On the design of minimal robots that can solve planning problems

Dylan A. Shell, Jason M. O'Kane, and Fatemeh Zahra Saberifar

***Abstract*—This article examines the selection of a robot's actuation and sensing hardware to minimize the cost of that design, while ensuring that the robot is capable of carrying out a plan to complete a task. Its primary contribution is in the study of the hardness of reasonable formal models for that minimization problem. Specifically, for the case in which sensing hardware is held fixed, we show that this algorithmic design problem is NP-hard even for particularly simple classes of cost functions, confirming what many perhaps have suspected about this sort of design-time optimization. We also introduce a formalism, based on the notion of label maps, for the broader problem in which the design space encompasses choices for both actuation and sensing components. As a result, for several questions of interest, having both optimality and efficiency of solution is unlikely. However, we also show that, for some specific types of cost functions, the problem is either polynomial time solvable or fixed-parameter tractable.**

***Note to Practitioners*—Despite the primary results being theoretical and, further, taking the form of bad news, this paper still has considerable value to practitioners. Specifically, assuming one has been employing heuristic or approximate solutions to robot design problems, the paper serves as a justification for doing so. Moreover, it delineates some circumstances in which one can, in a sense, do better and achieve genuine optima with practical algorithms.**

***Index Terms*—Primary Topics: Planning, Design**
Secondary Topic Keywords: Computational Complexity, Hardness

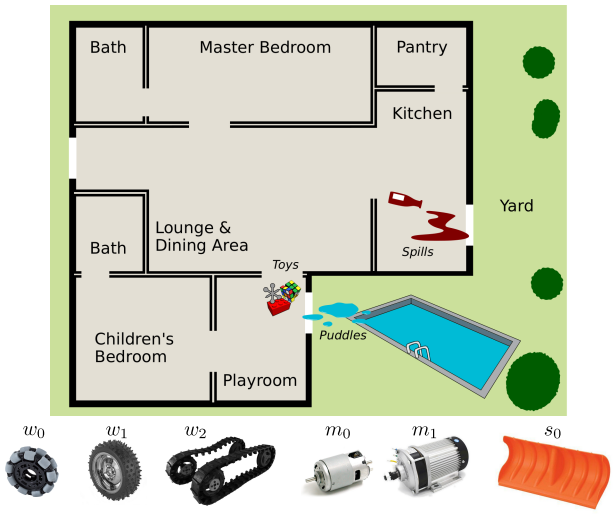

Fig. 1. A motivating example that illustrates how, for an otherwise simple planning problem, straightforward design constraints can quickly lead to considerations that are quite complex. [top] Suppose we want to design a cleaning robot that is able to navigate to any room in the house. [bottom] The robot's ability to navigate *spills*, *puddles*, and *toys* depends on the particular way it is equipped. Among other things, design involves choosing this equipment by reasoning about resources and their interactions in the context of the robot's task.

I. Introduction

Research on autonomous robots is entering a phase of maturation: already there is general agreement on the centrality of estimation and planning problems and there is broad consensus on basic representations and algorithms to address the underlying problems; the last decade has seen the emergence of (open source) software infrastructure with adoption that is increasingly widespread; and an inchoate industry is pursuing profitable applications. Some academic researchers have begun to move away from questions concerning how to program a given robot, turning to questions of the form: *Given some resources and a collection of resource constraints, what is the ideal robot, considering that design choices influence feasible behavior?*

Evidence for this growing interest can be seen in recently held workshops with titles such as 'Minimality and Design Automation' (RSS'16), 'Minimality and Trade-offs in Automated Robot Design' (RSS'17), and 'Autonomous Robot Design' (ICRA'18).[1] Naturally enough, the research is focused on the development of algorithmic tools to help answer such design questions. Several ideas have been proffered as useful ways to tackle robot design problems. They display great and refreshing variety, including angles on the problem that emphasize fabrication, prototyping and manufacturability [7], [9], [13], [18], [19]; formal methods for (controller and hardware) synthesis of robots [14], [19], [20], [27], [31]; simulators and methods for interactive design [11]; compositional frameworks along with catalogs of components [2], [3], [26]; software for fault tracking and component-based identification [34]; and so on.

The present paper deals with design problems that we believe many already suspect to be difficult problems, but for which actual hardness results have not appeared in the literature. (Somewhat surprisingly, at least to us!) Though many aspects of the hardness of planning have been examined, prior work has tended consider costs such as time or

D. A. Shell (dshell@tamu.edu) is with the Department of Computer Science and Engineering at Texas A&M University, College Station, TX, USA. J. M. O'Kane (jokane@cse.sc.edu) is with the Department of Computer Science and Engineering at the University of South Carolina, Columbia, SC, USA. F. Z. Saberifar (fz.saberifar@modares.ac.ir) is with the Department of Computer Science, Tarbiat Modares University, Tehran, Iran.

[1]For a report drawing on and summarizing aspects of the first two workshops, which was presented at the third, see [22].

energy consumption which are associated with plan *execution* and which often bear little relation to the cost of realizing the particular robot design. It is surprising that even rather immediate questions about robot design lack formal analysis from the complexity theoretic point of view. Thus, we begin to remedy this fact.

To help make matters concrete, consider the example illustrated in Figure 1. We are interested in designing a home cleaning robot and, to be effective, the robot must be able to navigate from region to region within the environment. The ability to navigate depends on the actuators that the robot is equipped with and their ability depends on the particular assortment of clutter that is encountered: 'spills' and 'puddles' and 'toys'. The cheapest wheels are w_0 which, though inexpensive, can't surmount any of these three. Wheel w_1, while operating fine in wet conditions, still fails with toys. The third option, w_2, can convey the robot over all three, but requires a heavier duty motor (m_1) than the standard one (m_0). If any robot is equipped with a scoop s_0, it is no longer hindered by dry detritus, though the scoop is ineffective with liquids. And, alas, the chassis we have available can't support both s_0 and m_1 simultaneously.

Our task requirements dictate that the cleaning robot ought to be able move from any room to any other, the morning following a lively party, that is, under worst-case conditions. Here, multiple designs satisfy these task requirements while also respecting the component limitations (e.g., the chassis weight requirement). Even if identifying that set of designs doesn't seem too onerous, the fact that the solutions in our example seem to need, at a glance, to satisfy distinct discrete constraints (overcome wet, or dry obstacles, or both), and may do so in multiple ways, suggests that the solution set has a non-trivial combinatorial structure. Beyond this single example, in problems with greater varieties of available components, one might imagine that the complexity of this structure may scale exponentially. (We revisit, formalize, and solve this particular example in Section III-D.)

We are interested in minimizing some cost, typically over the set of all useful designs. Suppose we are given some reasonable cost function that assigns a cost to every set of resources—where, for now, intuitive properties such as non-negativity make up 'reasonableness'. Perhaps, in a mood of generosity, we opt to simplify things (ignoring bulk purchase pricing breaks and economies of scale) and assume monotonicity in costs. Can one find the cheapest design efficiently then?

This paper formalizes questions about certain robot design choices and their effect on the resulting robot's ability to plan and achieve goals. The work contributes hardness results primarily, but the purpose and import of such analyses is not merely to underscore that we expect to have to forgo overall global optimality, but also to help understand whether thinking about such problems is fundamentally impractical or whether there is hope for good approximation algorithms.

An interesting (even novel) aspect of the approach taken in this paper is to think about design problems as planning problems, but with special sorts of cost functions. Because planning involves choosing actions, this point of view will perpetuate the distinction between actions and observations — the former having to do with design choices for actuators, the latter choices for sensors— in a way that may, at least initially, appear unnatural. In the penultimate section of the paper, both types of design choices are unified through the introduction of another concept, that of a label map. This concept will also forge a connection to some existing work.

After a review of related work (Section II), this paper contributes a formalization of the design cost minimization problem for actuators (Section III), a hardness result for that problem in general (Section IV), polynomial-time algorithms for certain natural special cases (Section V), and a generalization of the approach to enable design choices over sensors as well (Section VI).

A preliminary version of this work appeared at WAFR 2018 [30]. Substantial new results in this version include a treatment of the gadget selection cost in Section III-D and a generalization in Section VI to problems that include sensor design cost.

II. Related Work

Having already provided broad context for robot design questions, here we draw attention to particular threads bearing an especially close relationship to this paper. Detailed connection to the authors' own prior work is postponed until Section VII, after presentation of the technical results.

Censi [2], [3] introduced a theory of *co-design*, which adopts a poset-based optimization point of view in order to relate functionality, implementations, and resources to each another. He demonstrates his methodology by applying it to questions about the minimal resources required to realize some functionality. This question, in various forms, has a rich history (cf. [6], [23], [24], [32]). An especially noteworthy aspect of Censi's theory is how monotone properties enable efficient optimization; his work contrasts with the present work—here monotonicity offers only a limited salve. More recently, work [1] has examined co-design without the monotone property.

In contrast to that theory's formulation, the present paper emphasizes that choosing to include some gadget within a robot design may unlock some choices—but whether they are needed or not depends on sequential and conditional aspects of what the robot needs to do. When what it 'needs to do' looks like executing a plan that came from solving some planning problem, then that multi-stage interaction between the various design-time options is a planning problem itself—which hints at the likely difficulty of ensuring optimality.

In the examples that follow, we provide an illustrative instance where determining whether a set of resources yields a feasible design or not is expensive to compute (cf. Section III-D). Significantly, however, the hardness results do not depend on this source of complexity: even when the feasibility of a design is trivial to compute, and a preference over designs is cheap to evaluate, still optimization of designs remains challenging. This contrasts with the papers cited above.

It is also worth pointing out the influential work of Hauser [12] who examined a variant on motion planning

where, starting with an infeasible problem, he seeks the minimal set of constraints to be removed to make the problem solvable. In a sense, this is the inverse of the problem we examine: we ask not about removing challenging elements from the problem, but rather about the addition of capabilities to the robot.

III. Definitions and Problem Formulation

This section presents some definitions necessary to introduce the algorithmic problem addressed in the balance of the paper.

A. Planning problems and plans

We are interested in reasoning about the ability of robots equipped with varying action capabilities to complete certain tasks. Following our earlier work [29], we model both planning problems and the plans that solve them using procrustean graphs.

Definition 1: A *procrustean graph (or p-graph)* is a bipartite directed graph $G = (V_0, V, E, l_u, l_y, U, Y)$ in which:

1) The finite set of *states* V contains two disjoint kinds of states called *action states* $V_{\rm u}$ and *observation states* $V_{\rm y}$, with $V = V_{\rm u} \cup V_{\rm y}$.
2) The multiset of edges $E = E_{\rm u} \cup E_{\rm y}$ is composed of *action edges* $E_{\rm u}$ and *observation edges* $E_{\rm y}$.
3) Each action edge $e \in E_{\rm u}$ goes from an action state to an observation state, and is labeled with a finite nonempty set of actions $l_u(e)$ drawn from the *action space* U.
4) Likewise, each observation edge $e \in E_{\rm y}$ goes from an observation state to an action state, and is labeled with a finite nonempty set of observations $l_y(e)$ drawn from an *observation space* Y.
5) The set $V_0 \subseteq V$ represents a nonempty set of initial states. All states V_0 should be of the same kind: either $V_0 \subseteq V_{\rm u}$ or $V_0 \subseteq V_{\rm y}$.

The interpretation is that a p-graph describes a set of event sequences that alternate between actions and observations. We say that an event sequence —that is, a sequence of actions and observations— is an *execution* on a p-graph if there exists some start state in the p-graph from which we can trace the sequence, following edges whose labels include each successive event in the sequence. For two p-graphs G_1 and G_2, we say that an event sequence is a *joint execution* if it is an execution for both G_1 and G_2.

For simplicity, this paper focuses on *state-determined* p-graphs, which are those where, for any given state, the labels on the edges departing that state are mutually disjoint, and where V_0 is a singleton. In such p-graphs, any execution can be traced in no more than one way.

We can use p-graphs to model both planning problems and plans.

Definition 2: A *planning problem* is a p-graph G with a goal region $V_{\rm goal} \subseteq V(G)$. A *plan* is a p-graph P with termination region $P_{\rm term} \subseteq V(P)$.

We direct the reader again to the example in Figure 1. It is clear, certainly, how a planning problem involving motion from one particular room to another —when both rooms are given— can be posed as a p-graph with a goal region. See Figure 2. A relatively straightforward extension, leveraging the ability to designate multiple states as start states, can express the problem of being able to transit from any room to any other. Specifically, one would combine the individual p-graphs for each specific start and goal pair, and then perform an expansion of this combined graph to a state-determined presentation.

Informally, we say that a plan is *safe* on a planning problem, if the plan has observation edges for any observation that might be generated by the planning problem at any reachable state pair, and likewise the planning problem has action edges for any action that might be generated by the plan. We say that a plan is *finite* on a planning problem if there is some upper bound on the length of all joint executions. We omit the complete formal definitions of safe and of finite, which are somewhat tedious, referring the reader instead to [29].

Now we can define the notion of a plan solving a planning problem.

Definition 3: A plan $(P, P_{\rm term})$ with at least one execution *solves* a planning problem $(W, V_{\rm goal})$ if P is finite and safe on W, and every joint-execution $e_1 \cdots e_k$ of P on W either reaches a vertex in $P_{\rm term}$, or is a prefix of some execution that does and, moreover, all the $e_1 \cdots e_k$ that reach a vertex $v \in V(P)$ with $v \in P_{\rm term}$, reach a vertex $w \in V(W)$ with $w \in V_{\rm goal}$.

The intuition here is that a plan solves a planning problem when every possible joint execution eventually terminates in the goal region.

B. Design costs

To simplify the exposition, we attend first to plans that minimize a *design cost* function which depends on which actions are utilized in a plan. (We defer questions about cost minimization associated to observations until Section VI.) Note that we are concerned here only with each action's presence in (or absence from) the plan in question —we are not concerned with how many times an action is carried out on any particular execution of a plan. The next two definitions formalize this idea.

Definition 4: For an action set U, a *cost function* $c : 2^U \to \mathbb{R} \cup \{\infty\}$ assigns an extended real number cost to each subset of U.

Definition 5: For any plan $(P, P_{\rm term})$ that solves planning problem $(W, V_{\rm goal})$, we write $\mathcal{A}(P, W) \subseteq U$ to denote the set of actions that appear in any joint execution of P on W. We then define the *design cost of P on W* as $c(\mathcal{A}(P, W))$.

When the planning problem W is clear from the context, we overload the notation slightly by writing $\mathcal{A}(P)$ for $\mathcal{A}(P, W)$ and likewise $c(P)$, instead of $c(\mathcal{A}(P, W))$.

The essential idea entailed by Definitions 4 and 5 is that c is a measure of the cost of a plan that depends only upon which actions are used by the plan, rather than upon how frequently those actions are used when the plan is executed. The intent is to establish a dependence between $c(P)$ and the cost of constructing a robot that is capable of executing each action in $\mathcal{A}(P)$. Finding a plan that minimizes this design cost

can give some insight into the simplest robots, in the sense of actuator complexity, that can solve the planning problem.

Some example cost functions, intended to illustrate the expressive flexibility of the definitions, follow.

Example 1 (counter design cost): Given a plan P, consider the design cost $c(P) = |\mathcal{A}(P)|$. This cost function simply counts number of actions utilized by P. By minimizing this *counter design cost*, we minimize the number of distinct actions used in the plan.

Example 2 (weighted sum design cost): We can generalize the counter design cost by defining a weight function $w : U \to \mathbb{R}$ that assigns a specific cost to each action, and then defining $c(P) = \sum_{u \in \mathcal{A}(P)} w(u)$.

Example 3 (binary design cost): Suppose we are given a set of actions A' that a robot designer would prefer to use, if possible. Define $c_{A'} : 2^U \to \{0, 1\}$ as

$$c_{A'}(P) = \begin{cases} 0 & \text{if } \mathcal{A}(P) \subseteq A' \\ 1 & \text{otherwise} \end{cases}.$$

For a given set of actions A' and a plan P, design cost $c_{A'}$, called a *binary design cost*, gives a value of 0 if all actions used to carry out P are from the preferred set A', and a value of 1 if any additional action, not in A', is used.

Example 4 (ordered actions): Suppose we have a choice of options for which actuators to include, each of which subsumes its predecessors, both in ability and in expense. We would like to identify which of these options is the simplest that suffices to solve a particular planning problem. We can model this kind of situation by assuming that U is a finite ordered set $U = \{u_1, \ldots, u_n\}$, and defining

$$c(P) = \max\{i \mid u_i \in \mathcal{A}(P)\}.$$

Example 5 (monotone cost functions): Another natural class of cost functions are those that are monotone, in the following sense: A cost function is *monotone* if, for any sets $U_1 \subseteq U$ and $U_2 \subseteq U$, we have

$$U_1 \subseteq U_2 \implies c(U_1) \leq c(U_2).$$

Monotone cost functions are interesting because they capture the eminently sensible idea that adding additional abilities to the robot should not decrease the cost. Notice in particular that the counter design cost (Example 1), binary design cost (Example 3), and ordered action design cost functions (Example 4) are all monotone.

C. Design cost minimization

We can now state the main algorithmic problem. Following the standard pattern, we consider both optimization and decision versions of the problem.

Decision Problem: Design minimization (DECDM)

Input: A planning problem (G, V_{goal}), where G is state-determined, a cost function c, and a real number k.

Output: YES if there is a plan (P, P_{term}) that solves (G, V_{goal}), with design cost $c(\mathcal{A}(P, W)) \leq k$. NO otherwise.

Optimization Problem: Design minimization (OPTDM)

Input: A planning problem (G, V_{goal}) with state-determined G, and cost function c.

Output: A plan (P, P_{term}) that can solve (G, V_{goal}) such that $c(\mathcal{A}(P, W))$ is minimal, or NONE indicating that no solutions exist.

We can also form specialized versions of each of these problems by placing restrictions on the design cost function c. Our objective, in the following sections, is to classify the types of design cost functions for which this problem can be solved efficiently, and the types for which these problems are hard.

D. A more practical example: Gadget selection

Though the example cost functions thus far have been relatively abstract, we emphasize that our formalism is sufficiently expressive to encode practical robot design problems. As a concrete illustration, recall the example shown in Figure 1. The context there is to select, from a catalog of available components —called *gadgets* here because the letter g is unused in our notation so far— that satisfies some physical constraints and suffices to solve the planning problem, while minimizing the total cost of the selected gadgets.

In that spirit, suppose a roboticist needs to solve a planning problem (W, V_{goal}). Our hero would like to select, from a finite catalog of choices $G = \{g_1, \ldots, g_m\}$, a subset of gadgets with which to equip the robot. We might formalize that problem, including the costs of each gadget, the constraints on the overall design, and the ability each gadget to enable the robot to execute various actions, by defining:

1) A map from gadgets to their solo costs $c_{\text{g}} : G \to \mathbb{R} \cup \{\infty\}$. This map models the cost of each individual gadget.
2) A Boolean formula Ψ in which the gadget symbols from G appear as variables. The intended interpretation is that $g_i = \text{True}$ if and only if gadget g_i is included in the design. The formula Ψ models the constraints on which sets of gadgets are physically realizable. That is, Ψ is satisfied if and only if a given selection of gadgets is valid, in the sense of satisfying any relevant constraints. For convenience, we use the streamlined notation $\Psi(X)$ to refer to a Boolean value that indicates, for any $X \subseteq G$, whether gadget set X is valid according to Ψ.
3) For each action u in the action space U of the planning problem p-graph W, a Boolean formula Λ_u over those same variables. Each such formula Λ_u should be satisfied for exactly those gadget sets that would enable the robot to execute action u. As with Ψ, we write $\Lambda_u(X)$ for the Boolean value that indicates whether a robot equipped with gadget set X can execute action u.

In this setting, we can express the valid subsets of G for which a plan p-graph P is executable as

$$\mathcal{G}(P) = \left\{ S \subseteq G \;\middle|\; \Psi(S) \;\wedge \bigwedge_{u \in U} \Lambda_u(S) \right\}. \tag{1}$$

Thus, we can define a *gadget-selection cost function* that depends on G, Ψ, and collection of Λ_u formulae, that captures

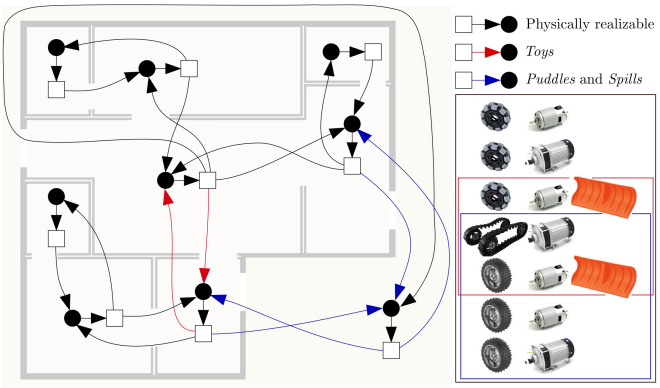

Fig. 2. A model of the environment from Figure 1 as a p-graph.

the notion of selecting gadgets that enable a set of actions sufficient to solve the planning problem, while minimizing the total cost and obeying the global constraints in Ψ:

$$c_{G,\Psi,\{\Lambda_u\}}(P) = \min_{S \in \mathcal{G}(P)} \sum_{s \in S} c_g(s). \tag{2}$$

That is, the cost of a plan is determined by the total cost of the set of gadgets that both enable the plan to execute and satisfy the global constraints.

Let us now return to the specific example in Figure 1 and solve it by minimizing a suitable gadget selection cost function. There are six available gadgets $G = \{w_0, w_1, w_2, m_0, m_1, s_0\}$. A portion of the p-graph model of this environment appears in Figure 2, in which actions direct the robot to move directly between adjacent rooms. The actions shown are colored according to the hazards the robot must overcome to complete each action: black for unobstructed actions, red for actions that force the robot to traverse toys, and blue for actions that direct the robot through puddles or spills. Recall that the objective is for our robot to be able to traverse reliably between any pair of locations in this environment, so the full planning problem p-graph would consist of a number of distinct copies of the fragment shown here, each contributing one state to the combined set of initial states and one state to the overall goal region.

We have several constraints on which sets of gadgets can be realized into a working robot, including the robot's need for both wheels and a motor, the dependence of w_2 on m_1, and the incompatibility of s_0 and m_1. Thus, we have:

$$\Psi = (w_0 \vee w_1 \vee w_2) \wedge (m_0 \vee m_1) \wedge (\neg w_2 \vee m_1) \wedge (\neg s_0 \vee \neg m_1).$$

Each action u has for its Λ_u one of three distinct action-enablement formulae:

1) Actions with neither toys nor wetness (black in Figure 2) have no additional requirements. For these actions, we use the trivial formula $\Lambda_u = \text{True}$.
2) Actions that cross toys (red in Figure 2) can be executed only by robots with either the top-of-the-line wheels (w_2) or the scoop (s_0). Thus, for these actions we set $\Lambda_u = w_2 \vee s_0$.
3) Actions that cross puddles or spills cannot be executed by robots using the weakest wheels (w_0). For these actions, we set $\Lambda_u = \neg w_0$.

Finally, we assign individual gadget costs to each gadget. If, for example, we choose $c_g(w_0) = 2$, $c_g(w_1) = 4$, $c_g(w_2) = 6$, $c_g(m_0) = 5$, $c_g(m_1) = 10$, $c_g(s_0) = 7$, then the optimal gadget set is $\{w_1, m_0\}$, with total cost 9. This selection enables actions that cross the normal and wet transitions, which in this scenario is sufficient to solve the planning problem, i.e. to navigate between any pair of locations.

Note, however, that changes to either the structure of the environment—for example, the addition of toys between the children's bedroom and their bath—or to the individual costs of each gadget would alter this solution. This highlights a central feature of the contribution of this paper, namely that the design optimization problem we consider can account for both general resource availability and component costs, as well as the spatial or temporal factors that determine which components suffice for the robot to complete its task.

The reader may also note that the cost function introduced in (2) depends, via (1), on enumerating satisfying assignments to Ψ. In light of the Cook-Levin theorem [4], [16], this implies that merely evaluating the cost of a design in this context is a hard algorithmic problem. This is perhaps not a surprise: The sorts of overlapping and interacting constraints that govern the validity of robot designs and the actions they enable are, in an informal sense, the essence of what makes problems NP-Complete. Nonetheless, the results in the next section remain interesting in part because they demonstrate that the design problem remains hard in cases where it is extremely simple, or trivial even, to determine the validity of, and preferences between, designs.

IV. Hardness of Design Cost Minimization

In this section, we prove that the decision version of the design cost minimization problem is NP-complete.

A. The General Case

Our proof proceeds by reduction from the standard set cover problem, which is known to be NP-complete [15]:

Decision Problem: SetCover
Input: A universe set R with n elements, a set T comprised of m sets $T_1, \ldots, T_m$ such that $\bigcup_{i=1}^{m} T_i = R$, and an integer k.
Output: Yes if there is some set $I \subseteq T$ such that I covers all elements of R and the size of I is at most k. No otherwise.

Given an instance (R, T, k) of SetCover, we construct a problem instance of $(G, V_{\text{goal}}, c, k')$ of DecDM as follows:

1) Begin with an empty p-graph G. Choose $U = \{u_1, \ldots, u_m\}$, with one action for each of the sets in T, for its action space. Choose $Y = \{\boxdot\}$, a singleton set containing a dummy observation, for its observation space.

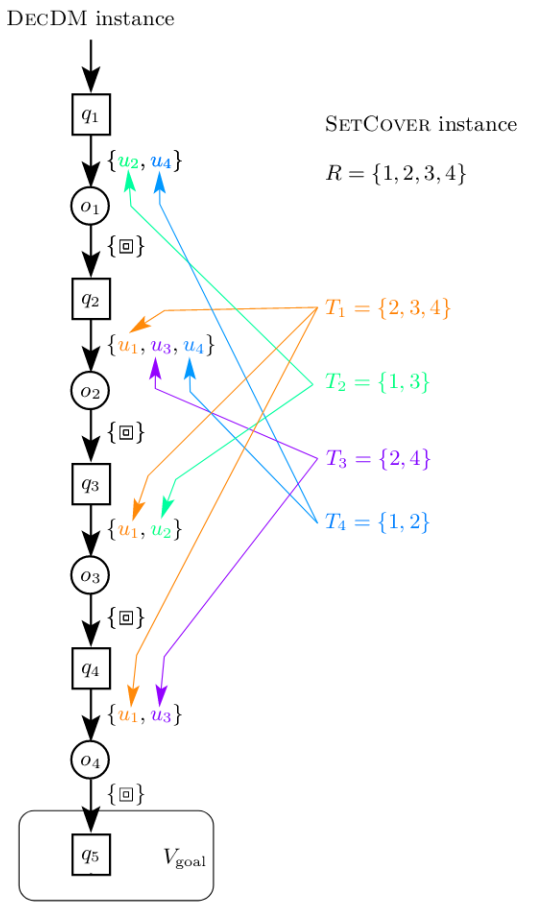

Fig. 3. An example of the construction of a DECDM instance from a SETCOVER instance. Given a set cover instance with $R = \{1, 2, 3, 4\}$, $T = \{\{2, 3, 4\}, \{1, 3\}, \{2, 4\}, \{1, 2\}\}$ and $k = 2$, we construct the planning problem shown on the left. Every subset of T that covers R corresponds to a plan that can reach V_{goal} from the initial state. For this example, we can cover R in the SETCOVER instance by choosing $\{1, 3\}$ and $\{2, 4\}$; likewise one can solve the planning problem using only the actions u_2 and u_3.

2) For each element $x_i \in R$ of the universe R, add to G an action state q_i and an observation state o_i. In addition, insert an extra action state q_{n+1} into G.
3) From each action state q_i, except q_{n+1}, connect the corresponding observation state o_i by a directed edge e_i. Determine the label $l_u(e_i)$ as follows: The label for edge e_i includes action u_j if and only if the set T_j contains x_i. That is, we set $l_u(e_i) = \{u_j \mid x_i \in T_j\}$.
4) Connect each observation state o_i to the subsequent action state q_{i+1} with a directed edge e'_i, labeled with the sole observation $\boxdot$, so that $l_y(e'_i) = \{\boxdot\}$.
5) Designate $V_0 = \{q_1\}$ as the only initial state of G.
6) Designate $V_{\text{goal}} = \{q_{n+1}\}$ as the goal region of the planning problem.
7) For design cost function, choose a counter design cost function,
$$c(P) = |\mathcal{A}(P)|.$$
8) Select $k' = k$.

Figure 3 shows an example of this construction. Note that the time needed for this construction is polynomial in the input size.

Fig. 4. The plan that solves the planning problem arising from the construction shown in Figure 3. It uses u_2 and u_3, each corresponding to $T_2 = \{1, 3\}$ and $T_3 = \{2, 4\}$, respectively.

Next, we prove that this construction is indeed a reduction from SETCOVER to DECDM. The intuition is that each action state in the constructed planning problem acts as a sort of 'gate' to check whether a certain element has been covered. If some choice of elements selected from T is adequate to fully cover R, then the corresponding plan will be able to transition through each of these gates to the goal. If not, not. The next two lemmas make this idea more precise.

Lemma 1: For any instance (R, T, k) of SETCOVER, consider the DECDM instance $(G, V_{\text{goal}}, c, k')$ constructed as described above. If there exists a subset of T of size at most k that covers R, then there exists a plan (P, P_{term}) that solves (G, V_{goal}), for which $c(P) \leq k$.

Proof: Let $I \subseteq T$ denote a coverage set for R, which has $|I| \leq k$. To produce a plan (P, P_{term}), we start with a copy of the constructed planning problem (G, V_{goal}), and remove from G all action labels that do not correspond to elements of I. That is, for any i for which $T_i \notin I$, we remove u_i from P. Note that $V_{\text{goal}} = P_{\text{term}} = \{q_{n+1}\}$. (Figure 4 shows our running example.) Clearly $c(P) = |I| \leq k$. So it remains only to show that (P, P_{term}) solves (G, V_{goal}).

First, we prove that P is finite and safe on G. Since the construction yields a linear chain of events in both G and P then there are joint-executions with lengths from 0 to at most $2n$. Thus, P is finite on G. Note also that, according to the construction of G and P, we can conclude that for every joint-execution $e_1 \cdots e_k$ on P and G that leads to $v \in P$ and $w \in G$, if v is an action state then the label set of action edge e, originating at v, is a subset of label set of action edge e', originating at w. We also know that P and G have the same single observation label $\boxdot$ for each of their observation states. Therefore, P is safe on G.

Because of the shared linear chain form of both G and P and existence of only one initial state and one goal state, there is one unique joint execution that reaches V_{term} in P, which by construction also reaches V_{goal} in G. The linear chain structure also ensures that every other joint execution is a prefix of this one, which implies that every joint-execution $e_1 \cdots e_k$ on P and G either leads to the goal state q_{n+1} or is a prefix of some execution that leads to q_{n+1}, as required by Definition 3. Therefore, (P, V_{term}) solves (G, V_{goal}). □

Lemma 2: For any instance (R, T, k) of SETCOVER, consider the DECDM instance $(G, V_{\text{goal}}, c, k)$ constructed as described above. If there exists a plan (P, P_{term}) that solves (G, V_{goal}), for which $c(P) \leq k$, then there exists a subset of T of size at most k that covers R.

Proof: Let (P, P_{term}) be some plan with $c(P) \leq k$ that solves (G, V_{goal}). Consider some execution $e_1 \cdots e_m$ on P. We may assume it reaches a vertex in P_{term} without sacrificing generality for, if it does not, it is certainly the prefix of some execution which does, according to Definition 3. Thus, when

e_m arrives at a vertex in $P_{\text{term}} \subseteq V(P)$, it must be that $e_1 \cdots e_m$ reaches a vertex in $V_{\text{goal}} \subseteq V(W)$. But, by construction of G, that means $e_1 \cdots e_m = u_{i_1} \boxdot u_{i_2} \boxdot \ldots u_{i_n} \boxdot$, and we see that $n = |R|$. Define $I = \{T_j \in T \mid j \in \{i_1, i_2, \ldots, i_n\}\}$ where j is taken over the set simply collecting the indices of the actions in the execution. Clearly $I \subseteq T$ and, because $T_x \mapsto u_x$ corresponds elements of I with $\mathcal{A}(P)$ in a one-to-one fashion, $|I| = |\mathcal{A}(P)| = c(P) \leq k$.

All that remains is to show that I covers R. Reaching the goal state requires transiting, linearly, through $q_1 o_1 q_2 \ldots o_n q_{n+1}$. So, for any $w \in R$, the action u_{i_w} was used to transition from action vertex q_{i_w} to observation vertex o_{i_w} en route to q_{n+1}. That means $T_{i_w} \in I$ and, since u_{i_w} is a feasible action from q_{i_w}, the construction ensures that $w \in T_{i_w}$. □

These two lemmas lead directly to the following result.

Theorem 1: DECDM is NP-complete.

Proof: We need to show that DECDM is in both NP and NP-hard. For the former, we must be able to verify that a given instance of DECDM is a YES instance efficiently. For any positive instance, there is a solution no larger than W via Theorem 27 of [10], the argument therein carrying over when considering plans subject to some design cost $c(P) \leq k$. Such a plan, which is itself state-determined, can be used as a certificate. Confirming that this plan does indeed solve the planning problem is straightforward via backchaining and, since both the plan and W are state-determined, this takes polynomial time. For the latter, we must show a polynomial-time reduction from a known NP-complete problem to DECDM. Part 6 of Karp's 'Main Theorem' [15] establishes that SETCOVER is NP-complete, and Lemmas 1 and 2 show that the construction described above is indeed a reduction. □

B. Special cases that are also hard

Given the kind of hardness result expressed in Theorem 1, one reasonable follow-up question is to consider various kinds of restrictions to the problem, in hope that some natural or interesting special cases may yet be efficiently solvable.

However, notice that the cost function c used in the reduction is the counter design cost (recall Example 1). This leads immediately to several stronger results.

Corollary 1: DECDM, restricted to weighted sum cost functions (Example 2), is NP-complete.

Proof: Use the same reduction as in Theorem 1, but replace the counter design cost c with a weighted sum cost function in which each action has weight 1. □

Corollary 2: DECDM, restricted to monotone design cost functions (Example 5), is NP-complete.

Proof: The reduction in Theorem 1 uses counter design cost, which happens to be monotone. □

The reader will note that we have not yet referred back to Example 3 nor to Example 4; we revisit these in Section V.

C. Hardness of approximation

Another avenue of attack for hard problems is to try to find efficient approximation algorithms that can guarantee to provide solutions close to the optimal. Unfortunately, we can show that design cost minimization is hard even to approximate.

Theorem 2: For every $\varepsilon > 0$, OPTDM is NP-hard to approximate to within ratio $(1-\varepsilon)\ln n$.

Proof: Let $\varepsilon > 0$. Suppose, *a contrario*, that there exists a polynomial time approximation algorithm $\mathbf{A}$ that solves OPTDM with ratio $(1-\varepsilon)\ln n$. For a given instance (G, V_{goal}, c) of OPTDM, let $\text{OPT}(G, V_{\text{goal}}, c)$ denote the smallest cost, according to c, for a plan that solves (G, V_{goal}). Similarly, let $\mathbf{A}(G, V_{\text{goal}}, c)$ denote the cost of the output plan generated by algorithm $\mathbf{A}$. By construction, we have $\mathbf{A}(G, V_{\text{goal}}, c) \leq (1-\varepsilon)\ln n \ \text{OPT}(G, V_{\text{goal}}, c)$.

Under this assumption, we introduce the following polynomial-time approximation algorithm, called $\mathbf{B}$, for SETCOVER.

1) For a given instance (R, T) of SETCOVER, we construct a planning problem using the construction in Section IV-A.
2) We choose counter design cost for c, and execute algorithm $\mathbf{A}$ to find a plan whose design cost is within an $(1-\varepsilon)\ln n$ factor of optimal.
3) Then, using Lemma 2, we extract a set cover for R from the plan.

We write $\mathbf{B}(R, T)$ for the size of the set cover generated by algorithm $\mathbf{B}$ and $\text{OPT}(R, T)$ for the minimum coverage set size. Note that the size of this set cover is equal to the design cost for the plan, so that $\mathbf{B}(R, T) = \mathbf{A}(G, V_{\text{goal}}, c)$. We know also know, from Lemmas 1 and 2, that $\text{OPT}(G, V_{\text{goal}}, c) = \text{OPT}(R, T)$.

Thus, for sufficiently large n, we have

$$\begin{aligned} \mathbf{B}(R,T) &= \mathbf{A}(G, V_{\text{goal}}, c) \\ &\leq (1-\varepsilon)\ln n \, \text{OPT}(G, V_{\text{goal}}, c) \\ &= (1-\varepsilon)\ln n \, \text{OPT}(R, T). \end{aligned}$$

Therefore, we have a polynomial-time approximation algorithm $\mathbf{B}$ for SETCOVER with approximation ratio $(1-\varepsilon)\ln n$. Unless $P = NP$, this contradicts the known inapproximability result for SETCOVER due to Dinur and Steurer [5]. □

This proof also carries over to narrower classes of cost functions, just as the basic hardness proof in Section IV-B does, thus:

Corollary 3: For every $\varepsilon > 0$, OPTDM is NP-hard to approximate to within ratio $(1-\varepsilon)\ln n$, even when the design cost function is restricted to weighted sums (Example 2), or to monotone functions (Example 5).

D. Fixed parameter hardness

Another general way to cope with NP-hard problems is the *fixed-parameter tractability (fpt)* approach. The intuition of the approach is to try to identify features called parameters of an input instance, other than the problem size, that govern the hardness of a problem. Specifically, an NP-hard problem is fpt if there exists an algorithm to solve it in time $f(k)n^{O(1)}$, in which $f(\cdot)$ is some computable function and k is some parameter of the input instance [8], [21]. There is bad news on this front as well.

Lemma 3: DECDM, restricted to the counter design cost, and parameterized by the cost of the output plan, is not FPT under commonly held complexity-theoretic assumptions.[2]

Proof: Consider the construction in Section IV-A, denoted by Γ. We show that Γ is an fpt-reduction from SETCOVER, parameterized by the size of cover set, to DECDM, parameterized by the cost of output plan. The definition of fpt-reduction [8] has three conditions:

1) For all x, x is a positive instance of parameterized SETCOVER if and only if $\Gamma(x)$ is an positive instance of parameterized DECDM.
2) The construction Γ is computable by an fpt algorithm.
3) There exists a computable function from the value of parameter k to the value of parameter k', such that for any instance x of parameterized SETCOVER, the value of parameter k' in the instance outputted by Γ is less or equal to the value of parameter k in the instance x.

Lemma 1 and Lemma 2 confirm that the first condition is satisfied. In Section IV-A, we mentioned that the construction Γ takes polynomial time with respect to the input size, which is time $f(c)n^{O(1)}$, for some constant-valued function f and some constant c. Thus, the second condition is satisfied. Finally, according to Lemma 1 and Lemma 2, the value of parameter k is equal to the value of parameter k'. The identity function is obviously computable, so the third condition is satisfied. Thus, construction Γ is a fpt-reduction.

Then, suppose that DECDM parameterized by the size of counter design cost of output plan is FPT. We know that FPT is closed under fpt-reductions. That is, if a parameterized problem Q with parameter k, denoted by (Q, k), is reducible to a parameterized problem (Q', k') and $(Q', k') \in FPT$, then $(Q, k) \in FPT$ [8]. So, our supposition implies that SETCOVER parameterized by size of coverage set is in FPT, which is a contradiction—unless the entire fixed parameter hierarchy collapses [8]. □

Corollary 4: DECDM, parameterized by the design cost, is not FPT even when restricted to weighted sum cost functions (Example 2) or to monotone cost functions (Example 5), under common assumptions.[2]

V. DESIGN COST MINIMIZATION IN POLYNOMIAL TIME

Section IV presented hardness results of various kinds for several variations of the design cost minimization problem. Now we present a modicum of good news, in the form of results that show certain versions of the problem can indeed be solved in polynomial time, or are fixed parameter tractable.

A. Binary design and ordered action costs are efficiently solvable

It is useful to identify a class of cost functions which are amenable to a particular sort of decomposition.

Definition 6: A cost function c is *n-partition orderable* if there exists a partition of the action set U into mutually disjoint sets $U_1, U_2, \ldots, U_n$ which form an increasing sequence of costs, where $c\left(\bigcup_{i\in\{1,\ldots,m\}} U_i\right) < c\left(\bigcup_{i\in\{1,\ldots,m+1\}} U_i\right)$ for $1 \leq m < n$, and $\forall x \in U_{i+1}$, $c(U_i) < c(U_i \cup \{x\})$.

The intuition is that one must be able to split U into ordered level-sets with respect to costs. These allow easy solution.

Lemma 4: DECDM, restricted to n-partition orderable cost functions, can be solved in time $O(n\,|U|\,|V(G)|)$.

Proof: For a planning problem (G, V_{goal}) with an n-partition orderable cost function $c(P)$ there are $n+1$ possible outcomes. A straightforward procedure determines which: apply standard backchaining to (G, V_{goal}), but restricting consideration only to actions within U_1; if a solution is found it has cost $c(U_1)$ and this minimizes the cost. If no plan has been found at this point, backchaining can be continued, but now permitting actions from U_2 as well. A solution found at this juncture has cost $c(U_1 \cup U_2)$, which must minimize the cost. Otherwise, this procedure is repeated, adding U_{i+1} only after the search with U_i fails. If after U_n has been added no solution has been found, then no plan solves (G, V_{goal}). Since (G, V_{goal}) is state-determined, there are no more than $O(|U|\,|V(G)|)$ edges that one need examine. □

We now turn to the particular cost functions mentioned in Examples 3 and 4.

Corollary 5: DECDM, restricted to binary design cost functions (Example 3) is in P.

Proof: A binary design cost function $c_{A'}(\cdot)$ is a 2-partition orderable cost function with $U_1 = A'$ and $U_2 = U \backslash A'$. □

Corollary 6: DECDM, restricted to ordered design cost functions (Example 4) is in P.

Proof: An ordered cost function is an $|U|$-partition orderable cost function with $U_i = \{u_i\}$, $i \in \{1, \ldots, n\}$. □

B. Fixed parameter tractability of counter design cost

Though counter design cost (the cost of Example 1), is not n-partition orderable, nor is it FPT with respect to output plan cost (Lemma 3), we need not be inconsolably bleak.

Lemma 5: DECDM, with counter design cost, parameterized by size of the action space is in FPT.

Proof: Let (W, V_{goal}) be the given planning problem, and let $\lambda = |U|$, i.e., let it denote the size of action space of the problem. Consider the following simple algorithm: Enumerate 2^U. Then, for each subset of $U_i \in 2^U$, construct a planning problem with only actions from U_i. Checking whether each of these new planning problems can be solved or not with $c(P_{U_i}) \leq k$, which takes polynomial time in $V(W)$ because W is state-determined. Thus, this algorithm is FPT, because its running time is $2^\lambda n^{O(1)}$. □

VI. AN ALTERNATIVE FORMULATION VIA LABEL MAPS

The preceding has examined a concept of cost functions that model, via the set of actions that make an appearance in some plan, design choices concerned with actuators for robots executing that plan. Of course, the designer of a robot must select other resources as well and may make choices to balance or trade between a variety of factors. Most obviously one might ask about sensors also. Since planning problems described as p-graphs already possess observation edges to

[2]The particular assumption being that $W[2] \neq FPT$.

describe sensing transitions (i.e., observations from the set Y), this seems a fitting place to start. Unfortunately it is not meaningful to proceed along lines directly analogous to our foregoing treatment of actions because the model requires the robot's plan be ready to respond to whatever observations the world provides (N.B. the definition of 'safe' above); we cannot simply omit observations from our robot's plan.

One effective way to model choices of sensors is to presume that the observations specified as part of the planning problem describe the intrinsic ceiling for what could be perceived in that setting, rather than what the robot perceives in particular. Then a specific robot's sensors can be viewed as degradations of those observations. Thus, a description of a sensor might describe transformation of the set Y to deteriorate those ideal observations. Then the design-time sensor choice becomes a question of committing to a particular transformation. Previously in [10], we formalized the idea of these kinds of transformations that operate on the sets of labels borne by observation edges by introducing the notion of observation maps:

Definition 7 (observation map): An *observation map* is a function $h_y : Y \to 2^{Y'} \setminus \{\varnothing\}$ mapping from an observation space Y to a non-empty set of observations in a different observation space Y'.

By defining these as mappings from individual observations to sets of observations, some aspects of sensors can be expressed very directly. For instance, if sensor s cannot distinguish observations o_1 and o_2, we construct $h^{(s)}$ with $o_1 \overset{h^{(s)}}{\mapsto} \{o_1, o_2\}$ and $o_2 \overset{h^{(s)}}{\mapsto} \{o_1, o_2\}$. (Here, to help develop intuition, we have taken the simple special case where $Y' = Y$.)

The preceding begs the question of whether the same map-based perspective might not be generalized to be employed for actions as well. The idea is that, in so doing, sensor and actuator selection could both fit within the same scheme. Indeed, in what follows, we first adopt a map-oriented perspective but extend it to planning problems and then, by introducing an appropriate notion of cost, connect this with results established in the earlier sections. The process of generalization, starting with Definition 7, leads to the following two definitions:

Definition 8 (action map): An *action map* is a function $h_u : U \to 2^{U'} \setminus \{\varnothing\}$ mapping from an action space U to a non-empty set of actions in a different action space U'.

Definition 9 (label map): Now form a conglomerate, so that a *label map*, $h : U \cup Y \to (2^{U'} \cup 2^{Y'}) \setminus \{\varnothing\}$, combines an action map h_u and a sensor map h_y, thus:

$$h(\ell) = \begin{cases} h_u(\ell) & \text{if } \ell \in U \\ h_y(\ell) & \text{if } \ell \in Y \end{cases}.$$

Given a p-graph G and label map h, then by $h(G)$ we mean the p-graph constructed from G but, where previously an edge was carrying a set of elements X in G, it now has the set $\bigcup_{x \in X} h(x)$. It is in this sense that a label map is applied to a p-graph, mapping both actions and observation labels to different ones. Then, next, we can pose a design minimization problem in terms of a cost function that applies, not to actions or to plan length, but to the label map. Let $\mathcal{F}$ denote the space of all label maps, then we have that:

Definition 10: A *label map design cost*, or *map cost* for short, is simply a function $c_\ell : \mathcal{F} \to \mathbb{R} \cup \{\infty\}$.

Notice that map costs produce a value dependent on a label map, and label maps are applied globally across the entire graph. If a map cost measures the cardinality of the map's image, say, then it bounds the complexity of the resources used—for, as the cost is driven down and the bound becomes tight, those actions or sensing conditions that are never utilized will not be distinguished, so do not 'consume' unique values in the output. In cases such as these, the following problems model the robot designer's dilemma:

Decision Problem: Label map design minimization (LM-DECDM)

Input: A planning problem (G, V_{goal}), where G is state-determined, a map cost function c_ℓ, and a real number k.

Output: YES if there is a plan (P, P_{term}) and a label map h such that (P, P_{term}) solves $(h(G), V_{\text{goal}})$ and $c_\ell(h) \leq k$. NO otherwise.

Optimization Problem: Label map design minimization (LM-OPTDM)

Input: A planning problem (G, V_{goal}), where G is state-determined, and a map cost function c_ℓ.

Output: A label map h and a plan (P, P_{term}) such that the latter solves $(h(G), V_{\text{goal}})$ and so that $c_\ell(h)$ is minimal, or NONE indicating that no solutions exist.

The next theorem connects the map-based point of view with the earlier one, via the notion of monotone design costs. (For brevity, this is presented directly in terms of the optimization versions of the problems; it could also be expressed for decision variants.)

Theorem 3: Given any design minimization problem $\text{OPTDM}[(G, V_{\text{goal}}), c]$ where $c : 2^U \to \mathbb{R} \cup \{\infty\}$ is a monotone cost function, there exists a label map design minimization problem, $\text{LM-OPTDM}[(G', V_{\text{goal}}), c^c_\ell]$ with label map cost function $c^c_\ell : \mathcal{F} \to \mathbb{R} \cup \{\infty\}$, so that any output from the latter is either

(i) a solution $h^\star$ that attains an optimum equal to that for the solution of the former problem, i.e., $c^c_\ell(h^\star) = c(\mathcal{A}(P^\star, G))$; or

(ii) is NONE, and corresponds to the former problem having no solution (i.e., NONE).

Proof: We give an explicit construction. First, create a planning problem (G', V_{goal}) from (G, V_{goal}) by

1) including a new action, u_{fail}, in the action space;
2) adding an additional observation state v_{crash} to V_{y};
3) connecting every $v \in V_{\text{u}}$ to v_{crash} with an action edge labeled $\{u_{\text{fail}}\}$.

No plan that executes u_{fail} at any point reaches a state in V_{goal}. Furthermore, any label map applied to G' that conflates actions with u_{fail} ensures that plans using those actions cannot be solutions.

If we denote the action space of G' by $U' = U \cup \{u_{\text{fail}}\}$, then we specifically consider label maps $U \cup Y \to (2^{U'} \cup 2^{Y}) \setminus \{\varnothing\}$ next. Let us say that any such label map is *conformant* if (1) it maps all elements $y \in Y$ to $\{y\}$ and, further, (2) for each element $u \in U$, maps u to $\{u\}$ or $\{u_{\text{fail}}\}$. Conformant label maps leave the observations alone and may effectively disable some subset of actions.

Now, using the given c, construct a label map design cost function as follows:

$$c_\ell^c(h) = \begin{cases} c\left(h^{-1}\left[\mathfrak{U}\right]\right) & \text{if } h \text{ is conformant} \\ \infty & \text{otherwise} \end{cases},$$

with $\mathfrak{U} = \{\{u\} \mid u \in U\}$, and where we have used the brackets and -1 superscript to denote the preimage. The set-builder expression defines set $\mathfrak{U}$ to have all the elements of U, but so that every element u is replaced by its form as a singleton $\{u\}$. Observe that the original cost is invoked on all non-disabled actions because the preimage does not pullback u_{fail}.

Suppose that $P^\star$ is a solution to (G, V_{goal}) and the value it attains, $c(\mathcal{A}(P^\star, G))$, is minimal. Then it never uses actions $U \setminus \mathcal{A}(P^\star, G)$, so a conformant label map, h_1 say, sending those actions to u_{fail} produces a planning problem that $P^\star$ will solve as well. But $c(h_1^{-1}[\mathfrak{U}]) = c(h_1^{-1}[\mathcal{A}(P^\star, G)])$. And thus, $c_\ell^c(h^\star) \leq c_\ell^c(h_1) = c(\mathcal{A}(P^\star, G))$.

Now, if conformant label map h disables some actions when applied to a planning problem, any solution to that problem only makes effective use of some of the non-disabled ones. So if P solves the planning problem $(h(G'), V_{\text{goal}})$, then it must be that $h^{-1}[\mathfrak{U}] \supseteq \mathcal{A}(P, h(G'))$. Because c is monotone, $c_\ell^c(h) \geq c(\mathcal{A}(P, h(G')))$ and, since applying label map h only potentially disqualifies some actions, $c(\mathcal{A}(P, h(G'))) \geq c(\mathcal{A}(P^\star, G))$. The preceding, being true for all conformant label maps, holds for $h^\star$ in particular. So $c(\mathcal{A}(P^\star, G)) \leq c_\ell^c(h^\star) \leq c(\mathcal{A}(P^\star, G))$, yielding the desired result.

All that remains is the edge case when there is no solution: especially since ∞ is part of the construction of c_ℓ^c, to ensure a solution with value of ∞ is never produced when it ought not to be. If the OPTDM$[(G, V_{\text{goal}}), c]$ problem has no solution then all conformant label maps never have a solution. Consider the identity label map. It is conformant and there is no solution under it, as $(\mathrm{id}(G'), V_{\text{goal}}) = (G', V_{\text{goal}})$ which just corresponds to (G, V_{goal}). So, we must show that no non-conformant label map leads to a solvable planning problem. But this fact follows because conflating either actions or observations only makes the robot less capable. □

Lest the preceding should appear obvious, or a trivial trick with pushing the constraints around, or merely the usual objective/constraints duality, we make one point of clarification. For an OPTDM instance, the planning problem is fixed; the work of optimizing is searching over the space of plans to solve that single constant planning problem. But for LM-OPTDM, the planning problems themselves are altered (potentially becoming non-state-determined) and one asks for the existence of some plan on that new planning problem. That existential quantifier is, in a sense, absent from the OPTDM problem. Intuitively, what occurs is that the label map optimization problem hems in the set of actions that can be selected, and gradually tightens this confinement. It is able to do this because any slack from a particular label map, in the form of excess elements in the preimage set, either has no effect on the objective value (because the costs are the same under c with or without the excess), or they will be driven out by the optimization (i.e., by sending them to $\{u_{\text{fail}}\}$). The latter aspect hinges on the monotone property inherited from c.

One aspect of the p-graph formalism is that plans and planning problems appear to be very similar objects; in the past we have even suggested that this may be seen as a virtue, but in the case of the preceding theorem, this similarity unfortunately camouflages some of the game.

The upshot of Theorem 3 is that, since LM-DECDM subsumes DECDM, the hardness results from Section IV carry over to this new setting.

Corollary 7: LM-DECDM is NP-hard.

Proof: Combine Theorem 1 with Theorem 3, noting that the construction in the proof of Theorem 3 requires only polynomial time. □

Corollary 8: For every $\varepsilon > 0$, LM-OPTDM is NP-hard to approximate to within ratio $(1 - \varepsilon) \ln n$.

Proof: Combine Theorem 2 with Theorem 3, noting once again that the construction takes polynomial time, and that it increases the problem size by only a constant. □

VII. DISCUSSION

This paper complements the authors' previous papers; it is worth discussing why those papers, some of which also describe the hardness of aspects related to plans, do not quite capture an appropriate notion of design cost, the subject of this work.

States: In [25], motivated by memory constraints, we examine the hardness of a problem termed 'concise planning', which minimizes the number of states in a plan that solves some planning problem. The cardinality of the set of states is distinct from the total number of actions or, indeed, any function of $\mathcal{A}(P, W)$, though, minimization turns out to be difficult nevertheless.

Observations: The first paper to consider label map-like optimization, with a connection to sensor selection, was [28]. Therein we introduce representations and algorithms for examining whether some specific deterioration of an idealized sensor is destructive to task achievement. In that article we consider *filters* under application of observation maps; filters can be thought of as (junior) siblings of plans. Filters process streams of observations from the world but, unlike plans, they are passive in that there is no consideration of choices which may then influence subsequent observations. A filter maps sequences of observations to sequences of equivalence classes over observation sequences—here the phrase 'equivalence classes over observation sequences' is expressed conventionally, and more concisely, with the word 'state'.

For an observation map from Y to Y' and a filter on Y, we might ask whether one can construct a new filter that behaves indistinguishably from the given one, but operating

on Y' instead, with each observation transported under the map. Indistinguishable behavior means here that, given the same inputs, it produces the same outputs. If the observation map characterizes a sensor, then naturally we would be interested in the following optimization question: what is the simplest sensor that preserves the desired behavior? The decision form of this problem—where simplicity is measured via the cardinality of the observation map's image set—was established to be NP-hard in Theorem 5.5 of [29]. Corollary 7 in the current paper is, in a sense, a parallel result for planning to that prior result on filters. Though this discussion about filters hasn't provided hope for practical design-time optimization, still, it indicates that thinking about label maps provides an alternative direction from which to approach these sorts of problems. Indeed, a recent paper adopting this point of view is [33].

State determined forms: The idea of a state-determined form was developed in [28] and [10]. Its importance, prior to the present paper, had been mainly abstract and conceptual. The requirement of a state-determined planning problem, directly in the definition of DECDM, is important for the proof of the completeness result in Theorem 1. Note, though, that without that requirement, the proof of containment within NP-hard is preserved.

VIII. PROSPECTS

In light of the results presented, it seems likely that future computational tools to help the practical roboticist solve design problems ought not to aim at optimality, even in merely approximate terms. Indeed, as was emphasized in [22], there are several ways in which interactive tools might help guide a person in thinking about robot design problems, including approaches that can leverage the strength of people in bringing insights even to poorly framed or incompletely specified problems. These are, in the authors' view, important directions where, as yet, the little work that has been attempted remains preliminary and tentative.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Awards IIS-1526862, IIS-1527436, and IIS-1453652.

REFERENCES

[1] L. Carlone and C. Pinciroli, "Robot Co-design: Beyond the Monotone Case," in *Proceedings of IEEE International Conference on Robotics and Automation*, Montreal, Canada, May 2019.

[2] 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, Jan. 2017.

[3] ——, "Uncertainty in Monotone Co-Design Problems," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1556–1563, Feb. 2017.

[4] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of ACM Symposium on Theory of Computing*, Shaker Heights, OH., May 1971, pp. 151–158.

[5] I. Dinur and D. Steurer, "Analytical approach to parallel repetition," in *Proceedings of ACM Symposium on Theory of Computing*, New York, NY., May 2014.

[6] B. R. Donald, "On Information Invariants in Robotics," *Artificial Intelligence — Special Volume on Computational Research on Interaction and Agency, Part 1*, vol. 72, no. 1–2, pp. 217–304, Jan. 1995.

[7] I. Fitzner, Y. Sun, V. Sachdeva, and S. Revzen, "Rapidly prototyping robots: Using plates and reinforced flexures," *IEEE Robotics & Automation Magazine*, vol. 24, no. 1, pp. 41–47, Mar. 2017.

[8] J. Flum and M. Grohe, *Parameterized Complexity Theory*. Berlin Heidelberg New York: Springer, 1998.

[9] S. Fuller, E. Wilhelm, and J. Jacobson, "Ink-jet printed nanoparticle microelectromechanical systems," *Journal of Microelectromechanical Systems*, vol. 11, no. 1, pp. 54–60, Feb. 2002.

[10] S. Ghasemlou, F. Z. Saberifar, J. M. O'Kane, and D. Shell, "Beyond the planning potpourri: reasoning about label transformations on procrustean graphs," in *Proceedings of Workshop on the Algorithmic Foundations of Robotics*, San Francisco, CA, Dec. 2016.

[11] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane, "Computational design of robotic devices from high-level motion specifications," *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1240–1251, Oct. 2018.

[12] K. Hauser, "The minimum constraint removal problem with three robotics applications," *International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[13] A. M. Hoover and R. S. Fearing, "Fast scale prototyping for folded millirobots," in *Proceedings of IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008.

[14] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit, "Accomplishing high-level tasks with modular robots," *Autonomous Robots*, vol. 42, no. 7, pp. 1337–1354, Oct. 2018.

[15] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[16] L. A. Levin, "Universal search problems," *Problemy Peredachi Informatsii*, vol. 9, no. 3, pp. 115–116, 1973, in Russian. Reprinted in [17].

[17] ——, "Universal search problems," *Annals of the History of Computing*, vol. 6, no. 4, pp. 384–400, 1984, English translation of [16] by B. A. Trakhtenbrot.

[18] K. S. Luck, J. Campbell, M. Jansen, D. Aukes, and H. B. Amor, "From the lab to the desert: Fast prototyping and learning of robot locomotion," in *Proceedings of Robotics: Science and Systems*, Cambridge, MA, Jul. 2017.

[19] A. Mehta, N. Bezzo, P. Gebhard, B. An, V. Kumar, I. Lee, and D. Rus, "A design environment for the rapid specification and fabrication of printable robots," in *Springer Tracts in Advanced Robotics*, 2015, pp. 435–449.

[20] A. M. Mehta, J. DelPreto, K. W. Wong, S. Hamill, H. Kress-Gazit, and D. Rus, "Robot creation from functional specifications," in *Springer Proceedings in Advanced Robotics*, Nov. 2018, pp. 631–648.

[21] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*. New York: Oxford University Press, 2006.

[22] A. Q. Nilles, D. A. Shell, and J. M. O'Kane, "Robot Design: Formalisms, Representations, and the Role of the Designer," in *IEEE ICRA Workshop on Autonomous Robot Design*, Brisbane, Australia, May 2018, https://arxiv.org/abs/1806.05157.

[23] J. M. O'Kane, "A theory for comparing robot systems," Ph.D. dissertation, University of Illinois, Oct. 2007.

[24] J. M. O'Kane and S. M. LaValle, "On comparing the power of robots," *International Journal of Robotics Research*, vol. 27, no. 1, pp. 5–23, Jan. 2008.

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

[26] L. Paull, G. Severac, G. Raffo, J. Angel, H. Boley, P. Durst, W. Gray, M. Habib, B. Nguyen, S. V. Ragavan, S. Saeedi, R. Sanz, M. Seto, A. Stefanovski, M. Trentini, and H. Li, "Towards an ontology for autonomous robots," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Algarve, Portugal, Oct. 2012, pp. 1359–1364.

[27] V. Raman and H. Kress-Gazit, "Towards minimal explanations of unsynthesizability for high-level robot behaviors," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, Nov. 2013.

[28] F. Z. Saberifar, S. Ghasemlou, J. M. O'Kane, and D. Shell, "Set-labelled filters and sensor transformations," in *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, Jun. 2016.

[29] 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*, vol. 38, no. 2, pp. 236–259, Mar. 2019.

[30] F. Z. Saberifar, J. M. O'Kane, and D. A. Shell, "The hardness of minimizing design cost subject to planning problems," in *Proceedings of Workshop on the Algorithmic Foundations of Robotics*, Mérida, México, Dec. 2018.

[31] A. Schulz, C. Sung, A. Spielberg, W. Zhao, R. Cheng, E. Grinspun, D. Rus, and W. Matusik, "Interactive robogami: An end-to-end system for design of robots with ground locomotion," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1131–1147, 2017.
[32] B. Tovar, "Minimalist models and methods for visibility-based tasks," Ph.D. dissertation, University of Illinois at Urbana Champaign, 2009.
[33] Y. Zhang and D. A. Shell, "Abstractions for computing all robotic sensors that suffice to solve a planning problem," in *Proceedings of IEEE International Conference on Robotics and Automation*, Paris, France, May 2020.
[34] J. Ziglar, R. Williams, and A. Wicks, "Context-aware system synthesis, task assignment, and routing," 2017, arXiv:1706.04580.

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

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

Fatemeh Zahra Saberifar Dr. Saberifar is an Assistant Professor at the Faculty of Mathematical Sciences, with the Department of Computer Science, at Tarbiat Modares University. She holds the Ph.D. (2018) and M.Sc. (2010) degrees in Mathematics and Computer Science from the Amirkabir University of Technology, Tehran. Her interests include algorithmic robotics and computational geometry.