The hardness of minimizing design cost subject to planning problems

Fatemeh Zahra Saberifar[1], Jason M. O'Kane[2], and Dylan A. Shell[3]

[1] Amirkabir University of Technology, Tehran, Iran,
[2] University of South Carolina, Columbia SC, USA,
[3] Texas A&M University, College Station TX, USA

Abstract. Assuming one wants to design the most cost-effective robot for some task, how difficult is it to choose the robot's actuators? This paper addresses that question in algorithmic terms, considering the problem of identifying optimal sets of actuation capabilities to allow a robot to complete a given task. We consider various cost functions which model the cost needed to equip a robot with some capabilities, and show that the general form of this problem is NP-hard, confirming what many perhaps have suspected about this sort of design-time optimization. As a result, 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.

1 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 and adoption 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 resource constraints c, d, and e, 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 & Design Automation' (RSS'16), 'Minimality and Trade-offs in Automated Robot Design' (RSS'17), and 'Workshop on Autonomous Robot Design' (ICRA'18)[4]. The research is, naturally, 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 [5, 7, 11, 14, 15]; formal

[4] For a summative report of the first two workshops, presented at the third, see [18].

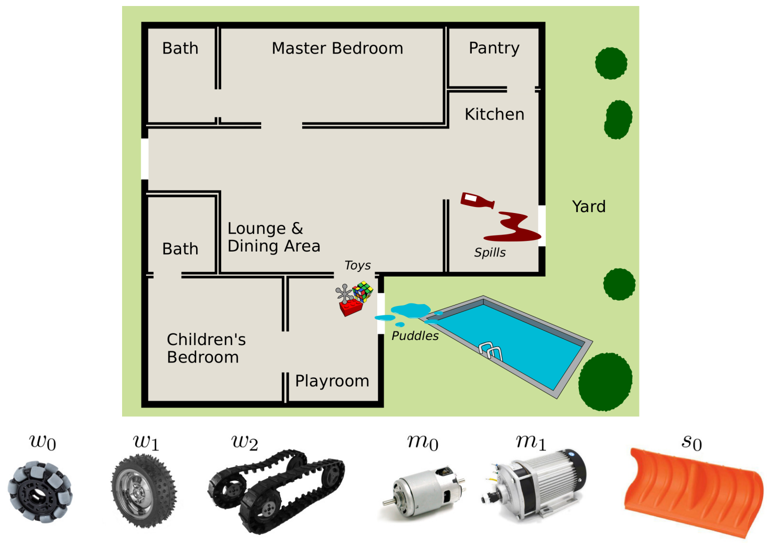

Fig. 1: A motivating example that illustrates how, for an otherwise simple planning problem, straightforward design constraints can quickly lead to complex considerations. [top] Suppose we want to design a cleaning robot that is able to navigate to any room in the house. [bottom] The robot's ability to navigate *spills*, *puddles*, and *toys* depends on the particular design choices we make about what resources with which the robot is to be equipped.

methods for (controller and hardware) synthesis of robots [12,15,16,23,25]; simulators and methods for interactive design [9]; compositional frameworks along with catalogs of components [1,2,22]; software for fault tracking and component-based identification [27]; and so on.

This paper deals with design problems that we believe many may already suspect to be difficult problems, but for which actual hardness results have not appeared in the literature. (At least to the authors, somewhat surprisingly!) Though many aspects of the hardness of planning have been examined, prior work has tended consider costs 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, while wheel w_1 operates fine in wet conditions, but 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 after a party, that is, under worst-case conditions. Here, multiple designs satisfy these task requirements while also respecting the component limitations (namely 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. One might imagine, for more complex problems with greater varieties of available components, that the complexity of this structure may scale exponentially.

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—reasonableness involves properties such as non-negativity. 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.

2 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 connections to the authors' prior work is postponed until Section 6, after presentation of the technical results.

Censi [1, 2] 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. [4,19,20,26]). 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. Indeed, at least when choosing the sets of actions with which to equip a robot, planning problems induce a multi-stage interaction between the various options, making it difficult to ensure optimality.

It is also worth pointing out the influential work of Hauser [10] 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.

3 Definitions and Problem Formulation

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

3.1 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 [8], 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_u *and* observation states V_y*, with $V = V_\mathrm{u} \cup V_\mathrm{y}$.*
2. *The multiset of edges $E = E_\mathrm{u} \cup E_\mathrm{y}$ is composed of* action edges E_u *and* observation edges E_y*.*
3. *Each action edge $e \in E_\mathrm{u}$ 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 $e \in E_\mathrm{y}$ 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 $V_0 \subseteq V_\mathrm{u}$ or $V_0 \subseteq V_\mathrm{y}$.*

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. Any execution can be traced in no more than one way in such p-graphs.

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 $V_{\text{goal}} \subseteq V(G)$. A* plan *is a p-graph P with termination region $P_{\text{term}} \subseteq V(P)$.*

We direct the reader to look again at 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. 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 specific start and goal pairs, 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 definitions of safe and of finite, which are somewhat tedious, referring the reader instead to [8].)

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

Definition 3. *A plan (P, P_{term}) with at least one execution* solves *a planning problem (W, V_{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_{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 $v \in P_{\text{term}}$, reach a vertex $w \in V(W)$ with $w \in V_{\text{goal}}$.*

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

3.2 Design costs

In this paper, we are interested specifically in plans that minimize a *design cost* function, which depends on which actions are utilized in a plan. 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 $c : 2^U \to \mathbb{R}$ *assigns a real number cost to each subset of U.*

Definition 5. *For any plan (P, P_{term}) that solves planning problem (W, V_{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 6 (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 7 (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 8 (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*

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

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 some additional action, not in A', is used.* ◇

Example 9 (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*

$$c(P) = \max \mathcal{A}(P).$$ ◇

Example 10 (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*

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

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 6), binary design cost (Example 8), and ordered action design cost functions (Example 9) are all monotone.

◇

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 the design cost of P is minimal.

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.

4 Hardness of Design Cost Minimization

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

4.1 The General Case

Our proof proceeds by reduction from the standard set cover problem, which is known to be NP-complete [13]:

Decision Problem: SetCover

Input: A universe set R with n elements, a set T including 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 problem, we construct an 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.
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 2 shows an example of this construction. Note that the time needed for this construction is polynomial in the input size.

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 enough elements have been selected from T 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 lemmata 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 $c(P) \le k$.*

Proof: Let $I \subseteq T$ denote a coverage set for R, which has $|I| \le k$. 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 $V_{\text{goal}} = P_{\text{term}} = \{q_{n+1}\}$. Clearly $c(P) = |I| \le k$. 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}). $\square$

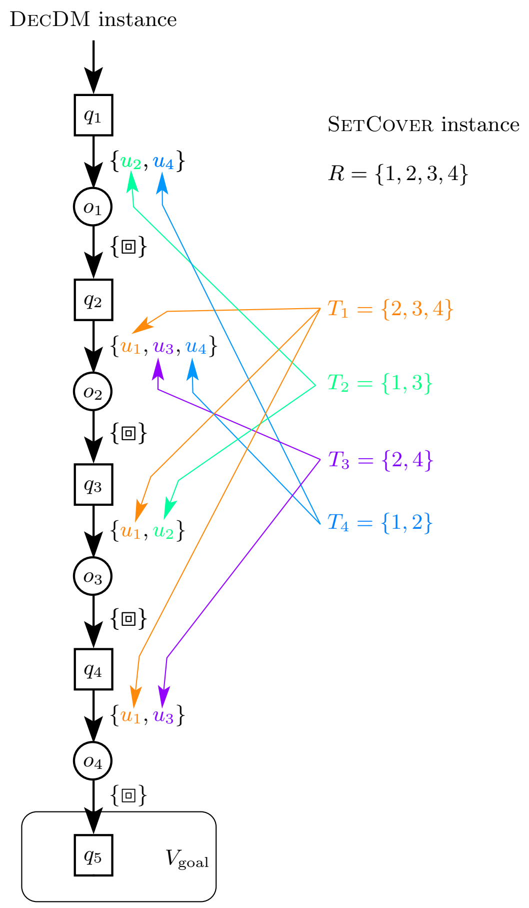

Fig. 2: An example of the construction of a DecDM instance from a SetCover instance. Given a set cover instance with $R = \{1, 2, 3, 4\}$, $T = \{\{2, 3, 4\}, \{1, 3\}, \{2, 4\}, \{1, 2\}\}$ and $k = 2$, we construct the planning problem shown on the left. Every subset of T that covers R corresponds 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.

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 $c(P) \le k$, then there exists a subset of T of size at most k that covers R.*

Proof: Let (P, P_{term}) be some plan with $c(P) \le k$ 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 $I = \big\{T_j \in T \mid j \in \{i_1, i_2, \ldots, i_n\}\big\}$ 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, $|I| = |\mathcal{A}(P)| = c(P) \le k$.

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 [8], the argument therein carrying over when considering plans subject to some design cost $c(P) \le k$. 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' [13] establishes that SetCover is NP-complete, and Lemmas 1 and 2 establish that the construction described above is indeed a reduction. □

4.2 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 6). This leads immediately to several stronger results.

Corollary 1 DecDM, restricted to weighted sum cost functions (Example 7), 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. $\square$

Corollary 2 DecDM, restricted to monotone design cost functions (Example 10), is NP-complete.

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

(The astute reader will note that we have not yet referred back to Example 8 nor Example 9; we revisit these in Section 5.)

4.3 Hardness of approximation

Another avenue of attack for a 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 4.1.
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 recover a set cover for R from the extracted 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 **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 [3]. □

This proof also carries over to narrower classes of cost functions, just as the basic hardness proof in Section 4.2 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 7), or to monotone functions (Example 10).

4.4 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 [6,17]. 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.*[5]

Proof: Consider the construction in Section 4.1, 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 [6] 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 4.1, 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.

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

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 means, 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 $(Q, k) \in FPT$ [6]. 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 [6]. $\square$

Corollary 4 DecDM, parameterized by the design cost, is not FPT even when restricted to weighted sum cost functions (Example 7) or to monotone cost functions (Example 10), under common assumptions.[5]

5 Design cost minimization in polynomial time

Section 4 presented a variety of 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.

5.1 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 11. *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. $\square$

We now turn to the particular cost functions mentioned in Examples 8 and 9.

Corollary 5 Any plan minimization problem with a binary design cost function is polynomial solvable.

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

Corollary 6 Any plan minimization problem with ordered action costs is polynomial solvable.

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

5.2 Counter design cost is fixed parameter tractable

Though counter design cost (the cost of Example 6), is not n-partition orderable, 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 subsets 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 $2^{\lambda} n^{O(1)}$. $\square$

6 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 [21], 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: A paper more obviously connected with design problems is [24], where we introduce representations and algorithms for examining whether some specific deterioration of an idealized sensor is destructive to task achievement. Determining the most aggressive non-destructive modification is hard.

In contrast, the present paper's focus is actions. While states are proportional to memory requirements, this is seldom the constraining factor in a robot design. Choice of sensors *is* an important design consideration, but the question there is usually whether they will be too noisy to be effective or not. In contrast, the mere inclusion of an action in a robot's repertoire usually demands some change in actuation. The connection between $\mathcal{A}(P, W)$ and cost of the robot is direct.

The idea of a state-determined form was developed in [24] and [8]. 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.

Future work could examine whether there are meaningful costs for observations that are "dual" to the design costs for actions. Imagine a design problem where the robot wishes to never receive some sensor reading: one might think that a robot should just omit any sensor which may receive such a signal, or that the robot could just forget the unwanted information. But these options may be impossible for particular contexts.

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

References

1. Censi, A.: A Class of Co-Design Problems With Cyclic Constraints and Their Solution. IEEE Robotics and Automation Letters 2(1), 96–103 (Jan 2017)
2. Censi, A.: Uncertainty in monotone co-design problems. IEEE Robotics and Automation Letters (February 2017)
3. Dinur, I., Steurer, D.: Analytical approach to parallel repetition. In: Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing. STOC '14, ACM (2014)
4. Donald, B.R.: On Information Invariants in Robotics. Artificial Intelligence — Special Volume on Computational Research on Interaction and Agency, Part 1 72(1–2), 217–304 (Jan 1995)
5. Fitzner, I., Sun, Y., Sachdeva, V., Revzen, S.: Rapidly prototyping robots: Using plates and reinforced flexures. IEEE Robotics & Automation Magazine 24(1), 41–47 (Mar 2017)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin Heidelberg New York (1998)
7. Fuller, S., Wilhelm, E., Jacobson, J.: Ink-jet printed nanoparticle microelectromechanical systems. J.of Microelectromechanical Systems 11(1), 54–60 (Feb 2002)
8. Ghasemlou, S., Saberifar, F.Z., O'Kane, J.M., Shell, D.: Beyond the planning potpourri: reasoning about label transformations on procrustean graphs. In: Proc. Workshop on the Algorithmic Foundations of Robotics (2016)

9. Ha, S., Coros, S., Alspach, A., Kim, J., Yamane, K.: Joint optimization of robot design and motion parameters using the implicit function theorem. In: Proceedings of Robotics: Science and Systems (2017)
10. Hauser, K.: The minimum constraint removal problem with three robotics applications. International Journal of Robotics Research 33(1), 5–17 (2014)
11. Hoover, A.M., Fearing, R.S.: Fast scale prototyping for folded millirobots. In: Proc. International Conference on Robotics and Automation (2008)
12. Jing, G., Tosun, T., Yim, M., Kress-Gazit, H.: An end-to-end system for accomplishing tasks with modular robots. In: Proceedings of Robotics: Science and Systems. AnnArbor, Michigan (June 2016)
13. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of computer computations, pp. 85–103. Springer (1972)
14. Luck, K.S., Campbell, J., Jansen, M., Aukes, D., Amor, H.B.: From the lab to the desert: Fast prototyping and learning of robot locomotion. In: Proceedings of Robotics: Science and Systems. Cambridge, Massachusetts (July 2017)
15. Mehta, A., Bezzo, N., Gebhard, P., An, B., Kumar, V., Lee, I., Rus, D.: A design environment for the rapid specification and fabrication of printable robots. In: Springer Tracts in Advanced Robotics. pp. 435–449 (2015)
16. Mehta, A.M., DelPreto, J., Wong, K.W., Hamill, S., Kress-Gazit, H., Rus, D.: Robot creation from functional specifications. In: Springer Proceedings in Advanced Robotics. pp. 631–648 (2018)
17. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, New York (2006)
18. Nilles, A.Q., Shell, D.A., O'Kane, J.M.: 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
19. O'Kane, J.M.: A theory for comparing robot systems. Ph.D. thesis, University of Illinois (October 2007)
20. O'Kane, J.M., LaValle, S.M.: On comparing the power of robots. International Journal of Robotics Research 27(1), 5–23 (January 2008)
21. O'Kane, J.M., Shell, D.: Concise planning and filtering: hardness and algorithms. IEEE Tran. on Automation Science and Engineering 14(4), 1666–1681 (Oct 2017)
22. Paull, L., Severac, G., Raffo, G., Angel, J., Boley, H., Durst, P., Gray, W., Habib, M., Nguyen, B., Ragavan, S.V., Saeedi, S., Sanz, R., Seto, M., Stefanovski, A., Trentini, M., Li, H.: Towards an ontology for autonomous robots. In: Proc. International Conference on Intelligent Robots and Systems. pp. 1359–1364 (2012)
23. Raman, V., Kress-Gazit, H.: Towards minimal explanations of unsynthesizability for high-level robot behaviors. In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (2013)
24. Saberifar, F.Z., Ghasemlou, S., O'Kane, J.M., Shell, D.: Set-labelled filters and sensor transformations. In: Proc. Robotics: Science and Systems (2016)
25. Schulz, A., Sung, C., Spielberg, A., Zhao, W., Cheng, R., Grinspun, E., Rus, D., Matusik, W.: Interactive robogami: An end-to-end system for design of robots with ground locomotion. The International Journal of Robotics Research 36(10), 1131–1147 (2017)
26. Tovar, B.: Minimalist Models and Methods for Visibility-based Tasks. Ph.D. thesis, University of Illinois at Urbana Champaign (2009)
27. Ziglar, J., Williams, R., Wicks, A.: Context-aware system synthesis, task assignment, and routing (2017), arXiv:1706.04580