Charting the trade-off between design complexity and plan execution under probabilistic actions

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

Abstract—**Practical robot designs must strike a compromise between fabrication/manufacture cost and anticipated execution performance. Compared to parsimonious designs, more capable (and hence more expensive) robots generally achieve their ends with greater efficiency. This paper examines how the roboticist might explore the space of designs to gain an understanding of such trade-offs. We focus, specifically, on design choices that alter the set of actions available to the robot, and model those actions as involving uncertainty. We consider planning problems under the Markov Decision Process (MDP) model, which leads us to examine how to relate the cost of some design to the expected cost of an execution for the optimal policies feasible with that design. The complexity of this problem—expressed via hardness in the fixed parameter tractability sense—depends on the number of actions to choose from. When that number is not negligible, we give a novel representation and an algorithm utilizing that structure that allows useful savings over naïve enumeration.**

I. Introduction

Each passing year brings with it a fresh harvest of technologies and innovations: novel sensors, new actuators, original mechanisms, and more capable computing devices, each offering some operational or economic advantage over the already available offerings. On the face of it, this seems to be unqualified good news for roboticists: more options afford the ability to make better choices. But alas and/or alack, the proliferation of choice can be an impediment. Because robots integrate mechanism with information processing, designing a robot for some task involves weighing a very wide range of considerations (e.g., from energetics: size, mass, and wear, to computational aspects: hardware, algorithms, data representations, etc.). Most of these considerations impinge on one another, so design necessarily entails the balancing of a variety of compromises. To a designer attempting to make informed choices for her robot, ever more options makes the design endeavor increasingly arduous. One potential solution is offered by tools that might help to 'navigate' the space of feasible options, so our designer might better understand the implications of various available choices.

To bring such tools closer to fruition the present paper offers algorithms that can help illuminate the trade-off between design complexity and anticipated robot performance. These algorithms are intended for use during the early phases that span the initial conception, exploration and consideration of alternate options. We adopt an abstract model which considers designs that differ in terms of the actions they are capable of executing. Actions themselves are regarded as having uncertain outcomes, treated as probabilistic transitions. Most directly, designs may differ in terms of physical actuators placed on the robot. Or, the actions available might represent sub-systems (perhaps utilizing low-level feedback) that enable execution of some action. Some combination can also be handled as well. The approach we describe allows comparison between robot designs capable of performing some action (say, to turn left) versus ones which cannot. More interestingly, since the transition dynamics are probabilistic, it also allows one to explore the effect of increasing precision, to determine whether paying additional cost to increase reliability of an action will actually manifest as a sufficiently large improvement in efficiency. Naturally, the effectiveness of such a change depends on the role of the action within the context of the other actions needed to reach the desired ends, that is, the relevant considerations are not entirely local.

As a basic example, consider the problem of designing a robot to deliver emergency supplies after flooding. Figure 1 shows a section of the map of Florida with Knight's Key (top left) and Boot Key (a larger landmass, at bottom). The robot starts at the position indicated with the yellow locator

F. Z. Saberifar is with the Department of Computer Sciences at Tarbiat Modares University, Tehran, Iran. D. A. Shell is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA. J. M. O'Kane is with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA. fz.saberifar@modares.ac.ir, dshell@tamu.edu, jokane@cse.sc.edu This material is based upon work supported by the NSF under Grants 1849249 & 1849291.

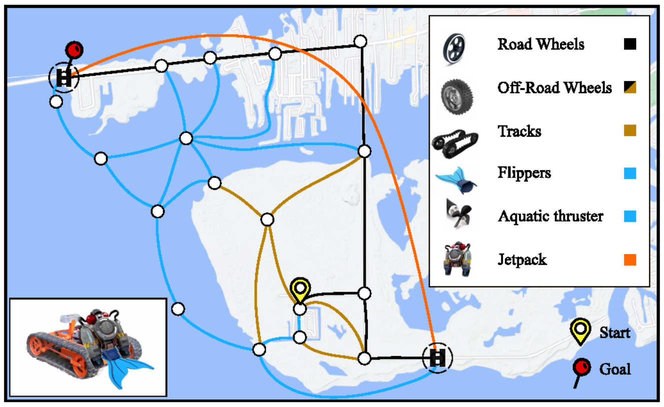


Fig. 1. Even simple settings can lead to complex trade-offs: Selecting the design for a robot to transport an item from the 'Start' to the 'Goal' by choosing the appropriate actuation. [Inset right:] These devices each have different purchase and integration costs, and different motion properties during execution, including power expended, range of availability (illustrated in summary form via colors), and reliability (treated as stochastic transitions). [Inset left:] An example of a highly-capable chimera-like tracked robot equipped with flippers and a jetpack, which happens to be a candidate in the design space.

and must arrive at the point marked with the red pin. As the right inset box shows, a variety of options for actuators exist, each with a different procurement price and resulting in different actions being available to the robot. The up-front cost to install an actuator may grant the ability to traverse whole sets of edges. Some of the actuators have broad applicability (like the various wheels), others (such as the 'jetpack') provide powerful capabilities but for a narrow niche. Certain choices might involve a difference in price for a difference performance: 'flippers' may allow slow reliable progress, while an 'aquatic thruster' unlocks the ability to move at greater speeds for comparable joules, albeit with more limited precision and hence poor fine manœuvrability. In our model, this reduced precision shows up as transition dynamics with greater entropy; energy expended in joules shows up as a cost. One might also replace a combination of specialized actuators with a single one that is applicable in the same circumstances, paying then only a single cost up-front. Still, a compromise between conditions likely loses some attractive properties: e.g., the more expensive combination of 'road wheels' and 'tracks' is more reliable off-road and faster on-road than the parsimonious design with just 'off-road wheels'.

These considerations are complex and raw intuition can be misleading. Algorithmic tools would be most useful to help provide insight and guide the design process. The contribution of this paper is to introduce such algorithmic tools (Sections V and VI) grounded in theoretical analysis of the problem's complexity (Section IV). We then investigate the effectiveness of those new tools (Section VII).

II. Related work

Recently, several robotics research groups have been exploring how algorithmic automation and computational methods can help improve the quality of the robot designs, can ease the design process, or shed light on specific elements that make the problem especially complex [22]. A core line of research is concerned with the notion of co-design: Censi [7], [8] developed the mathematical theory of co-design problems, which considers the relation between resource consumption and the functionality of robots. Carlone and Pinciroli [6] consider co-design when selecting robot system modules with a given cost to maximize performance. They solve the problem using binary linear programming. The term 'co-design' has also been used more broadly (see [3], [21]) when different aspects that play a role are considered jointly.

Aspects of design automation have additionally been focused on considerations of fabrication [19], rapid-prototyping [2], [14], interactive design [10], [11], [5]. Also, there have been approaches to structured knowledge representation that can help with design problems [17], [18], [13].

Most closely related is the authors' prior work [25], which examines a notion of designs similar to this paper, but the planning problems in that paper are under worst-case assumptions. The present paper makes two separate modifications, both of which change the setting rather more significantly than the authors originally anticipated. The first is to introduce a notion of execution cost. The key consideration in [25] is a decision problem, *viz.* whether a set of actuators suffices, or whether dropping some actuator (or gadget) will incapacitate the robot. Instead, the present study examines a Pareto front with differing costs. Secondly, we now consider probabilistic transitions. These confer a meaningful notion of expected performance, permitting the impact of some action's precision to be considered quantitatively.

Because part of the problem we tackle involves estimating value functions for Markov Decision Problems using an interval approach (and terminating early), we point out that this is similar in form to Haddad and Monmege [12], though their setting has only a single objective. There is substantial prior work on multi-objective MDPs (see [24], [9]), though that work considers costs which are not design costs. Also relevant, albeit more loosely, is literature dealing with hardness results for MDPs (see, for instance, [4]), and asynchronous approaches to value iteration [1], and reinforcement learning [26].

III. Problem statement

A. Markov decision processes and action restrictions

We are interested in sequential decision-making problems that can be modeled as Markov decision processes (MDPs). We begin with three standard definitions [**?**], [26].

Definition 1. *An* MDP *is a tuple* $(X, x_0, U, p, c, \gamma)$ *with*

1) *a finite nonempty* state space X,
2) *an* initial state $x_0 \in X$,
3) *a finite nonempty* action space U,
4) *a* transition probability function $p : X \times U \times X \to [0, 1]$, *in which* $p(x, u, x')$ *denotes the probability of transitioning to state* x' *upon selecting action* u *at state* x,
5) *a* cost function $c : X \times U \to \mathbb{R}$, *in which* $c(x, u)$ *denotes* single-step execution cost *of executing action* u *at state* x, *and*
6) *a* discount factor $\gamma \in [0, 1)$.

An MDP models a discrete time sequential decision-making scenario. Each execution begins at the initial state x_0. At each stage k, the robot is at state x_k and executes action u_k, whereupon the system transitions to a new state x_{k+1} drawn according to the probability distribution $p(x_k, u_k, \cdot)$. During this transition, the robot incurs a single stage cost $c(x_k, u_k)$. The actions selected by the robot are governed by a policy.

Definition 2. *For an MDP* $(X, x_0, U, p, c, \gamma)$, *a* policy *is a function* $\pi : X \to U$.

Intuitively, the policy describes the action selected by the robot from each state in the state space, so that $u_k = \pi(x_k)$.

The robot's objective is to minimize the costs incurred as it executes the policy, subject to discounting, which ensures convergence across executions that are, in principle, infinite.

Definition 3. *For an MDP* $(X, x_0, U, p, c, \gamma)$ *and a policy* π, *the* execution cost *of that policy is*

$$\mathbf{e}(\pi) := \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c(x_k, u_k)\right].$$

A policy is an optimal policy *if its execution cost is minimal among all policies.*

In this paper we will have occasion to consider families of MDPs formed by eliminating certain actions. The next definition makes this idea more precise.

Definition 4. *For an MDP $M = (X, x_0, U, p, c, \gamma)$ and an action set $U' \subseteq U$, the* restriction of M to U', *denoted $M(U')$, is an MDP identical to M except that its action space is U' rather than U. That is, $M(U') = (X, x_0, U', p', c', \gamma)$, in which the transition probability and step-stage cost function are $p' : X \times U' \times X \to [0, 1]$ and $c' : X \times U' \to \mathbb{R}$ respectively, with $p'(x, u, x') = p(x, u, x')$ and $c'(x, u) = c(x, u)$ for all x and x' in X, and all u in U'.*

Definition 5. *For an MDP $M = (X, x_0, U, p, c, \gamma)$ a* value function *is a scalar function $V_M : X \to \mathbb{R}$, and an* optimal value function *satisfies*

$$V^\star_M(x) = \min_{u \in U} \sum_{x' \in X} p(x, u, x') \left[c(x, u) + \gamma V^\star_M(x')\right]. \qquad (1)$$

Value functions V^-_M, V^+_M with $V^-_M(x) \leq V^\star_M(x) \leq V^+_M(x)$ for all $x \in X$ are termed, respectively, lower- and upper-bounding value functions.

B. Design cost

Next, we adapt the notion of design cost introduced in our prior work [25] from its original formulation based on worst-case reasoning to the present stochastic setting.

Definition 6. *A* design cost function $d : 2^U \to \mathbb{R} \cup \{+\infty\}$ *assigns an extended real number to each subset of the action space U.*

The key distinction, which contrasts with traditional plan costs, is that design cost depends on *which* actions may be executed by a given policy, rather than on how often those actions may be executed on any particular run of the system.

Definition 7. *For a given policy π, the* operative actions $\mathcal{A}(\pi)$ *are the actions associated with at least one state. That is, $\mathcal{A}(\pi) := \cup_{x \in X}\{\pi(x)\}$. The* design cost *of a policy π is the design cost of its operative actions, $\mathbf{d}(\pi) := d(\mathcal{A}(\pi))$.*

A property which holds often in practice is that adding additional capabilities to a robot will not decrease its cost, and these are called monotone design costs.

Definition 8. *A design cost function is* monotone *if, for any sets $U_1 \subseteq U$ and $U_2 \subseteq U$, we have*

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

We restrict our attention to monotone design cost functions in this paper.

Note that Definition 8 admits a variety of cost functions of varying complexity. For example, one useful family of design cost functions characterizes the design choice as one of selecting from a set of *gadgets* with which to equip the robot. Each gadget is defined by the design cost of choosing it —from which the overall design cost may be derived additively—along with a set of actions it enables. This is appropriate for the example in Figure 1.

C. Optimizing Execution cost and design cost

We now have the requisite elements in place to state the central algorithmic problem. The exposition above introduces two distinct measures by which a policy may be evaluated: its execution cost (Definition 3) and its design cost (Definition 7). Notice, however, that these measures can be expected to be in tradeoff with one another: In general, modifications to a policy that decrease its design cost by making some actions inoperative can, by virtue of restricting the set of available actions, increase the execution cost.

Thus, the root problem here is one of multi-objective optimization, and we are interested in policies that are *Pareto optimal* [28], in the sense that no other policy improves both the design cost and the execution cost.

Problem: Optimal Design-/Execution-Cost Policies (ODECP)

Input: An MDP M and a design cost function d.
Output: The set of Pareto-optimal policies for M and d.

For purpose of complexity analysis in Section IV we can also cast the problem as a decision problem.

Decision Problem: Feasible Design-/Execution-Cost Decision (FDECP)

Input: An MDP M, a design cost function d, a design cost bound $\bar{d}$, and an execution cost bound $\bar{e}$.
Output: YES if there exists a policy π for M with design cost at most $\bar{d}$ and execution cost at most $\bar{e}$; NO otherwise.

IV. Hardness and fixed-parameter tractability

Before attending to data structures and algorithms for these problems, a detour into their computational complexity will be instructive.

Two proofs will use reduction from the following standard problem, known to be NP-complete [16].

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 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.

Theorem 1. *FDECP is NP-hard.*

Proof. The construction used in Theorem 1 of [25] uses a reduction from SetCover; the same construction can be re-tooled for use here. (Only the required modifications are given here.) Pick $\gamma = \frac{1}{2}$. For a SetCover instance with the set size $|R|$, a state space of size $|R| + 2$ works: we have a 'goal' state, $x_{|R|+1}$, and a 'crash' state x_0. Actions are constructed analogously with that in [25]: If action a should connect state x_i to x_{i+1}, then p sets only that probability to one, and zero for all other x_j, $j \in \{0, \ldots, |R| + 1\} \setminus \{i + 1\}$. If action a doesn't depart from state x_i, have it transition with probability one to the 'crash' state x_0. All actions loop at state x_0. Assign cost $c(x_i, u) = 0$ for all u and every state x_i except x_0. Put $c(x_0, u) = 1$ for all u. Then, an optimal policy that reaches the goal will have expected cost 0; any other policy reaches x_0 after the first step but no later than the $|R|^{\text{th}}$-step, thus has expected cost that is $\in (\frac{1}{2}, 1]$. Hence, we choose $\bar{d}$ to be the cardinality of the desired cover (k in [25]), and $\bar{e}$ to be $\frac{1}{2}$. $\square$

Lemma 1. *Given MDP $(X, x_0, U, p, c, \gamma)$, an optimal policy can be found in polynomial time.*

Proof. The policy can be found using linear programming [20], which has a solution in polynomial time (e.g., [15]). In fact, Papadimitriou & Tsitsiklis [23] proved the stronger result that solving for the optimal policy is P-complete. $\square$

Theorem 2. *FDECP, parameterized by the size of the action space, is fixed-parameter tractable (FPT).*

Proof. Given MDP $M = (X, x_0, U, p, c, \gamma)$, choose as parameter the size of the action space, i.e., let $\lambda = |U|$. The obvious algorithm works: enumerate the set of MDPs via the restriction $\{M(V) \mid V \in 2^U \setminus \{\varnothing\}\}$. For each, evaluate the design cost $d(M(V))$. If it is less than or equal to $\bar{d}$, then construct an optimal policy, and evaluate its expected cost to see if it is no more than the execution cost bound $\bar{e}$, if so, answer YES. If all have been enumerated and none found, answer NO. Since construction of the optimal policy takes polynomial time via Lemma 1, this algorithm is FPT because its running time is $2^{\lambda} n^{O(1)}$. $\square$

Informally, the upshot of Theorems 1 and 2 is that, though the problem in general is computationally challenging (unless P = NP), that challenge is primarily concentrated in the number of actions in our MDPs.

Another, alternative take on the hardness is that even if we have the optimal MDP value function (say, via an oracle) then we will still have a difficult problem. Consider the following.

Problem: Optimal Design Cost Policy (ODCP)
Input: An MDP $M = (X, x_0, U, p, c, \gamma)$, an optimal value function $V^{\star}_M$, and a design cost function d.
Output: A policy π with $\mathbf{e}(\pi) = V^{\star}_M(x_0)$ such that $\mathbf{d}(\pi)$ is minimal.

The decision problem is as follows.

Decision Problem: Feasible Design Cost Policy (FDCP)
Input: An MDP $M = (X, x_0, U, p, c, \gamma)$, an optimal value function $V^{\star}_M$, and a design cost function d, and a design cost bound $\bar{d}$.
Output: YES if there exists a π for M with $\mathbf{e}(\pi) = V^{\star}_M(x_0)$ such that $\mathbf{d}(\pi) \le \bar{d}$; NO otherwise.

Theorem 3. *ODCP is NP-hard.*

Proof. To prove NP-hardness of ODCP, it is sufficient to prove that its decision problem (FDCP) is NP-complete. So, we need to prove FDCP $\in$ NP and all NP problems are reducible to FDCP. For the first part, if given a putative π claimed to correspond to a YES instance, we can check that the given action at each state is indeed a u which minimizes (1). This verification takes $O(|X|)$ time.

For the second part, we present a polynomial reduction, again, from SETCOVER: given an instance (R, T, k), construct an instance of FDCP, $(M, V^{\star}_M, d, \bar{d})$, as follows:

- We form an MDP $M = (X, x_0, U, p, c, \gamma)$ with state space $X = \{x_0, x_1, \ldots, x_{|R|}, x_g\}$, initial state x_0, and $U = \{u_1, u_2, \ldots, u_{|T|}\}$.
  For each $u \in U$, we define $\forall k \in \{0, g\}, p(x_0, u, x_k) = 0$, and $\forall k' \in \{1, \ldots, |R|\}, p(x_0, u, x_{k'}) = \frac{1}{|R|}$. To define p, for each u_i in U :
  $k \in \{1, \ldots |R|\}, p(x_0, u_i, x_k) = \frac{1}{|R|}$;
  $j \in \{1, \ldots, |R|\}$, $x \in X \setminus \{x_j, x_g\}, p(x_j, u_i, x) = 0$,
  $p(x_j, u_i, x_g) = \mathbf{1}_{T_i}(j)$, $p(x_j, u_i, x_j) = 1 - \mathbf{1}_{T_i}(j)$;
  $x \in X, p(x_g, u_i, x) = \mathbf{1}_{\{x_g\}}(x)$.
  Then, define to c, for each u_i in U:
  $c(x_0, u_i) = c(x_g, u_i) = 0$,
  $k \in \{1, \ldots |R|\}$, $c(x_k, u_i) = 1 - \mathbf{1}_{T_i}(k)$.
- Take $\gamma = \frac{1}{2}$.
- Compute $V^{\star}_M$ from M.
- Take as design cost $d(A) = |A|$.
- Set the design cost bound $\bar{d} = k$.

(Above, $\mathbf{1}_Y(\cdot)$ is the indicator function for set Y.) In light of Lemma 1, all steps in this construction take polynomial time. For any instance (R, T, k), consider the FDCP $(M, V^{\star}_M, d, \bar{d})$. Observe that $V^{\star}_M(x_0) = 0$ because SETCOVER stipulates that every element in R is covered by at least one element T_j, so each state $\{x_1, \ldots, x_{|R|}\}$ can take some action with cost zero.

If $I \subseteq T$ with $|I| \le k$ covers R, then for each $i \in \{1, \ldots |R|\}$, the policy π selects any action u_j where $T_j \in I$ covers the element in R corresponding to i. For x_0 and x_g, reuse one of the actions already used elsewhere. Such a policy π takes an action of cost 0 at each state (when at state $x_\ell \in \{x_1, \ldots, x_R\}$ it performs an action u_j with $\ell \in T_j$, so $c(x_\ell, u_j) = 1 - \mathbf{1}_{T_j}(\ell) = 0$). This is an optimal expected execution cost, since all costs are non-negative. But then $\mathbf{d}(\pi) \le k = \bar{d}$;

Conversely, if we have a policy π with $c(\pi) = V^{\star}_M(x_0) = 0$, and $\mathbf{d}(\pi) \le \bar{d}$, then collect all the actions $\{u_{j_1}, u_{j_2}, \ldots u_{j_n}\} = \cup_{i \in \{1, \ldots |R|\}} \{\pi(x_i)\}$, where know $j_n \le k$ because these are the operative actions. A zero cost action must be prescribed at every state because, if it did not, then $c(\pi) > \frac{1}{2} \cdot \frac{1}{|R|} \cdot 1 > 0$. Thus, sets $T_{j_1}, T_{j_2}, \ldots T_{j_n}$ cover R. $\square$

V. THE LATTICE OF BOUNDS DATA STRUCTURE

In spite of the discouraging news of the previous section, we turn now to the practical question of solving ODECP. This section describes a data structure called a lattice of bounds, which represents partial information about the design and execution costs achievable with various sets of actions. We describe its structure, its operation, and several important invariants it maintains. This data structure forms that basis of the algorithm in the next section.

Definition 9. *A* lattice of bounds *for an MDP $M = (X, x_0, U, p, c, \gamma)$ is a directed graph, in which each of the* *finitely-many vertices v is labeled with a set of actions $U_v \subseteq U$ and value functions $V_v^- : X \to \mathbb{R}$ and $V_v^+ : X \to \mathbb{R}$.*

The following two invariants form a connection between the lattice of bounds and Pareto optimal solutions we seek.

Invariant 1. *In a lattice of bounds, for each edge $v \to w$, $U_v \supset U_w$. That is, each edge represents a parent-child relationship under which the parent has access to a strict superset of the actions available to the child.*

Invariant 2. *In a lattice of bounds for MDP $M = (X, x_0, U, p, c, \gamma)$, at each vertex v, V^- and V^+ are lower and upper bound value functions, respectively, for the restricted MDP $M(U_v)$.*

These invariants are important because they establish a connection to our objective of finding the Pareto front.

Lemma 2. *For any lattice of bounds L in which Invariants 1 and 2 hold, and any policy π with execution cost $\mathbf{e}(\pi)$ and design cost $\mathbf{d}(\pi)$, if there exists some vertex v in L for which $V_v^+(x_0) \le \mathbf{e}(\pi)$ and $d(U_v) \le \mathbf{d}(\pi)$, then π is not a Pareto optimal policy.*

Proof. Vertex v provides a direct counterexample to the possibility of Pareto optimality of v. □

To create a lattice of bounds, it suffices to construct a collection of one or more vertices and to initialize the V^- and V^+ functions for each to (even very optimistic or pessimistic) lower- and upper-bounds for the true value. One safe way to do this, for each v_{u_k}, is to initialize $V_{u_k}^-$ and $V_{u_k}^+$ values to $\left(\min_{x\in X, u\in U_{u_k}} c(x,u)\right)/(1-\gamma)$, and $\left(\max_{x\in X, u\in U_{u_k}} c(x,u)\right)/(1-\gamma)$, respectively.

Lemma 3. *This initial lattice of bounds satisfies Invariants 1 and 2.*

Other operations may mutate an existing lattice of bounds, generally toward tighter bounds on the optimal value functions.

Operation 1. *For a vertex and a state, perform one Bellman update (the atomic step of value iteration) on V^- or V^+ at one state x. That is:*

$$V_v^-(x) \leftarrow \min_{u\in U_v} \sum_{x'\in X} p(x,u,x') \left[c(x,u) + \gamma V_v^-(x')\right], \text{ or}$$

$$V_v^+(x) \leftarrow \min_{u\in U_v} \sum_{x'\in X} p(x,u,x') \left[c(x,u) + \gamma V_v^+(x')\right], \text{ resp.}$$

The rationale for the previous operation follows from this lemma.

Lemma 4. *Steps of value iteration monotonically decrease (resp. increase) the value function when initialized from an upper-bounding (resp. lower-bounding) value function.*

Proof. The standard and, indeed, some modified methods — possessing superior performance — converge monotonically. A self-contained and explicit proof appears in [27]; for the argument showing that the property holds for a subset of states (or single state) see [1, Chapt. 7]. □

Operation 2. *For an edge $v \to w$ and a state x, assign*

$$V_v^+(x) \leftarrow \min\left(V_v^+(x), V_w^+(x)\right).$$

Rationale: As v has more actions, the costs at v never exceed those at w. Thus, if the value at w is at most $V_w^+(x)$, then the value at v can be at most $V_w^+(x)$ as well.

Operation 3. *For an edge $v \to w$ and a state x, assign*

$$V_w^-(x) \leftarrow \max\left(V_w^-(x), V_v^-(x)\right).$$

Rationale: As w has fewer actions, the costs at w can never be less than those at v. If the value at v is at least $V_v^-(x)$, then the value at w can be no less than $V_v^-(x)$ as well.

Definition 10. *For a lattice of bounds for MDP $M = (X, x_0, U, p, c, \gamma)$ with vertices V, a set of actions $U' \subseteq U$ is* unrepresented *if there is no vertex $v \in V$ such that $U_v = U'$. For an unrepresented set, a* bracketing pair *consists of two vertices $u, w \in V$ such that $U_u \supsetneq U' \supsetneq U_w$.*

Operation 4. *For an unrepresented set U' with bracketing pair u and w: add the vertex v with actions $U_v = U'$, and add all edges $p \to v$ with $U_p \supsetneq U'$, and add all edges $v \to q$ with $U' \supsetneq U_q$, and remove any edges $r \to s$ with $U_s \supsetneq U' \supsetneq U_r$. Set $V_v^- = V_u^-$ and $V_v^+ = V_w^+$.*

Lemma 5. *Operations 1–4 maintain Invariants 1 and 2.*

VI. An algorithm for computing the Pareto front

The concept of a lattice of bounds forms the core datastructure underlying our algorithm. Actually, the preceding definition invariants offer plenty of scope for a variety of different approaches and, in what follows, we describe one effective means for combining these elements. (The next section will provide evidence for the claim of effectiveness by revisiting the motivating scenario, as a case study).

The algorithm is composed of several elements, each of which periodically mutates the lattice bounds.

0. Initialization: For MDP $M = (X, x_0, U, p, c, \gamma)$, we construct an initial lattice of bounds comprising $|U|$ vertices. The action sets for these vertices are seeded with each distinct action u and then greedily expanded to contain a maximal set of actions that have the same design cost as u individually. The V^- and V^+ bounds for each vertex are initialized as described above.

1. Improving Execution Cost Estimates at the Widest Gaps: We maintain a priority queue of states, ordered by $\delta V_v(x) = V_v^+(x) - V_v^-(x)$, for all the states across all the vertices. This allows the state with the largest discrepancy between lower and upper bounds on the value to be selected and improved using Operation 1 (one iteration for V^- and another for V^+). After this, $\delta V_v(x)$ is re-computed (it may have decreased) and the element in the priority queue updated. Operations 2 & 3 are then applied recursively to propagate the available information up and down the lattice.

2. Improving Execution Cost Estimates Globally: Similarly, we also maintain a cyclic queue of all of the states at all of the vertices, using it to apply Operation 1 to them in a round-robin fashion. This ensures that progress continues to be made even when the widest gaps occur at states where value iteration cannot yet improve the bounds because the bounds at potential next states are as yet too weak.

3. Removing vertices from consideration: If at some point we have two vertices v and w with the property that

$$d(v) \geq d(w) \quad \text{and} \quad \forall x \in X,\ V_v^-(x) \geq V_w^+(x)$$

with at least one of the inequalities strict, then v is dominated and there is enough information to declare that it will remain so even with further improvement of the cost estimates. (Recall Lemma 2.) We can thus remove those elements of v from the priority queue and cyclic queue used for execution cost updates. (It remains in the lattice of bounds because it may be useful for the vertex creation step.)

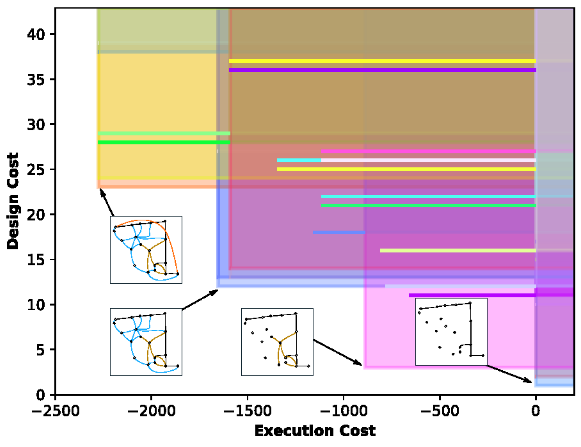
4. Creating vertices: New vertices are created by selecting an existing vertex, adding a new action, and then expanding that action set greedily to reach a plateau of the design cost function, just as in step 0.

The preceding text described a collection of mostly independent strategies for investing computational effort to make a lattice of bounds more accurately reflect the true Pareto front. How are these elements combined into a working algorithm? The initialization is done once at the start, obviously. Steps *1–3* can be executed under a variety of policies. We currently employ a simple approach where each of Steps *1–3*. operate at each iteration of a main loop. Step *4*, which expands the lattice into unexplored portions of the design space, is triggered with the $\delta V_u(x_0)$ bound in Step *1* falling below a chosen threshold ε; when the bounds for the currently-represented vertices begin to converge, the time is right to explore further.

The algorithm terminates either when no maximal (in the sense of being able to add actions without increasing design cost, as in Steps 0 and 4) action sets remain to be represented, or when a timeout expires.

VII. Case studies

In order to demonstrate and evaluate our python implementation of the algorithm, we encoded the problem shown in Figure 1 as an ODECP. Space constraints require that omit specific quantitative details and parameters of the model, but we emphasize a few aspects. Firstly, each arc in the graph shows up as an action. The design cost was constructed so that for a set of actions, the addition of any extra action which is already possible using the gadgets employed in the set does not increase the design cost. This satisfies the requirements for monotonicity. The specific procurement and per-action costs model the approximate ordering one would expect (e.g., the jetpack is two orders of magnitude more dear than the wheels, and one order more than the thruster). We modeled some actions as unreliable (like the thruster and flippers) by having some probability for failure, which the transition dynamics cause to incur cost with no change in state.



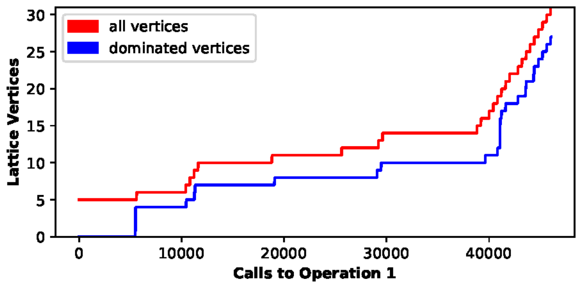


Fig. 2. [top] The Pareto front visualized for the scenario in Figure 1. Four choices dominate the other options. The inset figures show the actions selected for four designs: the bottom right, with lowest design cost has only 'road wheels'; the bottom center one has 'road wheels' and 'off-road wheels'; bottom left has 'flippers' and 'off-road Wheels'; the design at top has 'flippers', 'off-road wheels', the 'jetpack'. [bottom] A performance plot showing the progression of the algorithm as it executed.

Figure 2 [top] gives a visual summary of the final Pareto front as charted by the algorithm. Notice, in particular, the horizontal lines: these are vertices where V^- and V^+ have not needed to be iterated until convergence—they are detected as dominated and computation is saved by stopping early. The plot in Figure 2 [bottom] gives a progression of the implementation functioning across time, using calls to Operation 1, the most frequent operation, as the independent variable. The shrinking $\delta V_u(\cdot)$ will fall below ε, triggering Operation 4 periodically (visible in the diagram as vertical steps where additional vertices are introduced). The plot also shows the increase in number and proportion of dominated vertices.

VIII. Conclusion

This paper tackled the problem of designing robots and forming plans for those robots, in contexts where both execution cost (i.e. time, energy, etc.) and design costs (i.e. fixed costs to equip the robot with certain capabilities) are germane, and the tradeoffs between these costs must be explored. The particular setting involved uncertainty about the outcomes of actions. We established several computational complexity results and introduced a data structure and accompanying algorithm the solve that problem.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[2] N. Bezzo, A. Mehta, C. D. Onal, and M. T. Tolley, "Robot makers: The future of digital rapid design and fabrication of robots," *IEEE Robotics & Automation Magazine*, vol. 22, no. 4, pp. 27–36, 2015.

[3] G. Bravo-Palacios, A. D. Prete, and P. M. Wensing, "One robot for many tasks: Versatile co-design through stochastic programming," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1680–1687, 2020.

[4] T. Brázdil, K. Chatterjee, V. Forejt, and A. Kučera, "Trading performance for stability in markov decision processes," *Journal of Computer and System Sciences*, vol. 84, pp. 144–170, 2017.

[5] B. Canaday, S. Zapolsky, and E. Drumwright, "Interactive, iterative robot design," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1188–1195.

[6] L. Carlone and C. Pinciroli, "Robot co-design: beyond the monotone case," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3024–3030.

[7] 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.

[8] ——, "Uncertainty in monotone co-design problems," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1556–1563, 2017.

[9] K. Chatterjee, R. Majumdar, and T. A. Henzinger, "Markov decision processes with multiple objectives," in *Annual symposium on theoretical aspects of computer science*. Springer, 2006, pp. 325–336.

[10] R. Desai, Y. Yuan, and S. Coros, "Computational abstractions for interactive design of robotic devices," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1196–1203.

[11] S. Ghasemlou, J. M. O'Kane, and D. A. Shell, "Delineating boundaries of feasibility between robot designs," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 422–429.

[12] S. Haddad and B. Monmege, "Interval iteration algorithm for MDPs and IMDPs," *Theoretical Computer Science*, vol. 735, pp. 111–131, 2018.

[13] H. Hu, D.-y. Liu, and X.-y. Du, "Semi-automatic hardware design using ontologies," in *ICARCV Control, Automation, Robotics and Vision Conference*, vol. 2, 2004, pp. 792–797.

[14] M. Indri, F. Sibona, and L. O. Russo, "P&p-standard architecture to enable fast software prototyping for robot arms," in *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 721–728.

[15] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984, pp. 302–311.

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

[17] O. Karrenbauer, S. Rader, and T. Asfour, "An ontology-based expert system to support the design of humanoid robot components," in *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 1–8.

[18] E. E. Karsak, "Expert decision system for robot selection," *Wiley Encyclopedia of Computer Science and Engineering*, pp. 1–11, 2007.

[19] C. Liu, W. Yan, and A. Mehta, "Computational design and fabrication of corrugated mechanisms from behavioral specifications," 2020.

[20] A. S. Manne, "Linear programming and sequential decisions," *Management Science*, vol. 6, no. 3, pp. 259–267, 1960.

[21] M. Morelli and M. Di Natale, "Control and scheduling co-design for a simulated quadcopter robot: A model-driven approach," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2014, pp. 49–61.

[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] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[24] D. Scheftelowitsch, P. Buchholz, V. Hashemi, and H. Hermanns, "Multi-objective approaches to markov decision processes with uncertain transition parameters," in *Proceedings of EAI International Conference on Performance Evaluation Methodologies and Tools*, 2017, pp. 44–51.

[25] D. A. Shell, J. M. O'Kane, and F. Z. Saberifar, "On the design of minimal robots that can solve planning problems," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 876–887, 2021.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. MIT press, 2018.

[27] D. White, "Monotone value iteration for discounted finite markov decision processes," *Journal of mathematical analysis and applications*, vol. 109, no. 2, pp. 311–324, 1985.

[28] P. Yu, "Multiple criteria decision making: Five basic concepts," *Handbooks in Operations Research and Management Science*, vol. 1, pp. 663–699, 1989.