Beyond the planning potpourri: reasoning about label transformations on procrustean graphs

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

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

Abstract. We address problems underlying the algorithmic question of automating the co-design of robot hardware in tandem with its apposite software. Specifically, we consider the impact that degradations of a robot's sensor and actuation suites may have on the ability of that robot to complete its tasks. Expanding upon prior work that addresses similar questions in the context of filtering, we introduce a new formal structure that generalizes and consolidates a variety of well known structures including many forms of plans, planning problems, and filters, into a single data structure called a procrustean graph. We describe a collection of operations on procrustean graphs (both semantics-preserving and semantics-mutating), and show how a family of questions about the destructiveness of a change to the robot hardware can be answered by applying these operations. We also highlight the connections between this new approach and existing threads of research, including combinatorial filtering, Erdmann's strategy complexes, and hybrid automata.

1 Introduction

The process of designing effective autonomous robots—spanning the selection of sensors, actuators, and computational resources along with software to govern that hardware—is a messy endeavor. There appears to be little hope of fully automating this process, at least in the short term. There would, however, be significant value in *design tools* for roboticists that can manipulate partial or tentative designs, in interaction with a human co-designer.

To that end, this paper lays a formal foundation for answering questions about the relationship between a robot's hardware, specifically its sensors and actuators, and its ability to complete a given task. Interesting questions arise when one considers how modifications to a given robot's capabilities alter the planning and estimation efforts that robot must undertake. This paper develops theoretical tools that we believe to be helpful for thinking about such aspects.

This material is based upon work supported by the National Science Foundation under Grants IIS-1527436, IIS-1526862, IIS-0953503, IIS-1453652.

Prior work by the current authors [10] made some preliminary progress in this direction by considering a limited form of *sensor map*, which describes a coarsification of a sensor model. This paper strengthens and extends those results in several ways.

1. We contribute, in Section 2, a new general representation called a *procrustean graph*[4] that unifies several previously distinct conceptual classes of object. This representation is constructive, in that it can be used to instantiate a data-structure from which various questions can be posed and addressed concretely. We detail, in Section 3, how this representation can be used to reason about planning problems and their solutions.
2. We extend, in Section 4, the existing notion of a sensor map to model modifications to both sensors and actuators, including modifications that introduce uncertainty directly. This generalized map is called *label map*. We show how to decide whether a label map is *destructive* in the sense of preventing the achievement of a previously-attainable goal. (Prior results used a much more restrictive notion of destructiveness, in which a map was considered destructive if it engendered any change to the robot's behavior; our new results deem non-destructive any map under which the robot can still reach its goal, even if its strategy for doing so is forced to change.)

The dénouement of the paper includes a review of related work interleaved with a discussion of the outlook for continued progress (Section 5) and some concluding remarks (Section 6).

2 Procrustean graphs

2.1 Basic definitions

The work in this paper is connected with a variety of design-time concerns that can be represented with a single formal construct: a graph with set-labelled transitions.

Definition 1 (p-graph). *A* procrustean graph (p-graph) *is an edge-labelled bipartite directed graph in which*

1. *the finite vertex set, of which each member is called a* state*, can be partitioned into two disjoint parts, called the* action vertices V_{u} *and the* observation vertices V_{y}*, with* $V = V_{\mathrm{u}} \cup V_{\mathrm{y}}$*,*
2. *each edge* e *originating at an action vertex is labeled with a set of actions* $U(e)$ *and leads to an observation vertex,*
3. *each edge* e *originating at an observation vertex is labeled with a set of observations* $Y(e)$ *and leads to an action vertex, and*

[4] Named for Procrustes (Προκρούστες), son of Poseidon, who, according to myth, took the one-size-fits-all concept to extremes.

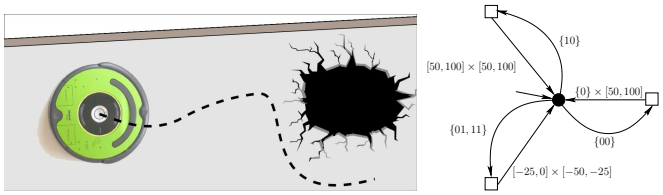

Fig. 1. [left] A differential drive robot with sensors for obstacles, both positive (walls) and negative (holes). [right] An example p-graph that models behavior in which the robot follows a wall while avoiding negative obstacles. This graph, and those that follow, have solid circles to represent elements of V_{u}, and empty squares for V_{y}. The arcs are labelled with sets; those that leave the central vertex have two digits, the first digit is '1' iff the wall is detected by the IR sensor on the left-hand side; the second digit is '1' iff the downward pointing IR sensor detects a cliff. The actions, on the edges leaving squares, represent sets of left and right wheel velocities, respectively.

4. *a non-empty set of states V_0 are designated as* initial states*, which may be either exclusively action states ($V_0 \subseteq V_{\mathrm{u}}$) or exclusively observation states ($V_0 \subseteq V_{\mathrm{y}}$).*

The general intuition is to encode an interaction between an agent or robot (which selects actions) and its environment (which dictates the observations made by the robot). The definition is intentionally ecumenical in regard to the nature of that interaction, because we intend this definition to serve as a starting point for more specific structures which, once specific context and semantics are added, lead to special cases that represent particular (and familiar) objects involved planning, estimation, and the like.

Example 2 (wheels, walls, and wells). *A small example p-graph, intended to illustrate the basic intuition, appears in Figure 1. It models a Roomba-like robot that uses single-bit wall and cliff sensors to navigate through an environment. Action states are shown as unshaded squares; observation states are shaded circles. Action labels are subsets of* $[0, 500] \times [0, 500]$*, of which each element specifies velocities for the robot's left and right drive wheels, expressed in mm/s. Observations are bit strings of length 2, in which the left bit is the output of the wall sensor, and the right bit is the output of the cliff sensor.* ◇

Note that, to keep the model amenable to direct algorithmic manipulation, we require that a p-graph consist of only finitely many states. The labels for each edge, either $U(e)$ or $Y(e)$, need not be finite sets. We instead rely on the availability of some simple operations on labels such as unions, intersection, and membership tests; full details appear in [10].

2.2 Some things you know about are actually secretly p-graphs

Several well-known kinds of objects can be recognizably expressed as p-graphs.

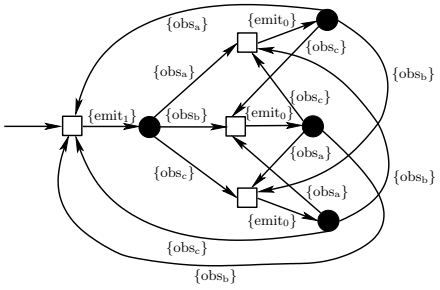

Fig. 2. The 'agents together' filter devised by Tovar *et al.* [12] expressed as a p-graph. The emit_0 action indicates that the agents are separated by a beam, and emit_1 indicates that the agents are together.

Example 3 (Combinatorial filters). *As formalized by LaValle [7], combinatorial filters are discrete expressions of estimation problems. Associated with each state in such a filter is a specific output. Filters can be cast as p-graphs by having observations and observation transitions exactly as in the filter, but with action vertices having only a single out-edge which has U as a singleton set bearing the output (which we label emit_x for various outputs x). Figure 2 shows a canonical example in which the property of interest is whether two agents in an annulus-shaped environment with three beam sensors are apart or not.* ◇

Example 4 (Schoppers's universal plans). *For observable domains, a universal plan [11] describes an appropriate action for each circumstance that a robot might find itself in. As such, they can be cast as p-graphs in a straightforward way: The p-graph has a single observation vertex, with one uniquely-labeled out-edge corresponding to each world state, and one action state for each of the distinct available actions.* ◇

Example 5 (Erdmann-Mason-Goldberg-Taylor plans). *Several classic papers [5,6,8] find policies for manipulating objects in sensorless (or near sensorless) conditions. The problems are usually posed in terms of a polygonal description of a part; while the solutions to such problems are sequences of actions. Such plans can be expressed as p-graphs in which actions (e.g., a squeeze-grasp or a tray tilt at a particular orientation) or ranges of acceptable actions are interleaved with a special ε which constitutes the sole element in all $Y(e)$. Figure 3 shows an example of such a plan. Of particular note is the fact that that plan exhibits an unexpected dimension of nondeterminism: it indicates sets of allowable actions, rather than a single predetermined one, at each step. Also of note is that the graphs of knowledge states generally searched to produce such plans are p-graphs themselves.* ◇

Example 6 (Nondeterministic graphs). *Recent work by Erdmann [3,4] encodes planning problems using finite sets of states, along with nondeterministic actions represented as collections of edges 'tied' together into single actions. One might convert such a graph to a p-graph by replacing each group of action edges with an observation node, with an outgoing observation edge for each edge constituting the original action.* ◇

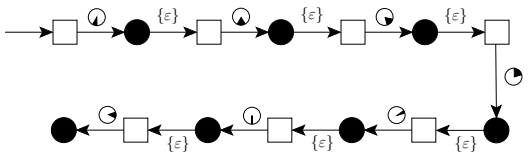

Fig. 3. A plan for orienting an Allen wrench via tray tilting, expressed as a p-graph. Action edges are labeled with sets of azimuth angles for the tray. There is a single dummy observation, ε. This plan is shown as Fig. 2 in Erdmann and Mason [5].

The intent in these examples is to illustrate that p-graphs form a general class that unifies, in a relatively natural way, a number of different kinds of objects that have been studied over a long period of time. The particular constraints applied in each case impose certain kinds of structure that proved useful in the original context. Our objective in this paper is to treat p-graphs, in a general sense, as first-class objects, suitable for manipulation by automated means.

2.3 Properties of p-graphs

At the most general level, we can view a p-graph as an implicit definition of a *language* of strings in which actions and observations alternate. The following definitions make this precise.

Definition 7 (event). *An* event *is an action or an observation.*

Definition 8 (transitions to). *For a given p-graph G and two states $v, w \in V(G)$, an event sequence $e_1 \cdots e_k$* transitions in G from v to w *if there exists a sequence of states $v_1, \ldots, v_{k+1}$, such that $v_1 = v$, $v_{k+1} = w$, and for each $i = 1, \ldots, k$, there exists an edge $v_k \xrightarrow{E_k} v_{k+1}$ for which $e_k \in E_k$.*

Note that v and w need not be distinct: for every v, the empty sequence transitions in G from v to v. Longer cycles may result in non-empty sequences of states that start at some v and return.

Definition 9 (valid). *For a given p-graph G and a state $v \in V(G)$, an event sequence $e_1 \cdots e_k$ is* valid *from v if there exists some $w \in V(G)$ for which $e_1 \cdots e_k$ transitions from v to w.*

Observe that the empty sequence is valid from all states in any p-graph.

Definition 10 (execution). *An* execution *on a p-graph G is an event sequence valid from some start state in $V_0(G)$.*

The preceding definitions prescribe when a sequence is valid on a p-graph, placing few restrictions on the sets involved. There are several instances of 'choices' recognizable as forms of non-determinism: (i) there may be multiple elements in V_0; (ii) from any $v \in V_\mathrm{u}$ some u may be an element in sets on multiple outgoing action edges; (iii) similarly, from any $w \in V_\mathrm{y}$ some y may qualify for multiple outgoing observation edges.

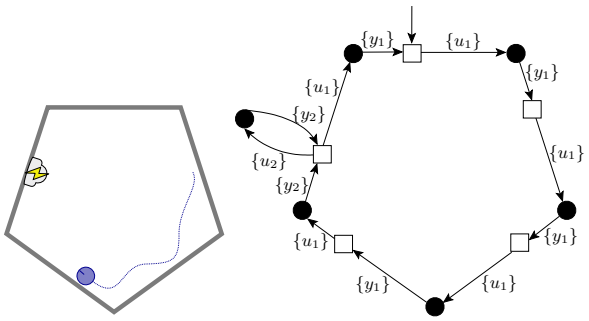

Fig. 4. [left] A robot wanders around a pentagonal environment; the segment with the lightning-bolt contains a battery charger. [right] A p-graph model of this world.

Example 11 (Pentagonal world). *Figure 4 presents concrete realizations of several of the preceding definitions in a single scenario. A robot moves in a pentagonal environment. Information—at least at a certain level of abstraction—describing the structure of the environment, operation of the robot's sensors, its actuators, and their inter-relationships is represented in the p-graph associated with the scenario. Both filtering and planning can be posed as problems on the p-graph representation in terms of valid event sequences.* ◇

Thus far, other than the potentially infinite action- and observation-alphabets and their specific alternating structure, the definitions are close to classic formal language theory. The set of executions on G are taken to comprise $L(G)$, the language induced by p-graph G. Also, since $V_0(G) \subseteq V(G)$, the language of every p-graph includes the empty sequence.

2.4 Properties of pairs of p-graphs

In Section 3, we model both planning problems and plans themselves as p-graphs. The next definitions will be helpful for formalizing the relationships between those two p-graphs.

Definition 12 (joint-execution). *An event sequence $e_1 \cdots e_k$ is a* joint-execution *on a pair of p-graphs G and W if it is an execution in both G and W.*

These are the executions that make use of labels and transitions in both p-graphs, and the joint-executions make up the intersection of their respective languages.

Definition 13 (finite on). *A p-graph G is* finite on *p-graph W if there exists an integer k that bounds the length of every joint-execution of G and W.*

There are two types of p-graph: those that start by producing an action, and those that start by receiving an observation.

Definition 14 (akin). *Two p-graphs are* akin *if both have initial states that are action states, or have initial states that are observation states.*

A stronger notion of the relationship between two p-graphs is safety, which has no simple analogy to a property on the languages involved.

Definition 15 (safe). *P-graph G is* safe *on p-graph W if G is akin to W and if, for every joint-execution $e_1 \cdots e_k$ on G and W, the following property holds: For every state $v \in V(G)$ reached by $e_1 \cdots e_k$ in G, and every state $w \in V(W)$ reached by $e_1 \cdots e_k$ in W, from every possible initial state, we have*

1. *if v is an action state, then for every action u associated with an edge in G originating at v, there exists an edge e in W originating at w, for which $u \in U(e)$, and*
2. *if v is an observation state, then for every observation y associated with an edge in W originating at w, there exists an edge e in G originating at v, for which $y \in Y(e)$.*

The intuition is that if P is safe on Q, then P never executes any action that is not allowed by Q, and is always prepared to respond to any observation that may arrive if chosen by Q.

2.5 Basic constructions of new p-graphs from old

We have shown that several existing structures can be described by p-graphs but the question remains as to why they ought to be expressed as such. We give two examples of constructive operations, applicable to the p-graph structure, which produce new p-graphs as output.

Definition 16 (union of p-graphs). *The* union *of two p-graphs U and W, each akin to the other, denoted by $U \uplus W$, is the p-graph constructed by including both sets of vertices, both sets of edges, and with initial states equal to $V_0(U) \cup V_0(W)$.*

The intuition is to form a graph that allows, via the nondeterministic selection of the start state, executions that belong to either U or W.

Definition 17 (state-determined). *A p-graph P is in a* state-determined presentation *if $|V_0(P)| = 1$ and from every action vertex $u \in V_u$, the edges $e_u^1, e_u^2, \ldots, e_u^\ell$ originating at u bear disjoint labels: $U(e_u^i) \cap U(e_u^j) = \varnothing, i \neq j$, and from every observation vertex $y \in V_y$, the edges $e_y^1, e_y^2, \ldots, e_y^m$ originating at y bear disjoint labels: $Y(e_y^i) \cap Y(e_y^j) = \varnothing, i \neq j$.*

The intuition is that in a p-graph in a state-determined presentation it is easy to determine whether an event sequence is an execution: one starts at the unique initial state and always has an unambiguous edge to follow. We note, however, that the p-graph with a state-determined presentation for some set of executions need not be unique.

Given any p-graph it is possible to construct a new p-graph that has the same set of executions on it, but which is in a state-determined presentation. We only sketch the procedure as it is a generalization of the observation-only algorithm presented in detail in [10]. The basic idea is a forward search that performs a powerset construction on the input p-graph. We begin by constructing a single state to represent the "superposition" of all initial states, and push that onto a empty queue. While the queue has elements, remove a vertex and examine the edges leaving the set of vertices associated with it in the original input p-graph. The labels on those edges are *refined* by constructing a partition of the set spanned by the union of the labels in a way that the subsequent sets of states in the input p-graph is clear. Edges are formed with the refined sets connecting to their target vertices, constructing new ones as necessary, and placing these in the queue.

3 Plans and planning problems

While a p-graph induces a structured state space, further enrichment is needed in order to talk meaningfully about plans and planning problems.

Definition 18 (planning problem). *A* planning problem *is a p-graph G equipped with a* goal region $V_{\text{goal}} \subseteq V(G)$.

The idea is that for a pair that make up the planning problem, the p-graph describes the setting and form in which decisions must be made, while the V_{goal} characterizes what must be achieved.

Definition 19 (plan). *A* plan *is a p-graph P equipped with a* termination region $V_{\text{term}} \subseteq V(P)$.

The intuition is that the out-edges of each action state of the plan show one or more actions that may be taken from that point—if there is more than one such action, the robot selects one nondeterministically—and the out-edges of each observation state show how the robot should respond to the observations received from the environment. If the robot reaches a state in its termination region, it may decide to terminate there and declare success, or it may decide to continue on normally. We can now establish the core relationship between planning problems and plans.

Definition 20 (solves). *A plan* (P, P_{term}) solves *the 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 reaches P_{term} 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 $w \in V_{\text{goal}}$.*

Example 21 (Charging around and in the pentagonal world). *We can construct a planning problem from the p-graph of Figure 4, along with a goal region consisting of only the fully-charged state reached by action u_2. Figure 5*

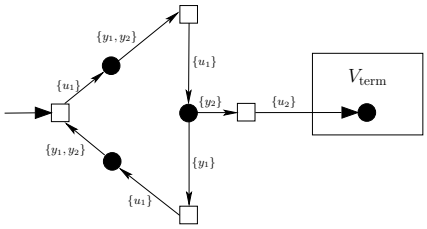

Fig. 5. A plan that directs the robot of Figure 4 to its charging station, along a hyperkinetic (that is, exhibiting more motion than is strictly necessary) path.

shows a plan that solves this problem. However, that plan, a cycle of three actions, is a bit surprising since it will take the robot along three full laps around its environment before terminating. The existence of such bizarre plans motivates our consideration of homomorphic plans, which behave rather more sensibly, in Section 3.1. ◇

Given a plan (P, P_{term}) and a planning problem (W, V_{goal}), we can decide whether (P, P_{term}) solves (W, V_{goal}) in a relatively straightforward way. First, we convert both P and W into state-determined presentations, using the technique described in Section 2.5. Then, the algorithm conducts a forward search using a queue of ordered pairs (v, w), in which $v \in V(P)$ and $w \in V(W)$, beginning from the (unique, due to Definition 17) start states of each. For each state pair (v, w) reached by the search, we can test each of the properties required by Definition 20:

- If P and W are not akin, return false.
- If (v, w) has been visited by the search before, then we have detected the possibility of returning to the same situation multiple times in a single execution. This indicates that P is not finite on W. Return false.
- If v and w fail the conditions of Definition 15 (that is, if v is missing an observation that appears in w, or w omits an action that appears in v) then P is not safe on W. Return false.
- If v is a sink state not in P_{term}, or w is a sink state not in V_{goal}, then we have detected an execution that does not achieve the goal. Return false.
- If $v \in P_{\text{term}}$ and $w \notin V_{\text{goal}}$, then the plan might terminate outside the goal region. Return false.

If none of these conditions hold, then we continue the forward search, adding to the queue each state pair (v', w') reached by a single event from (v, w). Finally, if the queue is exhausted, then—knowing that no other state pair can be reached by any execution—we can correctly conclude that (P, P_{term}) does solve (W, V_{goal}).

It may perhaps be surprising that both planning problems and plans are defined by giving a p-graph, along with a set of states at which executions should end. We view this symmetry as a feature—not a bug—in the sense that it clearly illuminates the duality between the robot and the environment with which it interacts. Observations can be viewed as merely "actions taken by nature" and vice versa. At an extreme, the planning problem and the plan may be identical:

Lemma 22 (self-solving plans). *If P is a p-graph which is acyclic and the set of its sink nodes is V_{sink}, then (P, V_{sink}) is both a planning problem and a plan. Moreover, (P, V_{sink}) solves (P, V_{sink}).*

Proof: The plan is obviously finite and safe on itself. As joint-executions are essentially just executions, the result follows from the fact that every execution on P either reaches an element of V_{sink}, or is the prefix of one that does. □

We have described, in Definitions 16–17, operations to construct new p-graphs out of old ones. We can extend these in natural ways to apply to plans.[5]

Definition 23 (∪-product of plans). *The* ∪-product *of plans (U, V_{goal}) and (W, V'_{goal}), with U and W akin, is a plan $(U \uplus W, V_{\text{goal}} \cup V'_{\text{goal}})$.*

Theorem 24 (state-determined ∪-products). *Given two plans (P, P_{term}) and (Q, Q_{term}), with P and Q akin, construct a new plan whose p-graph, denoted R, is the expansion of $P \uplus Q$ into a state-determined presentation. Recall that the expansion means that every state $s \in V(R)$ corresponds to sets $P_s \subseteq V(P)$ and $Q_s \subseteq V(Q)$ of states in the original p-graphs (either possibly empty, but not both). Define a termination region R_{term} as follows:*

$$R_{\text{term}} := \{ s \in V(R) \mid (P_s \neq \varnothing \wedge P_s \backslash P_{\text{term}} = \varnothing) \vee (Q_s \neq \varnothing \wedge Q_s \backslash Q_{\text{term}} = \varnothing) \}.$$

Then (R, R_{term}) is equivalent to $(P \uplus Q, P_{\text{term}} \cup Q_{\text{term}})$, in the sense that they have identical sets of executions on them, and moreover that any problem solved by the former is also solved by the latter.

Proof: The result follows directly from the executions that underly the state-determined expansion, and the definition of the ∪-product. □

This result illustrates how the state-determined expansion is useful — it permits a construction that captures the desired behavioral properties and, by working from a standardized presentation, can do this directly by examining states rather than posing questions quantified over the set of executions.

3.1 Homomorphic solutions

The following are a subclass of all solutions to a planning problem.

Definition 25 (homomorphic solution). *For a plan (P, V_{term}) that solves planning problem (W, V_{goal}), consider the relation $R \subseteq V(P) \times V(W)$, in which $(v, w) \in R$ if and only if there exists a joint execution on P and W that can end at v in P and in w in W. A plan for which this relation is a function is called an* homomorphic solution.

The name for this class of solutions comes via analogy to the homomorphisms — that is, structure-preserving maps — which arise in algebra. In this context, an homomorphic solution is one for which each state in the plan corresponds to exactly one state in the planning problem.

[5] . . . and—via the symmetry between Definitions 18 and 19—in the same stroke, to planning problems, though in this paper we'll use these operations only on plans.

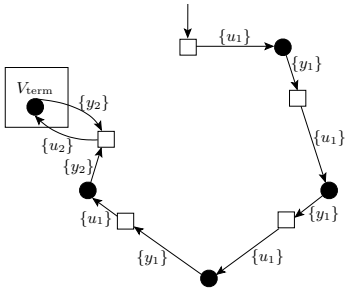

Fig. 6. An alternative, more direct plan that solves the problem of navigating Figure 4's robot to its charger. This plan is an homomorphic solution.

Example 26. *Recall Example 21, which shows a cyclic solution that involves tracing around the cyclic planning problem multiple times, until the least common multiple of their cycle lengths is found, in this case a series of 30 states in each graph. This plan is* not *an homomorphic solution, because each plan state corresponds to multiple problem states. However, a simpler plan, depicted in Figure 6, can be formed in which each plan state maps to only one problem state. This solution is therefore an homomorphic one.* ◇

The preceding example is a particular instance of a more general pattern.

Theorem 27. *If there exists a plan to solve a planning problem, then there exists an homomorphic solution.*

Proof: Suppose (P, P_{term}) is a solution to (W, V_{goal}). If every joint-execution arriving at v in P arrives at the same w in W, then $(v, w) \in R$ is a function, so (P, P_{term}) is an homomorphic solution. Thus, consider the cases for which there are elements $(v, w) \in R$ and $(v, y) \in R$, with $w \neq y$. Let $R_{\text{last}} \subset R$ be the relation where $(v_p, v_w) \in R_{\text{last}}$ iff there is a joint-execution $e_1 \cdots e_k$ arriving at v_p on P and v_w on W, and there are no joint-executions which extend the execution (e.g., $e_1 \cdots e_k \cdots e_m$, $m > k$) that arrive at v_w again. Then construct a new plan (Q, Q_{term}) with $V(Q) = V(W)$ and $V_0(Q) = V_0(W)$. For all edges departing $v \in P$ associated with $w \in Q$ where $(v, w) \in R_{\text{last}}$, we collect the label sets by unioning them to form V_e. Then edges departing w are included in Q by carrying over edges from W, intersecting V_e with all the labels of edges departing w, and dropping those for which the result is empty. Finally, an element w is included in Q_{term} if there is a $v \in P_{\text{term}}$ with $(v, w) \in R_{\text{last}}$. Then (Q, Q_{term}) is a solution to (W, V_{goal}) because, though (P, P_{term}) and (Q, Q_{term}) have different sets of executions, every execution on P that reaches P_{term} is transformed into another on Q reaching Q_{term} (and V_{goal}). Moreover, this ensures that R is a bijection, so that (Q, Q_{term}) is an homomorphic solution to (W, V_{goal}). □

4 Label maps and the damage they inflict

Since p-graphs are capable of representing several structures of interest, the next question is how they might enable a roboticist to evaluate tentative designs and

to better understand solution space trade-offs. One class of interesting design-time questions arises when one considers how modifications to a given robot's capabilities alter the planning and estimation efforts that the robot must undertake.

4.1 Label maps

We express modification of capabilities through maps that mutate the labels attached to the edges of a p-graph.

Definition 28 (action, observation, and label maps). *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'. Likewise, 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'. A* label map *combines an action map h_u and a sensor map h_y:*

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

Definition 29 (label maps on sets and p-graphs). *Given a label map h, its* extension to sets *is a function that applies the map to a set of labels:*

$$h(E) = \bigcup_{e \in E} h(e).$$

The extension to p-graphs *is a function that mutates p-graphs by replacing each edge label E with $h(E)$. We will write $h(P)$ for application of h to p-graph P.*

Example 30 (label maps on intervals). *If the action or observation space is $\mathbb{R}$, then we can implicitly represent some subsets of those events as a finite union of intervals [10]. To represent a label map on such an event space, we might, for example, take bounding polynomials $p_1(x)$ and $p_2(x)$, and define*

$$h(x) = \{x' \mid p_1(x) \leq x' \leq p_2(x)\}.$$

Given a finite-union-of-intervals label $\ell \subset \mathbb{R}$, we can evaluate this kind of h by decomposing h into monotone sections, selecting the minimal and maximal values of p_1 and p_2 within that range, and computing the union of the results across all of the monotone sections. Figure 7 shows an example. ◇

Label maps allow one to express weakening of capabilities as follows. If multiple elements in the domain of $h(\cdot)$ map to sets that are not disjoint, this expresses a conflation of two elements that formerly were distinct. When they are observations, this directly models a sensor undergoing a reduction in fidelity since the sensor loses the ability to distinguish elements. When they are actions, this models circumstances where uncertainty increases because a single

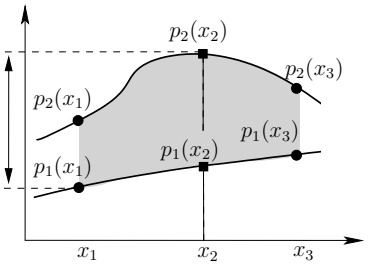

Fig. 7. A label map from $\mathbb{R}$ to $2^{\mathbb{R}}$ may be described by functions p_1 and p_2 as lower and upper bounds, respectively. The marked vertical interval, spanning $p_1(x_1)$ to $p_2(x_2)$ illustrates the image of h across the monotone segment from x_1 to x_2. Values for other monotone segments would be computed similarly.

action can now potentially produce multiple outcomes, and the precise outcome is unknown until after its execution.

Further, when the image of element E is a set with multiple constituents, this also expresses the fact that planning becomes more challenging. For observations, it means that several observations may result from the same state and, as observations are non-deterministic, this increases the onus for joint-executions to maintain safety (for example, plans must account for more choices).

For actions, while there is a seemingly larger choice of actions, this increase does not represent an increase in control authority because several actions behave identically. In both action and observation instances, the map may become detrimental when the outputs of $h(E)$ intersect for multiple Es and thus 'bleed' into each other. Broadly, one would expect that this is more likely when the output sets from $h(\cdot)$ are larger.

4.2 Destructive or not?

If a label map can express a change in a p-graph, the question is whether this change matters. One can pose this question meaningfully for planning problems as the added ingredients provide semantics that yield the notion of solubility.

Definition 31 (destructive and non-destructive). *A label map h is* destructive *on a set of solutions S to planning problem (G, V_{goal}) if, for every plan $(P, V_{\text{term}}) \in S$, $(h(P), V_{\text{term}})$ cannot solve $(h(G), V_{\text{goal}})$. We say that h is* non-destructive *on S if for every plan $(P, V_{\text{term}}) \in S$, $(h(P), V_{\text{term}})$ does solve $(h(G), V_{\text{goal}})$.*

Intuitively, destructiveness requires that the label map break all existing solutions; non-destructiveness requires that the label map break none of them.

Example 32 (single plans). *If $S = \{s\}$ is a singleton set, then we can determine whether h is destructive on S by applying the label map h —recall Definition 28— to compute $h(s)$ and $h(G)$, and then testing whether $h(s)$ solves $h(G)$ —recall the algorithm described in Section 3. If $h(s)$ solves $h(G)$, then h is nondestructive on S; otherwise, h is destructive on S. In this singleton case, we say simply that h is (non-)destructive on s.* ◇

Definition 31 depends on a selection of some class of solutions. Of particular interest is the maximal case, in which every solution is part of the class.

Definition 33 (strongly destructive and strongly non-destructive). *A label map h is* strongly (non-)destructive *on a planning problem (G, V_{goal}) if it is (non-)destructive on the set of all solutions to (G, V_{goal}).*

Note that, while strong destructiveness may be decided by attempting to generate a plan for $h(G)$ (perhaps by backchaining from V_{goal}), strong non-destructiveness may be quite difficult to verify in general, if only due to the sheer variety of extant solutions. (Recall Example 21, which solves its problem in an unexpected way.) The next results, while not sufficient in general to decide whether a map is strongly non-destructive, do perhaps shed some light on how that might be accomplished.

Lemma 34 (label maps preserve safety). *If P is safe on G, then for any label map h, $h(P)$ is safe on $h(G)$.*

Proof: Consider each pair of states (v, w), with $v \in V(P)$ and $w \in V(G)$ reached by some joint-execution on P and G. Suppose for simplicity that v is an action state. (The opposite case is similar.) Let E_1 denote the union of all labels for edges outgoing from v, and likewise E_2 for labels of edges outgoing from w. Since P is safe on G, we have $E_1 \subseteq E_2$. Then, in $h(P)$ and $h(G)$, observe that

$$h(E_1) = \bigcup_{e \in E_1} h(e) \subseteq \bigcup_{e \in E_2} h(e) = h(E_2),$$

and conclude that $h(P)$ is safe on $h(G)$. □

Lemma 35 (label maps never introduce homomorphism). *If (P, P_{term}) is a non-homomorphic solution to (G, V_{goal}) then no label map h results in $(h(P), P_{\text{term}})$ being an homomorphic solution to $(h(G), V_{\text{goal}})$.*

Proof: Since (P, P_{term}) is a non-homomorphic solution to (G, V_{goal}), there exist two joint-executions $e_1 \cdots e_k$ and $e'_1 \cdots e'_m$ on P and G such that both arrive at $v \in V(P)$ in P, but on G, the former arrives at $w \in V(G)$ and the latter arrives at $w' \in V(G)$ with $w \neq w'$. Now, given any $h(\cdot)$, pick any particular sequence $(h_1 \in h(e_1)) \cdots (h_k \in h(e_k))$, and $(h'_1 \in h(e'_1)) \cdots (h'_m \in h(e'_m))$, making choices arbitrarily. These are joint-executions on $h(P)$ and $h(G)$. Application of the label map means there is a way of tracing both $(h_1 \in h(e_1)) \cdots (h_k \in h(e_k))$ and $(h'_1 \in h(e'_1)) \cdots (h'_m \in h(e'_m))$ on $h(P)$ to arrive at v, while there is a way of tracing the former on $h(G)$ to arrive at w, and the latter at w'. So $(h(P), P_{\text{term}})$ cannot be an homomorphic solution to $(h(G), V_{\text{goal}})$. □

Theorem 36 (extensive destructiveness). *For a planning problem (G, V_{goal}), let $\mathcal{H}$ denote the set of homomorphic solutions that problem. Then any label map that is destructive on $\mathcal{H}$ is strongly destructive.*

Proof: Since h is destructive on $\mathcal{H}$, we know that $(h(G), V_{\text{goal}})$ can only have homomorphic solutions if some formerly non-homomorphic solution can become an homomorphic one under h, but Lemma 35 precludes that eventuality. This implies, via Theorem 27, that *no* plan solves $h(G)$. Therefore h is strongly destructive on (G, V_{goal}). □

The interesting thing here is that Theorem 36 shows that the class of homomorphic solutions play a special role in the space of all plans: By examining the behavior of h on $\mathcal{H}$, we can gain some insight into its behavior on the space of all plans. Informally, $\mathcal{H}$ seems to function as a 'kernel' of the space of all plans.

5 Related work and outlook

The examples presented early in the paper illustrate how p-graphs allow for a uniformity in treating multiple existing formal objects; the assortment of constructs is surely indicative of the ongoing search for foundational forms.

Combinatorial filters: This paper builds on our prior work [9, 10], which is strongly influenced by the combinatorial filtering perspective, with its use of simple, discrete objects that generalize beyond the methods used in traditional estimation theory, which has a strong reliance on probabilistic models. In both LaValle [7], which provides a tutorial introduction and overview to the approach, and the substantial paper on the topic [12], more work is needed to extend the theory, which only provides for inference, to express aspects of feedback-aware control for achieving tasks. The present paper makes some progress in extending the approach to deal with active (rather than merely passive) systems.

Strategy complexes: One formulation that emphasizes action from the outset is Erdmann's more recent work on strategy complexes [3]. He uses tools from classic and computational topology to relate plans, formulated broadly to include sources of non-determinism, to high-dimensional objects—his loopback complexes—whose homotopy type provides information about whether the planning problem can be solved. We speculate that preservation of plan existence under label maps might be productively studied across planning problems by examining the map's operation on loopback complexes: classes of maps that can be shown to preserve the homotopy type of such complexes (perhaps over restricted classes of planning problems) can be declared non-destructive.

Sensor maps on filters: The maps we have introduced here generalize that of our prior work [10] in three crucial ways: (1) These definitions consider modification of actuators and related resources involved in generating actions. (2) By mapping each element of some label set to another set and then taking the union on graph edges, one may express the notion of loss of information by having E grow under the action of h. This idea was expressed as a valuable feature for mutations in [10], but is not correctly achievable with the definitions therein. (3) The notion of non-destructiveness in that work is stronger than Definition 31, not only requiring that plan $(h(P), V_{\text{term}})$ solve $(h(G), V_{\text{goal}})$ but also that it solve it in the same way as (P, V_{term}) solves (G, V_{goal}).

Co-design: As we have already argued, the right formalism should aim to provide the representational basis for objects that can manipulated by algorithms in order to guide the design process. We mention here, perhaps related mostly by broader spirit, the recent work of Censi [2] wherein he poses and solves what he terms co-design problems where, given a network of monotone constraints, the selection of components is a process that can be automated. Part of the interest in studying labels maps is that they can model aspects of different components; forging connections with the co-design approach is interesting.

Hybrid automata: More immediately related, and also adopting an algorithmic stance on the design process, are methods based on hybrid automata (HA), several of which leverage powerful synthesis and verification techniques [1]. De-

spite some similarities, including extensive use of non-determinism, the relationship between p-graphs and HA is somewhat involved: guard expressions in a rich logical specification language have structure missing from the label sets we study; nor are actions labels intended to model continuous dynamics.

6 Conclusion

We believe that the most crucial intellectual contributions of the present work are in achieving a degree of abstraction of prior ideas in two ways: (1) We separate those entities which have been formalized in robotics because they have some interpretation that is useful (e.g., the idea of a plan, a filter), from their representation. The p-graph, in and of itself, lacks an obvious interpretation. Its definition does not include semantics belying a single anticipated use, rather context and any specific interpretation are only added for the special subclasses. (2) Even if the p-graph is a unifying representation, it is not a *canonical form*. This paper represents an important mental shift in lifting most of the notions of equivalence up to sets of executions (languages), rather than depending on operations on some specific graph. The present work continues to separate the notion of behavior from presentation.

References

1. Belta, C., Bicci, A., Egerstedt, M., Frazzoli, E., Klavins, E., Pappas, G.J.: Symbolic Control and Planning of Robotic Motion. IEEE Transactions on Robotics and Automation 14(1), 51–70 (Mar 2007)
2. 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)
3. Erdmann, M.: On the topology of discrete strategies. International Journal of Robotics Research 29(7), 855–896 (2010)
4. Erdmann, M.: On the topology of discrete planning with uncertainty. in advances in applied and computational topology. In: Zomorodian, A. (ed.) Proc. Symposia in Applied Mathematics. vol. 70. American Mathematical Society (2012)
5. Erdmann, M., Mason, M.T.: An Exploration of Sensorless Manipulation. IEEE Transactions on Robotics and Automation 4(4), 369–379 (Aug 1988)
6. Goldberg, K.Y.: Orienting Polygonal Parts without Sensors. Algorithmica 10(3), 201–225 (1993)
7. LaValle, S.M.: Sensing and Filtering: A Fresh Perspective Based on Preimages and Information Spaces. Foundations and Trends in Robotics 1(4), 253–372 (Apr 2012)
8. M. T. Mason and K. Y. Goldberg and R. H. Taylor: Planning Sequences of Squeeze-Grasps to Orient and Grasp Polygonal Objects. In: Seventh CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators (1988)
9. O'Kane, J.M., Shell, D.: Concise planning and filtering: Hardness and algorithms. IEEE Transactions on Automation Science and Engineering (2017), to appear.
10. Saberifar, F.Z., Ghasemlou, S., O'Kane, J.M., Shell, D.: Set-labelled filters and sensor transformations. In: Proc. Robotics: Science and Systems (2016)
11. Schoppers, M.J.: Universal Plans for Reactive Robots in Unpredictable Environments. In: Proc. International Joint Conference on AI. pp. 1039–1046 (1987)
12. Tovar, B., Cohen, F., Bobadilla, L., Czarnowski, J., LaValle, S.M.: Combinatorial filters: Sensor beams, obstacles, and possible paths. ACM Transactions on Sensor Networks 10(3) (2014)