Given a set of DFAs $\{\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n\}$ with common alphabet Σ, it is PSPACE-complete to check if $\cup_{1 \le i \le n} \mathcal{L}^A(\mathscr{A}_i) = \Sigma^*$.

V. Complexity of filter minimization

We are now ready to show that finding minimizers for nondeterministic input filters is hard.

The decision problem of the PFM problem is:

Decision Problem: P-filter Minimization (PFM-DEC)
Input: A filter $\mathscr{F}$ and $k \in \mathbb{N}^+$.
Output: YES only if there exists some $\mathscr{F}^\dagger$ with no more than k states, such that $\mathscr{F}^\dagger$ output simulates $\mathscr{F}$.

Analogously, PFDM-DEC is the decision version of PFDM.

A. The hardness of PFM and PFDM

Now, we will show that the decision versions of PFM and PFDM are, respectively, PSPACE-complete and PSPACE-hard. Consequently, both PFM and PFDM are PSPACE-hard.

To check the output simulation requirement in polynomial time, the following filter product will be helpful.

Definition 6 (tensor product)**.** Given filters $\mathscr{F}_1 = (V^1, V_0^1, Y^1, \tau^1, C^1, c^1)$ and $\mathscr{F}_2 = (V^2, V_0^2, Y^2, \tau^2, C^2, c^2)$, their product, a graph denoted $(\mathscr{F}_1 \odot \mathscr{F}_2)$, is constructed to capture strings in $\mathcal{L}^I(\mathscr{F}_1)$ via:

1) List all pairs of vertices in $V(F_1) \times (V(F_2) \cup \{\ominus\})$, where $\ominus$ is a placeholder for an empty vertex.
2) Mark vertex (v, w) an initial state in graph $(\mathscr{F}_1 \odot \mathscr{F}_2)$.
3) Build a transition from (v, w) to (v', w') under label y if $y \in \tau^1(v, v')$ and $y \in \tau^2(w, w')$. Notice that if y is not an outgoing label of vertex v, then we say $y \in \tau^1(v, \ominus)$.
4) Remove the pairs reached from any initial state.

The tensor product of two filters is a transition structure with initial states, i.e., a graph.

Lemma 7. PFM-DEC is in PSPACE.

Proof. Two steps: polynomial space suffices (1) to represent and search for a filter, and (2) to ascertain whether a filter output simulates $\mathscr{F}$. For (1), since PFM-DEC requires we encode filters of size k, we need to keep track of at most k^2 transitions, at most $|Y|$ labels for each transition, at most $|C|$ colors for each state, and at most k initial states. The space needed to enumerate output filters is $O(k^2 \times |Y| + k \times |C|)$.

For (2), we must verify both language inclusion and output consistency. To show $\mathcal{L}^I(\mathscr{F}) \subseteq \mathcal{L}^I(\mathscr{F}')$, form product graph $\mathscr{G} = \mathscr{F} \odot \mathscr{F}'$. If there is no vertex $(v, \ominus)$ in $\mathscr{G}$ such that $v \in V(\mathscr{F})$, then $\mathcal{L}^I(\mathscr{F}) \subseteq \mathcal{L}^I(\mathscr{F}')$ since every string reaching a vertex in $\mathscr{F}$ also reaches some vertex in $\mathscr{F}'$. If there exists some such a vertex $(v, \ominus)$, then we must determine whether the strings reaching $(v, \ominus)$ also reach some vertex in $\mathscr{F}'$. We build an NFA $\mathscr{A}$ from $\mathscr{G}$ by treating all states $\{(v, \ominus) \mid v \in V(\mathscr{F})\}$ as accepting states. Next, construct a second NFA, $\mathscr{B}$, from $\mathscr{F}'$ by treating every state in $\mathscr{F}'$ as accepting. Then we must show that strings reaching every $(v, \ominus)$ are accepted by $\mathscr{B}$, i.e., whether $\mathscr{A}$ and $\mathscr{A} \cap \mathscr{B}$ are equivalent (where $\cap$ is automata intersection). Creating automata $\mathscr{A}$ and $\mathscr{A} \cap \mathscr{B}$, and, via Lemma 3, showing their equivalence is in PSPACE.

Verifying output consistency also needs only polynomial space. Remove the states of the form $(v, \ominus)$ from $\mathscr{G}$, then, for every state (v, w) in $\mathscr{G}$ such that $c(v) \not\supseteq c(w)$, to output simulate, for every output $o \in c(w) \setminus c(v)$, strings reaching (v, w) must reach some state u in $\mathscr{F}$ with $o \in c(u)$. Otherwise o is not a legal output for some string. To see whether o is a legal output, we build an NFA $\mathscr{M}$ from $\mathscr{G}$ by treating (v, w) as accepting states, and another NFA $\mathscr{N}$ from $\mathscr{F}$ by treating the states with color o as accepting states. If $\mathcal{L}^A(\mathscr{M}) \subseteq \mathcal{L}^A(\mathscr{N})$, then o is safe. If every $o \in c(w) \setminus c(v)$ is safe, then the output of $\mathscr{F}'$ is consistent on that of $\mathscr{F}$. Otherwise, (v, w) is certificate for violation of output consistency. This procedure takes polynomial space. □

Lemma 8. PFM-DEC is PSPACE-hard.

Proof. We give a polynomial time reduction from NFA universality (Lemma 4) to PFM-DEC. To show the accepting language of a given NFA $\mathscr{A} = (Q, Q_0, \Sigma, \delta, A)$ is Σ^*, we first create a filter $\mathscr{F}$ from $\mathscr{A}$ in polynomial time as follows:

1) Add the states, transitions, initial states of $\mathscr{A}$ to the states, transitions, initial states of $\mathscr{F}$.
2) Add a new initial state v to $\mathscr{F}$, with a self loop bearing all labels Σ from $\mathscr{A}$.
3) Add a new vertex w to $\mathscr{F}$. For every state in $\mathscr{F}$ arising from an accepting state in $\mathscr{A}$, add a transition to w under some new label z, where $z \notin \Sigma$.
4) Add one more vertex u to $\mathscr{F}$, and a transition from v to u under z.
5) Color u blue, the all other vertices green.

Now, the interaction language for this filter is $\Sigma^* z$. Further, the outputs of strings $\mathcal{L}^A(\mathscr{A})z$ are both green and blue, while the outputs for the strings $(\Sigma^* \setminus \mathcal{L}^A(\mathscr{A}))z$ are blue only.

If $\mathcal{L}^A(\mathscr{A})$ is Σ^*, then the minimal filter for $\mathscr{F}$ has only one green state and it has a self loop bearing $\Sigma \cup \{z\}$. If $\mathcal{L}^A(\mathscr{A})$ is not Σ^*, then there there exists some string $s \notin \mathcal{L}^A(\mathscr{A})$ where s only outputs green, and sz only outputs blue. There must, therefore, be at least two states (one colored green, and one colored blue) in its minimizer. As a consequence, if the minimizer of $\mathscr{F}$ has only one state, then $\mathcal{L}^A(\mathscr{A})$ is Σ^*. Otherwise, $\mathcal{L}^A(\mathscr{A})$ is not Σ^*.

Therefore, we get a polynomial time reduction from NFA universality to PFM-DEC. PFM-DEC is PSPACE-hard since NFA universality is PSPACE-complete. □

Lemma 9. PFM-DEC is PSPACE-complete.

Proof. Combine Lemmas 7 and 8. □

Theorem 10. PFM is PSPACE-hard.

Proof. This is a direct consequence of Lemma 9. □

Having considered the case where both the input and the minimizer may be nondeterministic, next we show that limiting nondeterminism to only the input filter (what we dubbed PFDM-DEC earlier) still retains its hardness.

Theorem 11. PFDM-DEC is PSPACE-hard.

Proof. We show the PFDM-DEC is PSPACE-hard by reducing the DFA union universality problem (Lemma 5) to PFDM-DEC. Given a set of DFAs, $\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n$, let the union of their alphabet be Σ. The DFA union universality problem is to check $\mathcal{L}^A(\mathscr{A}_1) \cup \mathcal{L}^A(\mathscr{A}_2) \cup \cdots \cup \mathcal{L}^A(\mathscr{A}_n) = \Sigma^*$. For each DFA $\mathscr{A}_i$, we first, we construct a DFA $\mathscr{A}'_i$, such that $\mathcal{L}^A(\mathscr{A}'_i) = \mathcal{L}^A(\mathscr{A}_i)$ and $\mathcal{L}^I(\mathscr{A}'_i) = \Sigma^*$:

1) Initialize $\mathscr{A}'_i$ as a copy of $\mathscr{A}_i$.
2) If $\mathcal{L}^I(\mathscr{A}'_i) \neq \Sigma^*$, then add a trap state v' with a self loop bearing all labels in Σ for each DFA $\mathscr{A}'_i$. For each state w' in $\mathscr{A}'_i$ and every outgoing event $y \in \Sigma$, if y crashes when traced from w', build a transition from w' in $\mathscr{A}'_i$ to the trap state v' under y.
3) Make all states corresponding to accepting states in $\mathscr{A}_i$ the accepting states for $\mathscr{A}'_i$.

Next, we build an NFA $\mathscr{B}'$ as the union of all these $\mathscr{A}'_i$'s, so as to have $\mathcal{L}^A(\mathscr{A}_1) \cup \mathcal{L}^A(\mathscr{A}_2) \cup \cdots \cup \mathcal{L}^A(\mathscr{A}_n) = \mathcal{L}^A(\mathscr{B}')$. Additionally, no strings in Σ^* crash on $\mathscr{B}'$. The task, then, is to check whether $\mathcal{L}^A(\mathscr{B}') = \Sigma^*$ holds or not.

To do so, create a filter $\mathscr{F}$ from $\mathscr{B}'$ as follows:

1) Add the initial states, states, transitions of $\mathscr{B}'$ to $\mathscr{F}$.
2) Color the copies of the accepting states in $\mathscr{B}'$ green, and the copies of the non-accepting states red.
3) Add one more state, and color it green. Make this state the destination reached from one goal state under a fresh symbol z (i.e., where z is not a symbol from Σ).

By adding the new symbol z, we known that there is some string ending with z which outputs only green in $\mathscr{F}$.

Supposing $\mathscr{H}$ is a deterministic minimizer of $\mathscr{F}$, there are two cases. First, if $\mathscr{H}$ is a one-state filter, then it must be green because there are some strings that must output only green. Then every string in Σ^* must output at least green in $\mathscr{F}$. Hence, every string in Σ^* must reach the accepting states in $\mathscr{B}'$, and we conclude $\mathcal{L}^A(\mathscr{B}') = \Sigma^*$. Alternatively, if $\mathscr{H}$ has more than one state, then there is at least one green state and one red state. (Otherwise, $\mathscr{H}$ is not minimal.) Hence, there must be some string in Σ^* that can only output red. Those strings with only red output never reach the accepting states in $\mathscr{B}'$. So, consequently, $\mathcal{L}^A(\mathscr{B}') \neq \Sigma^*$.

The procedure to solve PFDM-DEC involves checking whether there is a one-state minimizer for $\mathscr{F}$. If there is such a minimizer, then the accepting language of the union of all DFAs is Σ^*. Otherwise, it is not. Having given a polynomial time procedure to reduce the DFA union universality problem to PFDM-DEC, which is itself known to be PSPACE-complete, shows that PFDM-DEC is PSPACE-hard. □

Since PFDM can be no easier than its decision version, PFDM is PSPACE-hard in terms of space complexity.

Theorem 12. PFDM is PSPACE-hard.

B. Is PFDM-DEC *PSPACE-complete?*

It seems natural to suppose that PFDM-DEC is simpler than PFM-DEC and should also be in PSPACE since the problem is narrower, focusing on more constrained (deterministic) minimizers. However, from the perspective of space consumption, this needn't be the case.

To help elucidate, it's useful to introduce some lightweight notation for the problems showing their inputs and outputs explicitly. We denote a minimization problem that converts something of type A to a corresponding minimal instance of type B as $\text{A}\xrightarrow{\text{min}}\text{B}$. We will compare and contrast filters with automata, and write deterministic and nondeterministic instances as **Det** and **NDet** respectively.

	$\mathbf{Det}(\mathscr{F}) \xrightarrow{\min} \mathbf{Det}(\mathscr{F}')$	$\mathbf{NDet}(\mathscr{F}) \xrightarrow{\min} \mathbf{NDet}(\mathscr{F}')$	$\mathbf{NDet}(\mathscr{F}) \xrightarrow{\text{'min'}} \mathbf{Det}(\mathscr{F}')$
Automata:	$\lvert V(\mathscr{F})\rvert \geq \lvert V(\mathscr{F}')\rvert$	$\lvert V(\mathscr{F})\rvert \geq \lvert V(\mathscr{F}')\rvert$	$\mathcal{F}'$ can be exponentially larger than $\mathcal{F}$. ([17])
Filters:	$\lvert V(\mathscr{F})\rvert \geq \lvert V(\mathscr{F}')\rvert$ (Figure 2 in [8])	$\lvert V(\mathscr{F})\rvert \geq \lvert V(\mathscr{F}')\rvert$ (Figure 2–3)	$\lvert V(\mathcal{F}')\rvert$ can be larger than any polynomial of $\lvert V(\mathcal{F})\rvert$.

Fig. 4: An overview about the size of the minimizer for both automata minimization and filter minimization problems.

Fig. 4 gives an overview about how the size may change during the process of filter minimization and automata minimization. From results known in the literature (and the previous figures in this paper), we know that for $\mathbf{Det}(\mathscr{F}) \xrightarrow{\min} \mathbf{Det}(\mathscr{F}')$ and $\mathbf{NDet}(\mathscr{F}) \xrightarrow{\min} \mathbf{NDet}(\mathscr{F}')$ in both filter minimization and automata minimization, the resulting minimal object is always smaller than (or equal to) the size of the object provided as input. But this need not be true when turning a $\mathbf{NDet}(\mathscr{F})$ element into a $\mathbf{Det}(\mathscr{F}')$ one, as the minimizer is constrained to be deterministic and it can be larger than the nondeterminsitic input. In particular, for automata minimization, the 'minimizer' (DFA) can be exponentially larger than the input automata (NFA) [14]. For filter minimization, take the problem shown in Fig. 3, and exchange the roles of the two graphs: the filter in Fig. 3a is a deterministic minimizer for the nondeterministic filter shown in Fig. 3b. There, the deterministic minimizer has one more state than its nondeterministic input filter.

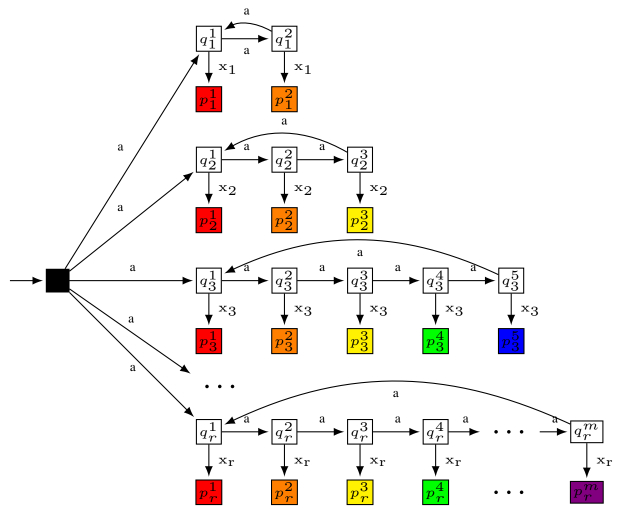
How big can the difference actually be? We give a construction for a family of filters demonstrating that the size of the deterministic minimizer may grow so that its size is beyond any polynomial in the input size (highlighted as blue in Fig. 4). First, we make a nondeterministic input filter, then we follow that by giving its deterministic minimizer.

Construction 13. Fix some natural number r, and construct the nondeterministic input filter with r rows depicted in Fig. 5a. Create a cycle of white states under 'a' where the number of white states at row $i \in \mathbb{N}^+$ is the cycle of length p_i, the i^{th} prime number. For example, the number of white states in rows 1, 2, 3 are 2, 3, 5, respectively. Create a black initial state that connects, via 'a', to one state at each of these r rows. At each row, starting from the state connected with the initial one, add a transition to a new child state. Color the child with a color from the color list $[o_1, o_2, \ldots, o_{p_r}]$, that excludes both black and white. Each child state is colored as the first one that is not chosen in the row.

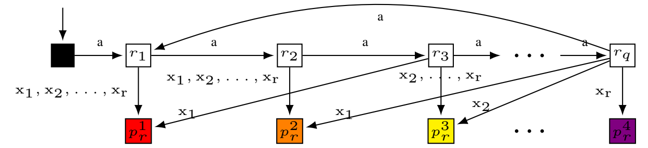
An equivalent deterministic filter, shown in Fig. 5b, is produced via the power set construction. Notice that all states in the nondeterministic filter of Fig. 5a reached by a common string share the same color, so there are no choices for each state in the deterministic filter: every state must be colored to correspond. (Part of the next lemma will show this to be the minimizer.)

Lemma 14. The deterministic minimizer of a nondeterministic input filter can exceed any polynomial of the input size.

Proof. First we argue that the deterministic form of the nondeterministic input filter from Construction 13 is a deterministic minimizer, and then show that the gap between the size of the nondeterministic input filter and its deterministic minimizer is larger than any polynomial of the input size.



(a) A nondeterministic filter with n rows, where the number of white states at the i^{th} row is the i^{th} prime number.



(b) A minimal deterministic filter for (a), where the total number of white states is the product of first n prime numbers.

Fig. 5: An example to show that the number of states in the deterministic minimizer is larger than polynomial size of the nondeterministic input filter.

The deterministic filter shown in Fig. 5b is already a minimal one for the filter depicted in Fig. 5a. The n colors must be included as they are each produced by some string; the white vertices could only be merged if there was a common divisor in the cycle lengths, but the cycle lengths are all distinct primes. Hence, no pair of states in Fig. 5b can be merged since they either have different outputs or disagree on the outputs of their common extensions.

Let n be the total number of states in this nondeterministic input filter. Then we have $n = 2 \cdot S(r) + 1$, where $S(r) = \sum_{i=1}^{r} p_i$ is summation of the first r prime numbers, and Bach and Shallit [15] have shown that $S(r) \sim \frac{1}{2} r^2 \ln r$ holds asymptotically. When $1 < r$, $n = r^2 \ln r + 1 < r^3$.

Let z represent the total number of states in the deterministic filter. Then we have $z = 1 + P(r) + p_r$, where $P(r) = \prod_{i=1}^{r} p_i$ is the primorial, i.e., the product of the first r prime numbers. According to the prime number theorem and the first Chebyshev function, we have that $P(r) \sim e^{(1+o(1))r \log r}$ and $p_r \sim r \log r$ holds asymptotically [16]. Hence,

$$\begin{aligned} z &= 1 + P(r) + p_r \\ &= 1 + e^{(1+o(1))r \log r} + r \log r > e^r \quad \text{for large } r. \end{aligned}$$

Since $r > \sqrt[3]{n}$, we have $z > e^{\sqrt[3]{n}}$. So we write this lower bound of z as $f(n) = e^{\sqrt[3]{n}} = \sum_{m=0}^{\infty} \frac{n^{\frac{m}{3}}}{m!}$ (Taylor series).

Now consider any polynomial of n of degree k and write it as $g(n, k) = \sum_{m=0}^{k} \alpha_m n^m$. Let $c = \max\{\alpha_0, \alpha_1, \ldots, \alpha_k\}$.

If $n > c \cdot (k+1)$, then we have for all $i \leq k$, the coefficients have $\alpha_i n^i < c \cdot n^k$, and the sum $\sum_{i=0}^{i=k} \alpha_i n^i < n^{k+1}$.

To bring the two bounds in relation to one another: when $n > (3k+6)!$, then $f(n) > \frac{n^{\frac{3k+6}{3}}}{(3k+6)!} > \frac{n^{\frac{3k+6}{3}}}{n} = n^{k+1}$, Hence, $f(n) > n^{k+1}$ if $n > (3k+6)!$. Thus for $n > \max\{c \cdot (k+1), (3k+6)!\}$, we have that $z > f(n) > n^{k+1} > g(n, k)$. This is true for any k, so the size of the deterministic minimizer, z, is larger than any polynomial of n. □

One implication of the preceding example is that:

Lemma 15. PFDM is not in P.

Proof. Since the size of the minimizer can be larger than any polynomial of the input size, it takes more than polynomial time to output the minimizer. Therefore, PFDM $\notin$ P. □

Then, considering time complexity further, we can conclude that PFDM is strictly NP-hard.

Theorem 16. PFDM is NP-hard, but not in P.

Proof. The deterministic input to deterministic output filter minimization problem, the decision problem form of which is NP-complete [5], is properly contained in PFDM (one just happens to select an input that is deterministic). We have that PFDM is NP-hard, and combining with Lemma 15, we can conclude that PFDM is strictly NP-hard. □

To summarize, Construction 13 and Lemma 14 show that the gap between the size of the deterministic minimizer can be larger than polynomial of the input size. It indicates that constructing and storing the deterministic minimizer in its entirety to determine its size would disqualify PFDM-DEC from PSPACE. Of course, other cleverer means may exist, so whether PFDM-DEC is PSPACE (as a consequence, PFDM-DEC is PSPACE-complete) or not remains an open question.

VI. A Comparison between Automata Minimization and Filter Minimization

With the preceding hardness results for filter minimization problems established, we now compare them with the hardness of automata minimization in Fig. 6. It is worthwhile to try distill intuition for a couple of reasons: firstly, the automata hardness results were used in the arguments above, so their connection might seem obvious at first blush. But the notion of equivalence beween two automata is quite different from that beween two filters, as, importantly, are specific requirements on interaction vs. accepting languages. Secondly, recall that the initial supposition that deterministic filter and deterministic automata minimization problems were identical, was wrong.

In the first column of Fig. 6 ($\mathbf{Det}(\mathscr{F}) \xrightarrow{\text{min}} \mathbf{Det}(\mathscr{F}')$): automata minimization problem ($\mathscr{F}$ is a DFA) can be solved efficiently by identifying Myhill–Nerode equivalence classes [17], while the decision version of filter minimization problem ($\mathscr{F}$ is a filter) is NP-complete. The main reason for this hardness separation between these two problems is the extra degree of freedom (DOF) for filter minimization. Filters can choose to assign *any* output for the strings that

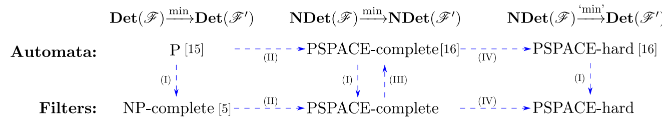


Fig. 6: A comparison between hardness results of decision versions of automata minimization and filter minimization.

crash (informally, we call this DOF I). To exploit this degree of freedom optimally, it is equivalent to searching for a minimum clique cover in the compatibility graph of the input filter [8], which makes the problem computationally hard.

For the other two columns of Fig. 6: As we consider nondeterminism in the input or output object, the hardness separation between automata minimization and filter minimization disappears. Informally speaking, it appears that the hardness arising from DOF I is dominated by other sources of complexity. When nondeterminism appears in both input and output, i.e., $\mathbf{NDet}(\mathscr{F}) \xrightarrow{\text{min}} \mathbf{NDet}(\mathscr{F}')$, the decision problems of both filter minimization and automata minimization are PSPACE-complete [18]. For both, there could be multiple states simultaneously reached by the same string (DOF II) though it takes no more than polynomial space to check the outputs of those states. Though both are PSPACE-complete, the problems differ in the degrees of freedom they have—though, clearly, this difference is not enough to manifest as a hardness gap. On the one hand, nondeterministic filter minimization ($\mathscr{F}$ is a nondeterministic filter) has DOF I while nondeterminsitic automata minimization ($\mathscr{F}$ is an NFA) does not. On the other, non-deterministic filter minimization requires all outputs of all states reached by the string be constrained, whereas nondeterministic automata minimization can choose to accept the states or not, as long as at least one is accepted (DOF III),

If we keep nondeterminism in the inputs but remove it from the outputs, the problems $\mathbf{NDet}(\mathscr{F}) \xrightarrow{\text{`min'}} \mathbf{Det}(\mathscr{F}')$ do not become any easier. When outputs are restricted to be deterministic, the size of the output can be substantially larger than that of the input filter, (IV). If one were to think of this as a search problem, a more restrictive type can drastically increase the search space size. It only ever takes polynomial space for $\mathbf{Det}(\mathscr{F}) \xrightarrow{\text{min}} \mathbf{Det}(\mathscr{F}')$, but it is unclear whether this holds for $\mathbf{NDet}(\mathscr{F}) \xrightarrow{\text{`min'}} \mathbf{Det}(\mathscr{F}')$, and the increase in output size is unfavourable (though inconclusive) evidence to the contrary.

VII. Conclusion

This paper shows the value of nondeterminism in combinatorial filter reduction, analyzes the hardness of nondeterministic filter minimization problems, and shows that the hardness separation between deterministic filter minimization problems disappears in the nondeterministic cases.

Future work might consider the hardness results for finding nondeterministic minimizers for deterministic input filters, which is only known to be in PSPACE. Another direction is to examine complete, approximation, and heuristic algorithms to solve nondeterministic filter minimization.

REFERENCES

[1] B. R. Donald, "On information invariants in robotics," *Artificial Intelligence*, vol. 72, pp. 217–304, 1995.

[2] S. M. LaValle, "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.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* MIT Press, 2005.

[4] B. Tovar, F. Cohen, L. Bobadilla, J. Czarnowski, and S. M. Lavalle, "Combinatorial filters: Sensor beams, obstacles, and possible paths," *ACM Transactions on Sensor Networks*, vol. 10, no. 3, pp. 1–32, 2014.

[5] J. M. O'Kane and D. A. Shell, "Concise planning and filtering: hardness and algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 4, pp. 1666–1681, 2017.

[6] F. Z. Saberifar, A. Mohades, M. Razzazi, and J. M. O'Kane, "Combinatorial filter reduction: Special cases, approximation, and fixed-parameter tractability," *Journal of Computer and System Sciences*, vol. 85, pp. 74–92, 2017.

[7] H. Rahmani and J. M. O'Kane, "Integer linear programming formulations of the filter partitioning minimization problem," *Journal of Combinatorial Optimization*, vol. 40, pp. 431–453, 2020.

[8] Y. Zhang and D. A. Shell, "Cover combinatorial filters and their minimization problem," in *Algorithmic Foundations of Robotics XIV*, S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu, Eds. Cham: Springer International Publishing, 2021, pp. 90–106.

[9] Y. Zhang, H. Rahmani, D. A. Shell, and J. M. O'Kane, "Accelerating combinatorial filter reduction through constraints," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2021.

[10] F. Z. Saberifar, S. Ghasemlou, J. M. O'Kane, and D. A. Shell, "Set-labelled filters and sensor transformations," in *Robotics: Science and Systems*, 2016.

[11] L. Stockmeyer and A. Meyer, "Word problems requiring exponential time: preliminary report," in *Symposium on Theory of Computing*, 1973, pp. 1–9.

[12] A. R. Meyer and L. J. Stockmeyer, "The equivalence problem for regular expressions with squaring requires exponential space," in *Symposium on Switching and Automata Theory*, 1972, pp. 125–129.

[13] N. Rampersad, J. Shallit, and Z. Xu, "The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages," *Fundamenta Informaticae*, vol. 116, no. 1-4, pp. 223–236, 2012.

[14] A. R. Meyer and M. J. Fischer, "Economy of description by automata, grammars, and formal systems," in *Symposium on Switching and Automata Theory*, 1971, pp. 188–191.

[15] E. Bach, J. O. Shallit, J. Shallit, and S. Jeffrey, *Algorithmic number theory: Efficient algorithms*. MIT press, 1996, vol. 1.

[16] G. J. O. Jameson, *The prime number theorem*. Cambridge University Press, 2003.

[17] B. Mikolajczak, *Algebraic and structural automata theory*, ser. Annals of Discrete Mathematics. Elsevier Science Inc., 1991.

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