Cover Combinatorial Filters and their Minimization Problem

Yulin Zhang and Dylan A. Shell

Department of Computer Science & Engineering,
Texas A&M University, College Station TX 77843, USA
yulinzhang|dshell@tamu.edu

Abstract. Recent research has examined algorithms to minimize robots' resource footprints. The class of combinatorial filters (discrete variants of widely-used probabilistic estimators) has been studied and methods for reducing their space requirements introduced. This paper extends existing combinatorial filters by introducing a natural generalization: cover combinatorial filters. In addressing the new—but still NP-complete—problem of minimization of cover filters, we show that multiple concepts previously believed about combinatorial filters (and actually conjectured, claimed, or assumed to be) are in fact false. For instance, minimization does not induce an equivalence relation. We give an exact algorithm for the cover filter minimization problem. Unlike prior work (based on graph coloring) we consider a type of clique-cover problem, involving a new conditional constraint, from which we can find more general relations. In addition to solving the more general problem, the algorithm also corrects flaws present in all prior filter reduction methods. In employing SAT, the algorithm provides a promising basis for future practical development.

1 Introduction

As part of the long history of research in robotic minimalism, a recent thread has devised methods that aim to automatically reduce and reason about robots' resource footprints. That work fits within the larger context of methodologies and formalisms for tackling robot design problems, being useful for designing robots subject to resource limits [1,2,3]. But, more fundamentally, the associated algorithms also help identify the information requirements of certain robot tasks. The methods have the potential to provide insights about the interplay of sensing, state, and actuation within the context of particular tasks. One class of objects where the problem of resource minimization can be clearly posed is in the case of combinatorial filters [4]. These are discrete variants of the probabilistic estimators and recursive Bayesian filters widely adopted for practical use in robots. Combinatorial filters process a stream of discrete sensor inputs and integrate information via transitions between states. The natural question, studied in [5], then is: *How few states are needed to realize specified filter functionality?* In this paper, we define a more general class of filters and ask the same question.

This work was supported by the NSF through awards IIS-1453652 and IIS-1527436.

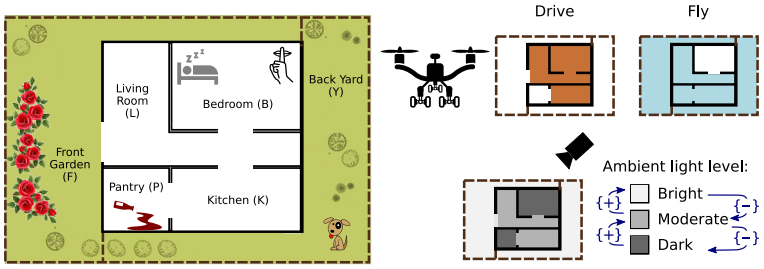
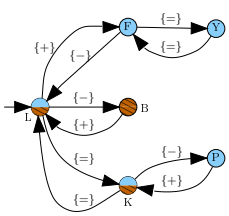
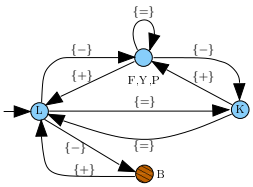


Fig. 1: A hybrid drone with a light sensor monitors a home. The robot is capable of driving or flying: the grass outdoors (F and Y) and liquids (P) necessitate that it be airborne; the noise requirement means it may only drive within the bedroom (B). It has a light sensor that distinguishes three levels of ambient brightness. Changes in brightness (increasing '+', decreasing '−', same '=') provide the robot cues about its location.

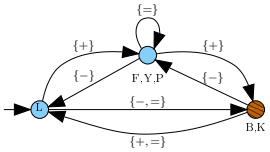
We start with a simple motivating scenario where the generalization we introduce is exactly what is needed. Figure 1 shows a driving drone patrolling a house.[1] The drone can either drive or fly, but its choice must satisfy navigability constraints. Its wheels can't drive on grass (F and Y) nor in the pantry (P), owing to spills. Spinning propellers, on the other hand, will disturb the tranquil bedroom (B). Otherwise, either means may be chosen (see inset map pair marking regions in brown/blue for driving/flying). The robot is equipped with an ambient light sensor that is useful because the living room and kitchen are lighter than the bedroom and pantry, while the outdoors is lightest of all.



(a) A naïve filter, with one state per region, codifies all valid choices.



(b) A minimal filter when choosing to fly in the living room (L) and kitchen (K).



(c) A minimal filter when opting to fly in the living room (L) but not the kitchen (K).

Fig. 2: Combinatorial filters that tell the hybrid drone how to locomote. The sequence of symbols {+,−,=} is traced on the graph, and the color of the resultant state is the filter's output (blue for flight, brown for driving mode).

We wish to construct a filter for the drone to determine how to navigate, with the inputs being brightness changes, and the filter's output providing some valid mode of locomotion. It is easy to give a valid filter by using one state for each location — this naïve filter is depicted in Figure 2a. In the living room and kitchen, the filter lists two outputs since both modes are applicable there (both

[1] Such bizarre chimera robots are not our invention, e.g., see the Syma X9 Flying Car.

locations are covered by the brown and blue choices). Now consider the question of the smallest filter. If we opt to fly in both the living room and kitchen, then the smallest filter is shown in Figure 2b with 4 states. But when choosing to fly in the living room but drive in the kitchen, the minimal filter requires only 3 states (in Figure 2c).

This last filter is also the globally minimal filter. The crux is that states with multiple valid outputs introduce a new degree of freedom which influences the size of the minimal filter. These arise, for instance, whenever there are 'don't-care' options. The flexibility of such states must be retained to truly minimize the number of states.

2 Preliminary Definitions and Problem Description

To begin, we define the filter minimization problem in the most general form, where the input is allowed to be non-deterministic and each state may have multiple outputs. This is captured by the procrustean filter (p-filter) formalism [6].

2.1 P-filters and their minimization

We firstly introduce the notion of p-filter:

Definition 1 (procrustean filter [6])**.** A *procrustean filter*, *p-filter* or *filter* for short, is a tuple (V, V_0, Y, τ, C, c) with:

1) a finite set of states V, a non-empty initial set of states $V_0 \subseteq V$, and a set of possible observations Y,
2) a transition function $\tau : V \times V \to 2^Y$,
3) a set C, which we call the output space, and
4) an output function $c : V \to 2^C \setminus \{\varnothing\}$.

The states, initial states and observations for p-filter F will be denoted $V(F)$, $V_0(F)$ and $Y(F)$. Without loss of generality, we will also treat a p-filter as a graph with states as its vertices and transitions as directed edges.

A sequence of observations can be traced on the p-filter:

Definition 2 (reached)**.** Given any p-filter $F = (V, V_0, Y, \tau, C, c)$, a sequence of observations $s = y_1 \dots y_n \in Y^*$, and states $w_0, w_n \in V$, we say that w_n is a state *reached by* some sequence s from w_0 in F (or s reaches w_n from w_0), if there exists a sequence of states $w_0, \dots, w_n$ in F, such that $\forall i \in \{1, \dots, n\}, y_i \in \tau(w_{i-1}, w_i)$. We denote the set of all states reached by s from state w_0 in F as $\mathcal{V}_{w_0}(F, s)$. For simplicity, we use $\mathcal{V}(F, s)$, without the subscript, to denote the set of all states reached when starting from any state in V_0, i.e., $\mathcal{V}(F, s) = \cup_{v_0 \in V_0} \mathcal{V}_{v_0}(F, s)$. Note that $\mathcal{V}(F, s) = \varnothing$ holds only when sequence s *crashes* in F starting from V_0.

For convenience, we will denote the set of sequences reaching state $v \in V$ from some initial state by $\mathcal{S}_v^F$.

Definition 3 (extensions, executions and interaction language)**.** An *extension* of a state v on a p-filter F is a finite sequence of observations s that does not crash when traced from v, i.e., $\mathcal{V}_v(F, s) \neq \varnothing$. An *extension* of any initial state

$v_0 \in V_0(F)$ is also called an *execution* or a *string* on F. The set of all extensions of a state v on F is called the *extensions* of v, written as $\mathcal{L}_F(v)$. The extensions of all initial vertices on F is also called the *interaction language* (or, briefly, just *language*) of F, and is written $\mathcal{L}(F) = \cup_{v_0 \in V_0(F)} \mathcal{L}_F(v_0)$.

Note in particular that the empty string ϵ belongs to the extensions of any state on the filter, and belongs to the language of the filter as well.

Definition 4 (filter output)**.** Given any p-filter $F = (V, V_0, Y, \tau, C, c)$, a string s and an output $o \in C$, we say that o is a *filter output* with input string s, if o is an output from the state reached by s, i.e., $o \in \cup_{v \in \mathcal{V}(F,s)} c(v)$. We denote the set of all filter outputs for string s as $\mathcal{C}(F, s) = \cup_{v \in \mathcal{V}(F,s)} c(v)$.

Specifically, for the empty string ϵ, we have $\mathcal{C}(F, \epsilon) = \cup_{v_0 \in V_0(F)} c(v_0)$.

Definition 5 (output simulating)**.** Given any p-filter F, a p-filter F' *output simulates* F if $\forall s \in \mathcal{L}(F)$, $\mathcal{C}(F', s) \neq \varnothing$ and $\mathcal{C}(F', s) \subseteq \mathcal{C}(F, s)$.

Plainly in words: for one p-filter to output simulate another, it has to generate some of the outputs of the other, for every string the other admits.

We are interested in practicable p-filters with deterministic behavior:

Definition 6 (deterministic)**.** A p-filter $F = (V, V_0, Y, \tau, C, c)$ is *deterministic* or *state-determined*, if $|V_0| = 1$, and for every $v_1, v_2, v_3 \in V$ with $v_2 \neq v_3$, $\tau(v_1, v_2) \cap \tau(v_1, v_3) = \varnothing$.

Any non-deterministic p-filter F can be state-determinized, denoted $\textsc{Sde}(F)$, following Algorithm 2 in [7].

Then the p-filter minimization problem can be formalized as follows:

Problem: P-filter Minimization (PFM)

Input: A deterministic p-filter F.

Output: A deterministic p-filter $F^\dagger$ with fewest states, such that $F^\dagger$ output simulates F.

'PF' denotes that the input is a p-filter, and 'M' denotes that we are interested in finding a deterministic minimal filter as a solution.

2.2 Complexity of p-filter minimization problems

Definition 7 (state single-outputting and multi-outputting)**.** A p-filter $F = (V, V_0, Y, \tau, C, c)$ is *state single-outputting* or *single-outputting* for short, if c only maps to singletons, i.e., $|c(v)| = 1, \forall v \in V(F)$. Otherwise, we say that F is *multi-outputting*.

Depending on whether the state in the input p-filter (PF) is single-outputting (SO) or multi-outputting (MO), we further categorize the problem PFM into the following problems: SO-FM and MO-FM.

Lemma 8. The filter minimization problem FM of [5] is SO-FM, and SO-FM is NP-Complete (Theorem 2 in [5]).

Theorem 9. MO-FM is NP-Complete.

Proof. Firstly, SO-FM is a special case of MO-FM problems. These MO-FM problems are at least as hard as SO-FM. Hence, MO-FM are in NP-hard. On the other hand, a solution for MO-FM can be verified in polynomial time. (Change the equality check on line 7 of Algorithm 1 in [5] to a subset check.) Therefore, MO-FM is NP-Complete. □

Next, we examine related prior work on SO-FM closely as a means to develop new insights for our algorithms, first for SO-FM (Section 4), and then MO-FM (Section 5).

3 Related work: Prior filter minimization ideas (SO-FM)

Several elements come together in this section and Figure 3 attempts to show the inter-relationships graphically. The original question of minimizing state in filtering is first alluded to by LaValle [4] as an open problem, who suggested that it is 'similar to Nerode equivalence classes'. The problem of filter reduction, i.e., SO-FM in our terms, was formalized and shown to differ in complexity class from the automata problem in [5]. That paper also proposed a heuristic algorithm, which served as a starting point for subsequent work. The heuristic algorithm uses conflict graphs to designate which vertices cannot be merged (are conflicting). It starts with a conflict relation where two vertices are in conflict when they have different outputs, then iteratively refines the conflict relation. Refinement has two steps: (*i*) *introducing edges*: two vertices are determined to be conflicting or not via a graph coloring subroutine, and edges are added between conflicting vertices; (*ii*) *propagating conflicts upstream*: filter states are marked as conflicted when they transition to conflicted states under the same observation. An example input filter, shown in Figure 4a, is reduced by following this procedure, which is depicted step-by-step in Figures 4b–4e.

A conjecture in [5] was that this algorithm is guaranteed to find a minimal filter if the graph coloring subroutine gives a minimal coloring. (Put another way: the inexactness in arriving at a minimal filter can be traced to the graph coloring giving a suboptimal result.) But this conjecture was later proved to be false by Saberifar et al. [8]. They show an instance where there exist multiple distinct optimal solutions to the graph coloring subproblem, only a strict subset of which lead to the minimal filter. One might naturally ask, and indeed they do ask, the question of whether some optimal coloring is sufficient to arrive at the optimal filter. Following along these lines (see §7.3 in [8]), one might sharpen the original conjecture of [5] to give the following statement:

Idea 1. In the step-wise conflict refinement procedure of O'Kane and Shell's heuristic algorithm [5], some optimal coloring is sufficient to guarantee a minimal filter for SO-FM.

Lemma 10. Idea 1 is false.

Proof. This is simply shown with a counterexample. Consider the problem of minimizing the input filter shown in Figure 4a, the heuristic algorithm will first

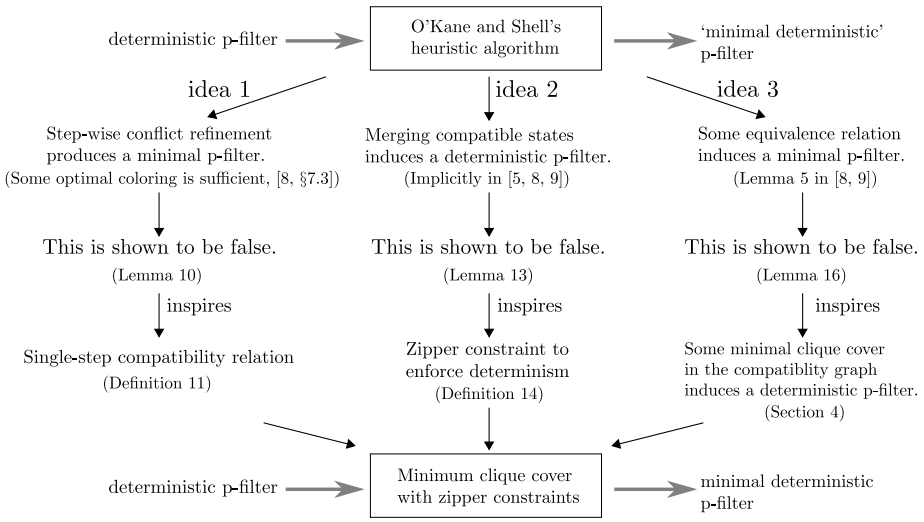


Fig. 3: Three distinct insights led to the development of a new algorithm (described in Section 4) for SO-FM. This roadmap shows the provenance of those insights in terms of previous ideas in SO-FM, which we examine carefully.

initialize the colors of the vertices with their output. Next, it identifies the vertices that disagree on the outputs of extensions with length 1 as shown in Figure 4b, and then refines the colors of the vertices as shown in Figure 4c following a minimal graph coloring solution on the conflict graph. Then it further identifies the conflicts on extensions with length 2, via the conflict graph shown in Figure 4d, and the vertex colors are further refined as shown in Figure 4e. Now, no further conflicts can be found. A filter, with 6 states, is then obtained by merging the states with the same color. However, there exists a minimal filter, with 5 states, shown in Figure 4f, that can be found by choosing coloring solution for the conflict graph shown in Figure 4b. That coloring is suboptimal. □

This appears to indicate a sort of *local optimum* arising via sub-problems associated with incremental (or stepwise) reduction. Since optimal colorings for individual steps are seen to be insufficient to guarantee a minimal filter, to find a minimal filter, we would have to enumerate all colorings (suboptimal or otherwise) at each iteration. That is, however, essentially a brute force algorithm. A more informed approach is to compute implications of conflicts more *globally*, in a way that doesn't depend on earlier merger decisions. In our algorithm, rather than tracking vertices which are in conflict, we introduce a new notion of compatibility between vertices that may be merged. This notion differs from the one recursively defined in [9], as our compatibility relation is computed in one fell swoop, before making any decisions to reduce the filter:

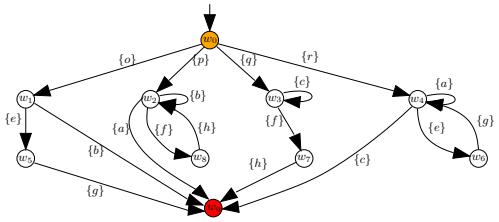
Definition 11 (compatibility)**.** Let F be a deterministic p-filter. We say a pair of vertices $v, w \in V(F)$ are *compatible*, denoted $v \sim_c w$, if they agree on the

outputs of all their extensions, i.e., $\forall s \in \mathcal{L}_F(v) \cap \mathcal{L}_F(w), \forall v' \in \mathcal{V}_v(F,s), \forall w' \in \mathcal{V}_w(F,s), c(v') = c(w')$. A *mutually compatible* set consists of vertices where all pairs are compatible.

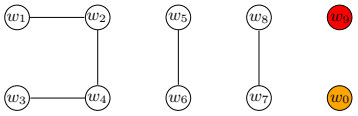
Via this notion of compatibility, we get an undirected compatibility graph:

Definition 12 (compatibility graph)**.** Given a deterministic filter F, its compatibility graph $\mathcal{K}(F)$ is an unlabeled undirected graph constructed by creating a vertex associated with each state in F, and building an edge between the pair of vertices associated with two compatible states.

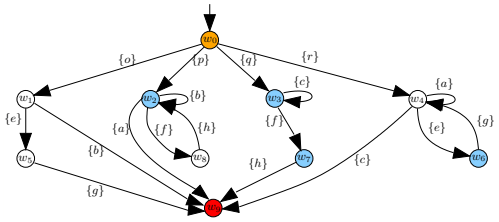
This compatibility graph can be constructed in polynomial time. As every filter state and associated compatibility graph state are one-to-one, to simplify notation we'll use the same symbol for both and context to resolve any ambiguity.



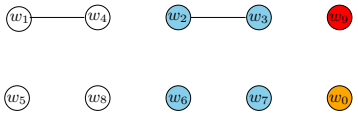
(a) Example input p-filter.



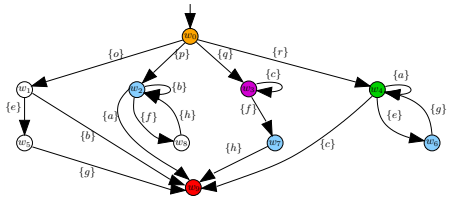
(b) Graphs of initial conflicts for all vertices with the same output.



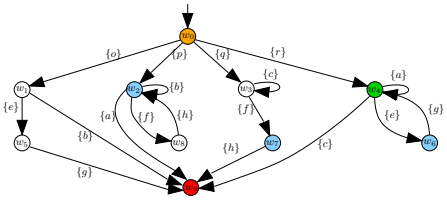
(c) The first refinement of the filter following an optimal coloring of the conflict graphs.



(d) Reduced conflict graphs.



(e) Second refinement of filter following an optimal coloring of the conflict graphs.



(f) The coloring that gives the minimal filter.

Fig. 4: An example run of the heuristic minimization algorithm in [5] (a)–(e). This particular input also shows that optimal step-wise conflict refinement may fail to yield a minimal filter (Lemma 10).

The second idea relates to the type of the output one obtains after merging states that are compatible or not in conflict. Importantly, the filter minimization problem SO-FM requires one to give a minimal filter which is deterministic.

Idea 2. By merging the states that are compatible, the heuristic algorithm always produces a deterministic p-filter.

The definition of the reduction problems within [5,8,9] are specified so as to require that the output obtained be deterministic. But this postcondition is never shown or formally established. In fact, it does not always hold.

Lemma 13. Idea 2 is false.

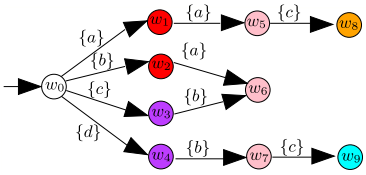
Proof. We show that the existing algorithm may produce a non-deterministic filter, which does not output simulate the input filter, and is thus not a valid solution. Consider the filter shown in Figure 5a as an input. The vertices with the same color are compatible with each other, with the following exception for w_5, w_6 and w_7. Vertex w_5 is compatible with w_6, vertex w_6 is compatible with w_7, but w_5 is not compatible with w_7. The minimal filter found by the existing algorithm is shown in Figure 5b. The string *aac* suffices to shows the non-determinism, reaching both orange and cyan vertices. It fails to output simulate the input because cyan should never be produced. □

If determinism can't be taken for granted, we might constrain the output to ensure the result will be a deterministic filter. To do this, we introduce a zipper constraint when merging compatible states:

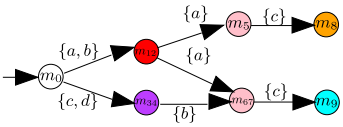
Definition 14 (zipper constraint)**.** In the compatibility graph $G = \mathcal{K}(F)$ of filter F, if there exists a set of mutually compatible states $U = \{u_1, u_2, \ldots, u_n\}$, then they can only be selected to be merged if they always transition to a set of states that are also selected to be merged. For any sets of mutually compatible states $U, W \subseteq V(G)$ and some observation y, we create a *zipper constraint* expressed as a pair $(U, W)_y$ if $W = \{w \in V(G) \mid y \in \tau(u, w) \text{ for some } u \in U\}$. We denote the set of all zipper constraints on compatibility graph $G = \mathcal{K}(F)$ by $\mathscr{Z}(F)$.

The zipper constraints for the input filter shown in Figure 5a consist of $(\{w_1, w_2\}, \{w_5, w_6\})_a$ and $(\{w_3, w_4\}, \{w_6, w_7\})_b$. Constraint $(\{w_1, w_2\}, \{w_5, w_6\})_a$ is interpreted as: if w_1 and w_2 are selected for merger, then w_5 and w_6 (reached under a) should also be merged. We call it a zipper constraint owing to the resemblance to a zipper fastener: merger of two earlier states, merges (i.e., pulls together) later states. In the worst case, the number of zipper constraints can be exponential in the size of the input filter.

A third idea is used by O'Kane and Shell's heuristic algorithm and is also stated, rather more explicitly, by Saberifar et al. (see Lemma 5 in [8] and Lemma 5 in [9]). It indicates that we can obtain a minimal filter via merging operations on the compatible states, which yields a special class of filter minimization problems. For this class, recent work has expoited integer linear programming techniques to compute exact and feasible solutions efficiently [10].



(a) An input filter.



(b) The non-deterministic minimal filter found by the existing algorithm.

Fig. 5: A counterexample showing how compatible merges may introduce non-determinism (Lemma 13). The input filter also illustrates a violation of the presumption that an equivalence relation can yield a minimum filter (Lemma 16).

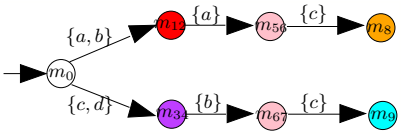
Idea 3. Some equivalence relation induces a minimal filter in SO-FM.

Before examining this, we rigorously define the notion of an induced relation:

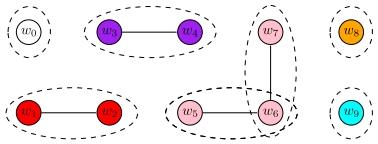
Definition 15 (induced relation)**.** Given a filter F and another filter F', if F' output simulates F, then F' induces a relation $R \subseteq V(F) \times V(F)$, where $(v, w) \in R$ if and only if there exists a vertex $v' \in V(F')$ such that $\mathcal{S}_v^F \cap \mathcal{S}_{v'}^{F'} \neq \varnothing$ and $\mathcal{S}_w^F \cap \mathcal{S}_{v'}^{F'} \neq \varnothing$. We also say that v and w corresponds to state v'.

Lemma 16. Idea 3 is false.

Proof. It is enough to scrutinize the previous counterexample closely. The minimization problem SO-FM for the input filter shown in Figure 5a, is shown in Figure 6a. It is obtained by (i) splitting vertex w_6 into an upper part reached by a and a lower part reached by b, (ii) merging the upper part of w_6 with w_5, the lower part of w_6 with w_7, and other vertices with those of the same color. This does not induce an equivalence relation, since w_6 corresponds to two different vertices in the minimal filter. □



(a) A minimal filter for Figure 5a.



(b) Cliques from the minimal filter.

Fig. 6: A minimal filter for Figure 5a and its induced cliques.

In light of this, for some filter minimization problems, there may be no quotient operation that produces a minimal filter and an exact algorithm for minimizing filters requires that we look beyond equivalence relations.

Some strings that reach a single state in an input filter may reach multiple states in a minimal p-filter (e.g., ba and cb on Figure 5a and 6a). On the other hand, strings that reach different states in the input p-filter may reach the same state in the minimal filter (e.g., a and b on those same filters). We say that a state from the input filter corresponds to a state in the minimal filter if there exists some string reaching both of them and, hence, this correspondence is many-to-many. An important observation is this: for each state s in some hypothetical minimal filter, suppose we collect all those states in the input filter that

correspond with s. When we examine the associated states in the compatibility graph for that collection, they must all form a clique. Were it not so, the minimal filter could have more than one output associated for some strings owing to non-determinism. But this causes it to fail to output simulate the input p-filter.

After firming up and developing these intuitions, the next section introduces the concept of a clique cover which enables representation of a search space that includes relations more general than equivalence relations. Based on this new representation, we propose a graph problem use of zipper constraints, and prove it to be equivalent to filter minimization.

4 A new graph problem that is equivalent to SO-FM

By building the correspondence between the input p-filter in Figure 5a and the minimal result in Figure 6a, one obtains the set of cliques in the compatibility graph shown visually in Figure 6b. Like previous approaches that make state merges by analyzing the compatibility graph, we interpret each clique as a set of states to be merged into one state in the minimal filter. The clique containing w_3 and w_4 in Figure 6b gives rise to m_{34} in the minimal filter in Figure 6a (and w_1 and w_2 yields m_{12}, and so on). However, states may further be shared across multiple cliques. We observe that w_6 was merged with w_5 in the minimal filter to give m_{56}, and w_6 also merged with w_7 to give m_{67}. The former has an incoming edge labeled with an a, while the latter has an incoming edge labeled b. The vertex w_6, being shared by multiple cliques, is split into different copies and each copy merged separately.

Generalizing this observation, we turn to searching for the smallest set of cliques that cover all vertices in the compatibility graph. Further, to guarantee that the set of cliques induces a deterministic filter, we must ensure they respect the zipper constraints. It will turn out that a solution of this new constrained minimum clique cover problem always induces a minimal filter for SO-FM, and a minimal filter for SO-FM always induces a solution for this new problem. The final step is to reduce any MCCZC problem to a SAT instance, and leverage SAT solvers to find a minimal filter for SO-FM.

4.1 A new minimum clique cover problem

To begin, we extend the preceding argument from the compatibility clique associated to single state s, over to all the states in the minimal filter. This leads one to observe that the collection of all cliques for each state in the minimal p-filter forms a clique cover:

Definition 17 (induced clique cover)**.** Given a p-filter F and another p-filter F', we say that a vertex v in F *corresponds to* a vertex v'_i in F' if $\mathcal{S}_v^F \cap \mathcal{S}_{v'_i}^{F'} \neq \varnothing$. Then, denoting the subset of vertices of F corresponding to v'_i in F' with $K_{v'_i} = \{v \in V(F) \,|\, v \text{ corresponds to } v'_i\}$, we form the collection of all such sets, $\mathbf{Q}(F, F') = \{K_{v'_1}, K_{v'_2}, \ldots, K_{v'_n}\}$, for $i \in \{1, \ldots, n\}$ where $n = |V(F')|$. When F' output simulates F, then the $K_{v'_i}$ form cliques in the compatibility graph $\mathcal{K}(F)$. Further, when this collection of sets $\mathbf{Q}(F, F')$ covers all vertices in F, i.e., $\cup_{K_i \in \mathbf{Q}(F,F')} = V(F)$, we say that $\mathbf{Q}(F, F')$ is an induced clique cover.

It is worth repeating: the size of filter F' (in terms of number of vertices) and the size of the induced clique cover (number of sets) are equal.

Without loss of generality, here and henceforth we only consider the p-filter with all vertices reachable from the initial state, since the ones that can never be reached will be deleted during filter minimization anyway.

Each clique of the clique cover represents the states that can be potentially merged. But the zipper constraint, to enforce determinism, requires that the set of vertices to be merged should always transition under the same observation to the ones that can also be merged. Hence, the zipper constraints (of Definition 14) can be evaluated across whole covers:

Definition 18. A clique cover $\mathbf{K} = \{K_0, K_1, \ldots, K_m\}$ satisfies the set of zipper constraints $\textsc{Zip} = \{(U_1, W_1)_{y_1}, (U_2, W_2)_{y_2}, \ldots\}$, when for every zipper constraint $(U_i, W_i)_{y_i}$, if there exist a clique $K_s \in \mathbf{K}$, such that $U_i \subseteq K_s$, then there exists another clique $K_t \in \mathbf{K}$ such that $W_i \subseteq K_t$.

Now, we have our new graph problem, MCCZC.

Problem: Minimum clique cover with zipper constraints (MCCZC)

Input: A compatibility graph G, a set of zipper constraints Zip.
Output: A minimal cardinality clique cover of G satisfying Zip.

4.2 From minimal clique covers to filters

Given a minimal cover that solves MCCZC, we construct a filter by merging the states in the same clique and choosing edges between these cliques appropriately:

Definition 19 (induced filter)**.** Given a clique cover $\mathbf{K}$ on the compatibility graph of deterministic p-filter F, if $\mathbf{K}$ satisfies all the zipper constraints in $\mathscr{Z}(F)$, then it *induces a filter* $F' = M(F, \mathbf{K})$ by treating cliques as vertices:

1. Create a new filter $F' = (V', V_0', Y, \tau', C, c')$ with $|\mathbf{K}|$ vertices, where each vertex v' is associated with a clique $K_{v'}$ in $\mathbf{K}$;
2. Add each vertex v' in F' to V_0' iff the associated clique contains an initial state in F;
3. The output of every v' in F', with associated clique $K_{v'}$, is the set of common outputs for all states in $K_{v'}$, i.e., $c'(v') = \cap_{v \in K_{v'}} c(v)$.
4. For any pair of v' and w' in F', inherit all transitions between states in the cliques of v' and w', i.e., $\tau(v', w') = \cup_{v \in K_{v'}, w \in K_{w'}} \tau(v, w)$.
5. For each vertex v' in F' with multiple outgoing edges labeled y, keep only the single edge to the vertex w', such that all vertices $K_{v'}$ transition to under y are included in $K_{w'}$. This edge must exist since $\mathbf{K}$ satisfies all $\mathscr{Z}(F)$.

The size of the cover (in terms of number of sets) and size of the induced filter (number of vertices) are equal.

Notice that the earlier intuition is mirrored by this formal construction: states belonging to the same clique are merged when constructing the induced filter; states in multiple cliques are split when we make the edge choice in step 5. Next, we establish that the induced filter indeed supplies the goods:

Lemma 20. Given any clique cover $\mathbf{K}$ on the compatibility graph $\mathcal{K}(F)$ of a deterministic p-filter F, if $\mathbf{K}$ satisfies the zipper constraints $\mathscr{Z}(F)$ and covers all vertices of $\mathcal{K}(F)$, then the induced filter $F' = M(F, \mathbf{K})$ is deterministic and output simulates F.

The proof can be found in the extended version [11].

A surprising aspect of the preceding is how the zipper constraints —which are imposed to ensure that a deterministic filter is produced— enforce output-simulating behavior, albeit indirectly, too. One might have expected that this separate property would demand a second type of constraint, but this is not so.

On the other hand, needing to satisfy the zipper constraints of the input filter does not entail the imposition of any gratuitous requirements:

Lemma 21. Given any deterministic p-filters F and F', if F' output simulates F, then the induced clique cover $\mathbf{Q}(F, F')$ on the compatibility graph of F satisfies all zipper constraints in $\mathscr{Z}(F)$.

The proof can be found in the extended version [11].

4.3 Correspondence of MCCZC and SO-FM solutions

To establish the equivalence between MCCZC and SO-FM, we will show that the induced filter from the solution of MCCZC is a minimal filter for SO-FM, and the induced clique cover from a minimal filter is a solution for MCCZC.

Lemma 22. Minimal clique covers for MCCZC induce minimal filters for SO-FM.

The proof can be found in the extended version [11].

Lemma 23. A minimal filter for SO-FM with input F induces a clique cover that solves MCCZC with compatibility graph and zipper constraints of F.

The proof can be found in the extended version [11].

Together, they establish the theorem.

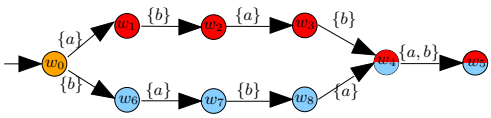
Theorem 24. The solution for MCCZC with compatibility graph and zipper constraints of a filter F induces a solution for SO-FM with input filter F, and vice versa.

Proof. Lemma 22 and Lemma 23 comprise the complete result. □

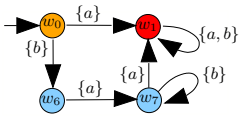
Having established this correspondence, any SO-FM can be solved by tackling its associated MCCZC problem, the latter problem being cast as a SAT instance and solved via a solver (see Section 6 for further details).

5 Generalizing to MO-FM

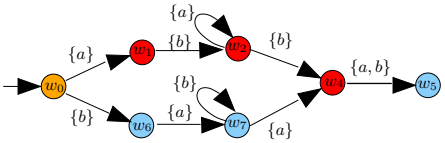
Finally, we generalize the previous algorithm to multi-outputting filters. In MO-FM problems, the input p-filter is deterministic but states in the p-filter may have multiple outputs. One straightforward if unsophisticated approach is to enumerate all filters under different output choices for the states with multiple outputs, and then solve every one the resulting deterministic single-outputting



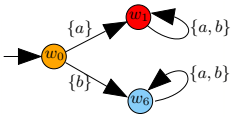
(a) An input filter with multi-outputting states. The colors in each vertex represents its output.



(b) A minimal filter when choosing to output the same color for w_4 and w_5.



(c) A minimal filter when choosing to output different colors for w_4 and w_5.



(d) A minimal filter for the input filter.

Fig. 7: A multi-outputting filter minimization problem.

filters as instances of SO-FM. The filter with the fewest states among all the minimizers could then be treated as a minimal one for the MO-FM problem.

Unfortunately, this is too simplistic. Prematurely committing to an output choice is detrimental. Consider the input filter shown in Figure 7a, it has two multi-outputting states (w_4 and w_5). If we choose to have both w_4 and w_5 give the same output, the SO-FM minimal filter, shown in Figure 7b, has 4 states. If we choose distinct outputs for w_4 and w_5, the SO-FM minimal filter, shown in Figure 7c, now has 7 states. But neither is the minimal MO-FM filter. The true minimizer appears in Figure 7d, with only 3 states. It is obtained by splitting both w_4 and w_5 into two copies, each copy giving a different output.

The idea underlying a correct approach is that output choices should be made together with the splitting and merging operations during filter minimization. Multi-outputting vertices may introduce additional split operations, but these split operations can still be treated via clique covers on the compatibility graph. This requires that we define a new compatibility relationship—it is only slightly more general than Definition 11:

Definition 25 (group compatibility)**.** Let F be a deterministic p-filter. We say that the set of states $U = \{u_1, u_2, \ldots, u_n\}$ are *group compatible*, if there is a common output on all their extensions, i.e.,

$$\forall s \in \bigcup_{u \in U} \mathcal{L}_F(u), \bigcap_{w' \in W'} c(w') \neq \varnothing, \text{ where } W' = \mathcal{V}_{u_1}(F, s) \cup \mathcal{V}_{u_2}(F, s) \cup \cdots \mathcal{V}_{u_n}(F, s).$$

With this definition, the minimization of a deterministic multi-outputting filter can also be written and solved as a slightly generalized MCCZC problem, where (*i*) the compatibility graph is generalized toward a *compatibility simplicial complex*; (*ii*) to capture the set of vertices that can be merged, the clique in the MCCZC problem is generalized to be a simplex in the *compatibility simplicial complex*; (*iii*) the zipper constraints are redefined by replacing "mutually compatible" with the "group compatible" states; (*iv*) the objective is to find the set of simplices that covers all vertices in the compatibility simplicial complex. This generalized MCCZC problem, termed GMCZC, will be solved in the next section.

6 Reduction from MCCZC and GMCZC to SAT

Prior algorithms for filter minimization used multiple stages to find a set of vertices to merge, solving a graph coloring problem repeatedly as more constraints are identified. In contrast, an interesting aspect of MCCZC and its generalized version is that it tackles filter minimization as a constrained optimization problem with all constraints established upfront. Thus the clique (simplex) perspective gives an optimization problem which is tangible and easy to visualize. Still, being a new invention, there are no solvers readily available for direct use. But reducing MCCZC (GMCZC) to Boolean satisfaction (SAT) enables the use of state-of-the-art solvers to find minimum cliques (simplices).

We follow the standard practice for treating optimization problems via a decision problem oracle, *viz.* define a k-MCCZC (k-GMCZC) problem, asking for the existence of a cover with size k satisfying the zipper constraints; one then decreases k to find the minimum cover. Each k-MCCZC (k-GMCZC) problem can be written as a logic formula in conjunctive normal form (CNF), polynomial in the size of the k-MCCZC (k-GMCZC) instance, and solved. Detailed explanation of the CNF generation from the k-MCCZC (k-GMCZC) problem must be deferred to the extended version [11].

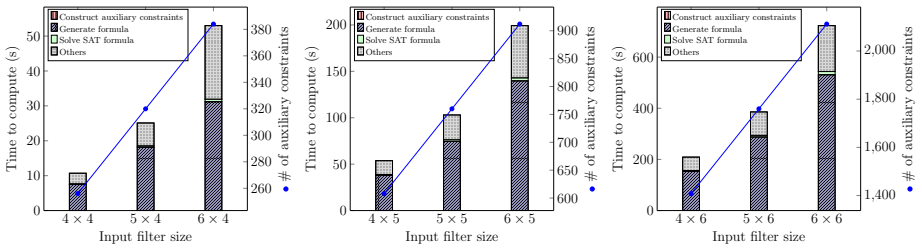
7 Experimental results

The method described was implemented by leveraging a Python implementation of 2018 SAT Competition winner, MapleLCMDistChronoBT [12,13]. Their solver will return a solution if it solves the k-MCCZC problem before timing out. If it finds a satisfying assignment, we decrease k, and try again. Once no satisfying assignment can be found, we construct a minimal filter from the solution with minimum k.

First, as a sanity check, we examined the minimization problems for the inputs shown in Figure 2a, Figure 5a and Figure 7a. Our implementation takes 0.05 s, 0.06 s and 0.26 s, respectively, to reduce those filters. All filters found have exactly the minimum number of states, as reported above.

Next, designed a test where we could examine scalability aspects of the method. We generalized the input filter shown in Figure 7a to produce a family of instances, each described by two parameters: the $n \times m$-input filter has n rows and m states at each row. (Figure 7a is the 2×3 version.) Just like the original filter, the states in the same row share the same color, but the states in different rows have different colors. The initial state w_0 outputs a single unique color; the last two states, w_4 and w_5, output any of the n colors. In this example, the states in the same row, together with w_4 and w_5, are compatible with each other.

The time to construct the zipper constraints, prepare the formulas and the time used by the SAT solver were recorded. We also measured the number of zipper constraints found by our algorithm. Figure 8 summarizes the data for $(n, m) \in \{4, 5, 6\} \times \{4, 5, 6\}$. The result shows that about 70 % of the time is used in preparing the logical formula, with the SAT solver and construction of the zipper constraints accounting for only a very small fraction of time.

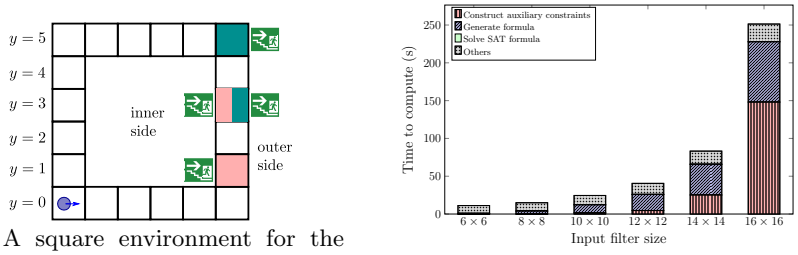


(a) Minimizing inputs with 4 vertices per row.

(b) Minimizing inputs with 5 vertices per row.

(c) Minimizing inputs with 6 vertices per row.

Fig. 8: A scalability test on the multi-outputting filter minimization problems. Inputs are $n \times m$ parameterized instances akin to Figure 7a.



(a) A square environment for the robot to track the exits with the observation indicating its current row.

(b) Compute times to minimize filters with different sizes.

Fig. 9: Navigating to exit a square environment with a row sensor.

In light of this, to further dissect the computational costs of different phases, we tested a robot in the square grid environment shown in Figure 9a. The robot starts from the bottom left cell, and moves to some adjacent cell at each time step. The robot only receives observations indicating its row number at each step. We are interested in small filter allowing the robot to recognize whether it has reached a cell with an exit (at the inner side or outer side). States with both inner and outer exits have multiple outputs. To search for a minimal filter, we firstly start with deterministic input filters for a grid world with size 6×6, 8×8, 10×10, 12×12, 14×14, 16×16, and then minimize these filters. We collected the total time spent in different stages of filter minimization, including the construction of zipper constraints, SAT formula generation and resolution of SAT formula by the SAT solver. The results are summarized visually in Figure 9b.

In this problem, the number of states in the input filter scales linearly with the size of the square. So does the minimal filter. But the particular problem has an important additional property: it represents a worst-case in a certain sense because there are no zipper constraints. We do not indicate this fact to the algorithm, so the construction of zipper constraints examines many cliques, determining that none apply. The results highlight that the construction of the zipper constraints quickly grows to overtake the time to generate the logical formula — even though, in this case, the zipper constraint set is empty.

The preceding hints toward our direction of current research: the construction of $\mathscr{Z}(F)$ by naïvely following Definition 14 is costly. And, though the SAT

formula is polynomial in the size of the MCCZC instance, that instance can be very large. On the other hand, the need for a zipper constraint can be detected when the output produced fails to be deterministic. Hence, our future work will look at how to generate these constraints lazily.

8 Conclusion

With an eye toward generalizing combinatorial filters, we introduced a new class of filter, the cover filters. Then, in order to reduce the state complexity of such filters, we re-examined earlier treatments of traditional filter minimization; this paper has shown some prior ideas to be mistaken. Building on these insights, we formulate the minimization problem via compatibility graphs, examining covers comprised of cliques formed thereon. We present an exact algorithm that generalizes from the traditional filter minimization problem to cover filters elegantly.

References

1. A. Censi, "A Class of Co-Design Problems With Cyclic Constraints and Their Solution," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 96–103, 2017.
2. A. Pervan and T. D. Murphey, "Low complexity control policy synthesis for embodied computation in synthetic cells," in *WAFR*, Mérida, México, 2018.
3. F. Z. Saberifar, J. M. O'Kane, and D. A. Shell, "The hardness of minimizing design cost subject to planning problems," in *WAFR*, Mérida, México, 2018.
4. S. M. LaValle, "Sensing and filtering: A fresh perspective based on preimages and information spaces," *Found. and Trends in Rob.*, vol. 1, no. 4, pp. 253–372, 2010.
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, S. Ghasemlou, J. M. O'Kane, and D. A. Shell, "Set-labelled filters and sensor transformations," in *Robotics: Science and Systems*, 2016.
7. F. Z. Saberifar, S. Ghasemlou, D. A. Shell, and J. M. O'Kane, "Toward a language-theoretic foundation for planning and filtering," *International Journal of Robotics Research—WAFR'16 special issue*, vol. 38, no. 2-3, pp. 236–259, 2019.
8. 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.
9. H. Rahmani and J. M. O'Kane, "On the relationship between bisimulation and combinatorial filter reduction," in *Proceedings of IEEE International Conference on Robotics and Automation*, Brisbane, Australia, 2018, pp. 7314–7321.
10. ——, "Integer linear programming formulations of the filter partitioning minimization problem," *Journal of Combinatorial Optimization*, vol. 40, pp. 431–453, 2020.
11. Y. Zhang and D. A. Shell, "Cover Combinatorial Filters and their Minimization Problem (Extended Version)," *arXiv preprint arXiv:2002.07153*, 2020.
12. A. Nadel and V. Ryvchin, "Maple_LCM_Dist_ChronoBT: featuring chronological backtracking," in *Proc. of SAT Competition*, 2018, pp. 29–30.
13. A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437.