Delineating boundaries of feasibility between robot designs

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

Abstract— **Motivated by the need for tools to aid in the design of effective robots, we examine how to determine the role that particular sensing and actuator resources play in enabling a robot to achieve useful ends. Rather than merely asking "will this sensor suffice?" we classify general modifications to the set of sensors and actuators based on the feasibility of accomplishing given tasks using these sets. The goal is to probe the boundary between modifications that are *destructive* on a given planning problem, and modifications that are not. Since this boundary itself can be impractically large, classic search methods are of no avail to summarize discriminatory features on this boundary. Instead, we propose a decision tree learning method to efficiently construct a compact implicit representation of the boundary. The idea is to allow the designer to use prior knowledge to constrain the search, then use the tool to probe the boundary subject to those constraints, gaining insight into the information necessary for a robot to ensure task achievement. Ultimately we envision a interactive process where additional constraints are repeatedly included as new light is shed. We aim to pave the way for interactive tools that help the roboticist navigate the complexities of the design space. We describe an implementation of this approach along with experimental results that show that the method can construct decision trees with explanatory value. Our experiments suggest that some domain knowledge (specifically picking features that emphasize monotonicity) substantially improves running-time with only negligible reduction in accuracy.**

I. Introduction

The last two decades of research have seen substantial progress on algorithms for robots—we now have mature reusable software, such as planners and packages for processing and fusing sensor readings, which run on-board of robots. Large strides have been made in applications and libraries for validating, verifying, and simulating robots and their controllers. Classification and regression methods have also become commodities that the nonspecialist can use. Software tools to help with design decisions, however, have lagged behind. When it comes to making informed choices about which sensors and actuators are required for a robot to carry out some mission, perhaps to a specific level of performance, the roboticist has little automated support.

In order to tame design problems, our philosophy is that one requires tools that are iterative and interactive. Tools without these characteristics will do too little to help navigate the morass of design possibilities. We are interested in using automation to help explore and elucidate structure within the problem domain, the task, and the robot's context. Naturally, we wish to judge tools by the insights their outputs provide the designer, but the underlying challenges include: (i) evaluating a design is a costly (computational) operation; (ii) it remains far from clear how and what to communicate to the human user in order to stimulate design insight.

The approach taken in this paper is to begin with a specification of task-achieving functionality, and then to explore questions phrased in terms of *destructiveness* of modifications made to sensors or actuators with respect to the requisite functionality. A modification operation —for example, the substitution of sensors or actuators for weaker ones— is said to be non-destructive if it still permits the robot to realize the desired functionality. In this paper, the specification of task constraints is implicit in the artefact provided as input—a discrete transition system called a procrustean graph. This contrasts with most formal synthesis approaches wherein some language, typically of a declarative form, is used to provide a specification.

Another distinguishing feature of our approach is that questions about destructiveness are posed over the set of conceivable sensors and actuators, rather than the particular ones we might have at hand. Doing so gets to the very kernel of what information must be obtained or preserved in the circumstances under consideration. But it has the disadvantage that the space is enormous. For this reason, we are interested in how a user can introduce structure in order to express some forms of domain knowledge.

The method proposed in this paper drills down into the informative boundary between successful and unsuccessful designs. This is done by collecting discriminatory features so that an assay of these features helps the human understand the fundamental elements of the problem domain. We adopt a decision tree learning approach to identify sparse but critical features to summarize crucial resources (or informational aspects of those resources) which play an important role in the robot's task. Decision trees are among the learning tools that provide the most straightforward interpretation. In this setting, the internal nodes of the decision tree are labeled with specific elements of information that may or may not be available in a particular design, and the leaves are labeled with bits indicating whether designs that reach those leaves are successful or not. The intuition is that, in a decision tree induced by an appropriate training set, internal nodes nearer to the root of the tree are likely to correspond to elements of information that are particularly important for identifying successful designs.

There are two main contributions in this paper: First, we present a method to induce a decision tree that gives us the information needed to decide what set of actions and

[1]Shervin Ghasemlou and Jason M. O'Kane are with Department of Computer Science & Engineering, University of South Carolina, Columbia, South Carolina, United States sherving|jokane@cse.sc.edu

[2]Dylan A. Shell is with the Department of Computer Science & Engineering, Texas A&M University, College Station, Texas, United States dshell@cs.tamu.edu

observations are crucial in a given planning problem. Second, we evaluate our method by presenting the results of our experiments, where the was method employed on different problem sizes. We show that domain knowledge can be effectively injected into the system via the feature extraction step. For our simple examples, this structure improves efficiency substantially —giving some evidence for reasonable scalability of the approach— with no loss to output accuracy.

II. Related Work

Though this work has a practical bent, it asks questions about the sufficiency, interchangeability, and the necessity of sensors, actuators, or both (which we call resources, more broadly) for some given task. These questions relate to some of the most fundamental concerns of any theory for robots, and are considerations that have a long history of development within robotics (e.g., [1], [2], [3], [4], [5], [6], [7], [8], [9]).

Beyond those classical works, the recent research most closely related to this paper examines relationships between sensors in detail, in a variety of ways, and in order to propose abstractions to aid in thinking about design problems. In [10], the authors compare pairs of robotic systems expressing power by using idea of one system simulating (in an informational sense) another. That paper, and a subsequent one [11], makes extensive use of the concept of *dominance*, even building a lattice where the partial order of the lattice is the dominance relation that is close to the notion of a refinement relation we use in this paper. A different approach, introduced by [12], compares *families* of parameterizable sensors by seeking maximal performance from sensors for a given task, and then performing an analysis of the relationship between power and performance. Using Mean-Square-Error as a metric of performance, the authors are able to examine the effects of structured noise, showing that different prior information affects the relation between families of sensors—culminating in a comparison of standard and event-based cameras. Also related, though without the design-centric perspective we adopt, is [13], wherein the authors show how to identify abstractions which are necessary and sufficient for a given planning problem. Within the context of automated synthesis, Raman and Kress-Gazit [14] introduce an algorithm that produces an explanation for why some specification cannot be fulfilled. Finally, Censi [15] examines the dual problem, showing results that suggest one could construct a catalog of components (sensors and actuators), and then search over compositions from this catalog—somewhat surprisingly, this may even be tractable for certain classes of components.

In this paper, the task of a robot is expressed as a planning problem. Sensors and actuators then provide the means by which the goal is to be achieved. One representation that is sufficiently rich to express planning problems, plans, degradation and modifications to resources, is the notion of the p-graph described in [16], [17]. In the next section a compressed definition of these structures is presented to ensure that the present paper is (mostly) self-contained. The definition will be seen to be a generalization to some earlier models, including LaValle's [18] combinatorial filters for modeling discrete state estimation problems. That work differs from ours in the way actions and observations are modelled: his states represent actions exclusively, not observations, and in each state only one action may be executed.

In Schoppers [19], universal plans describe actions to be taken by the robot for any of the circumstances that may arise and, each possible observation is followed by an appropriate action. A universal plan is a p-graph with only one observation state. In non-deterministic graphs [20], non-determinism in actions is modelled by edges that encapsulate a set of possible outcomes, instead of one for each action edge, as in p-graphs. Some early studies (see [8], [21], [7]) examined how to achieve goals without sensors. There, the plans produced consist exclusively of a sequence of consecutive actions, and observations play no role—these again have a p-graph representation.

The preceding discussion emphasizes generality and, as our algorithms operate on p-graph inputs, the variegated cases above are all possible with the approach. But the primary utility of p-graphs to us here is as a conveniently manipulable *data-structure* to enable automated processing of plans, planning problems, and filters—as is essentially demonstrated in [22].

III. Preliminaries and basic definitions

In order to introduce the problem that we tackle in this paper, in this section we first present our model for robot-environment interactions (*p-graphs*) and the model for how these interactions can be altered (*label maps*). The definitions provided here are condensed versions of more detailed, comprehensive, formal definitions that appear in [17].

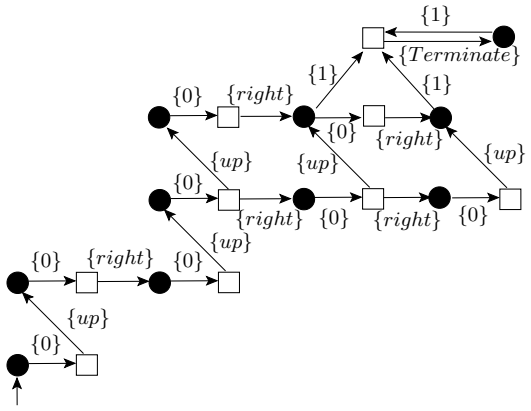
A. Procrustean graphs and planning problems

We model the world using p-graphs, which represent the robot's interactions with its environment as a sequence of actions and observations, the former executed by the robot, the latter being responses the robot receives through its sensors from the environment after performing each action.

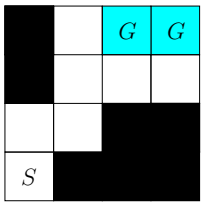
Definition 1 (p-graph). *A* procrustean graph (p-graph) *is an edge-labelled bipartite directed graph in which*

1) *the finite vertex set V, of which each member is called a* state*, can be partitioned into two disjoint subsets, 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 (observation) vertex is labeled with a set of actions $U(e)$ (observations $Y(e)$) and leads to an observation (action) vertex,*
3) *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 ($V_0 \subseteq V_{\mathrm{y}}$).*

Labels are sets of either exclusively actions, or, exclusively observations. We write U, called the *action space*, to represent the union of all labels attached to the outgoing edges from all action vertices. Similarly, we write Y, called



(a) The problem of moving from S to G in the grid of Figure 1b is encoded as a p-graph. Circle nodes are observation nodes and square nodes are action nodes. Observation *0* means the robot is not in the goal region and observation *1* indicates that the robot is in the goal region.



(b) The grid environment in which the robot moves.

Fig. 1: A grid environment with initial node S, goal region cells G and a set of obstacle cells. The robot can move in two directions, *up* and *right*, and has a goal sensor.

the *observation space*, to represent the union of all labels attached to the outgoing edges from all observation vertices.

P-graphs describe sequences of actions and observations. A particular sequence of actions and observations that, beginning at initial state, can be traced on a p-graph is called an *execution*. Here tracing involves traversing an edge with either an action or an observation that is an element within the set labelling the edge.

A *planning problem* is a p-graph G equipped with a subset of states, V_{goal}, distinguished so as to be part of the *goal region*. A *plan* is a p-graph P along with a set of states, V_{term}, we call its *termination region*. Thus, p-graphs are effective at modelling both and, surprisingly, with identical formal structures.

A plan (P, V_{term}) is said to *solve* a planning problem (G, V_{goal}) if there exists an integer k that bounds the length of all sequences that are executions on both graphs. Additionally, we require that the plan handle every observation it can receive from the world, and the world every action from the plan. Finally, every execution must either arrive at a state V_{goal} when within V_{term}, or be the prefix of some such sequence. (See [17] for the precise formalization of the required notions of safety, liveness, and correctness.)

Figures 1a and 1b show a simple example of a planning problem, and Figure 2 a plan which solves that problem.

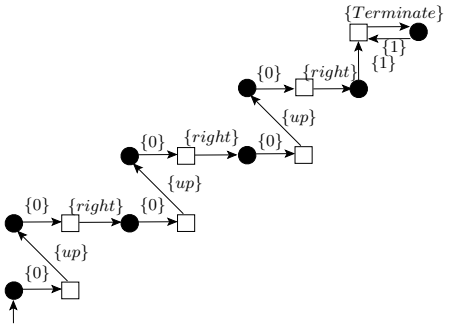


Fig. 2: A plan that solves the planning problem of Figure 1b.

B. Label maps and destructiveness

We formalize modifications to a robot's sensors or actuators as changes to the p-graph. We consider a class of such changes in the form of a transformation from one p-graph to another, expressing a change in capabilities by transforming sets of actions and observations to new ones.

Definition 2 (**action, observation, and label maps**). *An* action map *is a function $U \to U'$ mapping from an action space U to another action space U'. Likewise, an* observation map *is a function $Y \to Y'$ mapping from an observation space Y to another observation space Y'. A* label map *combines a sensor map h_y and an action map h_u:*

$$h(a) = \begin{cases} h_y(a) & \text{if } a \in Y \\ h_u(a) & \text{if } a \in U \end{cases}. \tag{1}$$

Definition 3 (**applying a map on a p-graph**). *Given a label map h and a p-graph G, we say h is applied on G, if for each label l in the p-graph, we replace each $a \in l$ with $h(a)$. The resulting p-graph is denoted as $h(G)$.*

We are interested in determining whether applying map h is destructive on a planning problem. In other words, does applying the map preclude the existence of a plan to solve the planning problem?

Definition 4 (**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 $(h(G), V_{\text{goal}})$. If S is the set of all plans that solve (G, V_{goal}), then we say h is* strongly destructive.

We are interested in identifying the strongly destructive maps for a given planning problem. The next proposition shows that we can do so by applying the map on the planning problem itself, and checking whether the new planning problem has any solutions.

Proposition 1 (**checking strong destructiveness**). *Given a planning problem (G, V_{goal}) for which at least one solution exists, along with a label map h, if no plan exists that solves $(h(G), V_{\text{goal}})$ then h is strongly destructive on (G, V_{goal}).*

Proof. If no plan exists that solves $(h(G), V_{\text{goal}})$, then it can't be a $(h(P), V_{\text{term}})$ for any (P, V_{term}). □

To decide if there exists a solution for a planning problem (G, V_{goal}), one can run a backchaining algorithm on G. The

Algorithm 1: SOLVABLE((G, V_{goal}))

```
1  G ← G.ToStateDetermined()
2  Solution ← ∅
3  for v in V_goal do
4  |  Solution.add(v)
5  Changed ← False
6  while NOT Changed do
7  |  Changed ← False
8  |  for v in V(G) do
9  |  |  if v is an 'action' node then
10 |  |  |  if exists a v' ∈ V(G) that v is connected to
   |  |  |    and v' ∈ Solution then
11 |  |  |  |  Solution.add(v)
12 |  |  |  |  Changed ← True
13 |  |  if v is an 'observation' node then
14 |  |  |  if All nodes v'_1...v'_k that v is connected to
   |  |  |    are in Solution then
15 |  |  |  |  Solution.add(v)
16 |  |  |  |  Changed ← True
17 if all v ∈ V_0 are in Solution then return True
18 return False
```

algorithm constructs a solution incrementally by identifying nodes in the planning problem from which we can guarantee to reach the goal, starting with V_{goal}.

The pseudocode for the backchaining algorithm is presented in Algorithm 1. In the first line, the p-graph of the planning problem will be converted to a *state-determined* equivalent p-graph, where each node's outgoing edges have disjoint labels. For any given p-graph an equivalent state-determined p-graph exists, and an algorithm to generate it is known [17], [16]. In Algorithm 1, each action node will be added to that solution if any of its outgoing edges is connected to a node that has previously been added to the solution (Line 9). For observation nodes, each observation node will be added to a solution if all of its outgoing edges are connected to a node that is already in the solution (Line 13). This different treatment is based on the fact that in observation nodes, what observation the sensor gets is unknown, therefore, any plan that solves the planning problem should be able to solve it for all possible observations it gets in any of its observation nodes.

The algorithm continues until the *while* loop updates no vertices, then no other vertex remains to be added to the solution. The final *if* statement checks whether all initial vertices have a plan, and if so returns *True*. Otherwise *False* is returned.

Now we have the two required pieces that we need in the next section: label maps to model possible changes to the set of actions and observations, and, an algorithm to decide whether those changes are strongly destructive.

IV. NON-DESTRUCTIVENESS BOUNDARY

Our goal, given a planning problem, is to generate a description of the boundary between those maps that are strongly destructive on the given planning problem, and those maps which are not. To formally define the problem the following definitions are required.

Definition 5 (**equivalence**). *We say r and s, which either both are actions or both are observations, are* equivalent *under map h, if $h^{-1}(h(r)) = h^{-1}(h(s))$.*

We have used $h^{-1}(S)$ to denote the preimage set of S. Therefore the equality above is on sets.

This equivalence relation is important because, for p-graphs with action space U and observation space Y, we can fully characterize ability of a map to be destructive, based on the equivalence relation it induces on $U \cup Y$. That is, by considering all partitions of $U \cup Y$, we can effectively consider all label maps on that p-graph. Each element of each such partition is an equivalence class of some $r \in U \cup Y$ under the maps that induce that partition. The intuition is that members of equivalence classes are indistinguishable, since they are mapped to a common element. Using the definition of equivalence, we define the refinement relation in the usual way.

Definition 6 (**refinement relation**). *Given two maps, $h : U_1 \cup Y_1 \to U_1' \cup Y_1'$ and $g : U_2 \cup Y_2 \to U_2' \cup Y_2'$, we say h is a* refinement *of g, denoted $h \prec g$, if for each label $y_1' \in U_1' \cup Y_1'$, there exists a $y_2' \in U_2' \cup Y_2'$, such that $h^{-1}(y_1') \subset g^{-1}(y_2')$. We say maps h and g are* comparable *if either $h \prec g$ or $g \prec h$.*

If $h \prec g$, then map h possesses greater distinctiveness between images of the elements in its domain. In other words, by having more "injective-ness," h preserves more information than g when applied to a common set of elements. Therefore g is more likely to endanger the robot's ability to reach its goals. The refinement relation defines a partial order over the set of all maps that could be applied on a p-graph.

Based on these two definitions, now we can define the boundary of non-destructiveness.

Definition 7 (**non-destructiveness boundary**). *Given a planning problem (G, V_{goal}), we say that a map m is on the* non-destructiveness boundary *if m is not strongly destructive and for any other map m' comparable to m:*

1) *if $m \prec m'$ then m' is strongly destructive.*
2) *if $m' \prec m$ then m' is not strongly destructive.*

The objective in this paper, given a planning problem, is to find those crucial actions or observations or combinations thereof that distinguish the boundary maps from all other maps. The idea is to find the non-destructiveness boundary of a planning problem, without performing the destructiveness test on all possible maps. This is because the space of all possible maps is extremely large even for comparatively small planning problems, and therefore using classic search methods will be of no avail. In fact, it is known that the problem of finding such boundary is NP-hard [16].

V. Learning the Non-destructiveness Boundary

Next, we describe a method for learning a compact, legible description of the non-destructiveness boundary for a given planning problem. Our approach uses decision tree induction along with applications of Theorem 1, which enables us to decide whether a map is strongly destructive on a planning problem. We generate training data automatically via random samples from the space of all label maps for the given planning problem.

A. Feature Extraction

Label maps themselves, being functions $U \cup Y \to U' \cup Y'$, cannot be used as features directly. Precisely how to do this involves some freedom of choice and, as will be shown by our experimental results, affects the performance (output quality and running-time) of the whole process. Our ultimate vision is to probe the destructiveness boundary interactively: feature extraction is one place where domain knowledge can inform and structure the learning process. Consequently, we describe three feature extraction methods:

1) The full pairwise feature set: We extract one feature for each pair of distinct actions and each pair of distinct observations. Specifically, for a given map h over n actions and m observations, we encode h using the $m(m-1)+n(n-1)$ feature vector $[x_{u_i u_j}, x_{y_k y_\ell}]$ where $i, j \in \{1, \ldots, n\}, i \neq j$, and $k, \ell \in \{1, \ldots, m\}, k \neq \ell$, and

$$x_{ab} = \begin{cases} 0 & \text{if } h(a) = h(b) \\ 1 & \text{if } h(a) \neq h(b) \end{cases}.$$

Each 1 in the feature vector indicates that the corresponding pair of labels is distinguishable, and each 0 indicates they are indistinguishable.

For example, given maps h_1 and h_2, with $h_1(a) = a', h_1(b) = a', h_1(c) = c', h_1(d) = d'$ and $h_2(a) = c', h_2(b) = b', h_2(c) = c', h_2(d) = b'$. The feature vectors for h_1 and h_2 will be [0 1 1 1 1 1] and [1 0 1 1 0 1], respectively.

2) Monotone feature set: In considering all pairwise features, one is disregarding any notion of action or observation locality. To do better, suppose that one has a meaningful ordering on the set of labels. Then, for some particular problems, we might expect that compact sets of labels could be usefully treated equivalently and the (comparatively sparser) boundaries between those sets might be where important distinctions ought to be made. In such cases, something like step-function between the sets can encode this information. The label map itself need not be monotone, but we can bias selection of features by considering pairs, as in the previous case, but now paying attention only to neighboring actions and observations. The result is that one has the substantially smaller $(n-1)+(m-1)$ vector: $[x_{u_i u_{i+1}}, x_{y_k y_{k+1}}]$ where $i \in \{1, \ldots, n-1\}$, and $k \in \{1, \ldots, m-1\}$, with the analogous definition of the basic elements.

3) Randomized feature set: Whatever the observed differences in performance of two feature sets, any comparison that ignores vastly different sizes inevitably fails to address the question of whether sparsity is playing an important role. To resolve this dilemma, we introduce a baseline feature set: $[x_{u_i u_j}, x_{y_k y_\ell}]$ where the indices were selected uniformly at random to produce a subset of the all pairwise features but with only $(n+1)+(m-1)$ elements, the first term being actions, the second observations.

B. Sampling

To form a set of training samples, we generate maps uniformly at random from the space of all partitions of $U \cup Y$. Pseudocode for an algorithm to accomplish this appears as Algorithm 2. Given the domain of the maps as input, the algorithm's first *for* loop initializes a set equivalence classes with empty sets. The second loop randomly assigns each element of the domain to one of the classes. The third nested loop assembles the final map, in which all members of each equivalence class are mapped to a common element (without loss of generality, the first member of that class).

C. Feature Selection

The full pairwise feature vector in the previous section grows quickly and becomes impracticable for sufficiently large problems. To avoid the effects of enormous vectors on the tree induction process, we select features by choosing only the most informative features, removing the features with low variance across the training set. As the features are boolean, variance is computed via an expression for Bernoulli random variables, where p is the feature's probability of being 1 among all the samples, so that $Var[x] = p(1-p)$. We choose a non-negative real number parameter τ, and eliminate any feature x for which $Var[x] < \tau$.

D. Decision Tree Induction

We use the Classification and Regression Trees algorithm (CART) [23] to induce a decision tree for the non-destructiveness boundary. The idea is for a given planning problem, we sample the space of all maps for that problem, generate features for each map, apply destructiveness test on all of these maps, and assign classes to each instances based on the outcome of the test. These automatically labeled instances become the input to CART. The output of CART is a decision tree, in which each node is tagged with a pair of actions or a pair of observations. The more crucial pairs are expected to show up in the higher levels of this tree.

VI. Experimental Results

We performed a set of experiments to measure the effectiveness of the proposed methods. The experiments used an Intel Core i7 machine with 4 gigabytes of RAM. We used the implementation of CART provided by the scikit-learn package [24]. In all of these experiments, the depth of the induced tree is limited to 7 levels.

Our first experiment used the planning problem of Figure 3 left to measure the classification accuracy. The p-graph of this planing problem is illustrated in Figure 4. For this experiment, we used a variety of sizes of training sets. Two examples of decision trees learned for this problem, each from a different training set size, are shown in Figure 5 and

Algorithm 2: GENERATESAMPLEMAP(U, Y)

$h \leftarrow$ identity map over $U \cup Y$
$E \leftarrow$ array of size $|U|+|Y|$
for $i = 0$ *to* $|U|+|Y|$ **do**
  $E[i] \leftarrow$ empty set
for $x \in U \cup Y$ **do**
  $r \leftarrow$ random integer between 0 and $|U|+|Y|-1$
  $E[r].add(x)$
for *each* $c \in E$ **do**
  for *each non-empty* $e \in c$ **do**
    $h(e) = c[0]$
return h

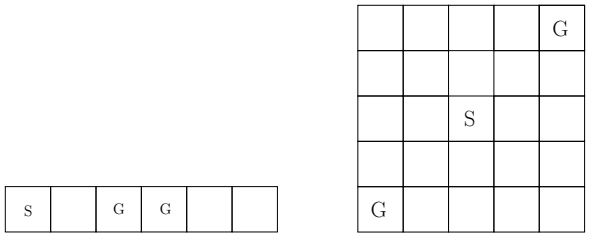


Fig. 3: Two simple scenarios where a robot is equipped with a sensor that measures the robot's position with the grid. Movement uncertainty means that actions may move the robot one or two cells in the desired direction. Movement that might otherwise take the robot outside the grid boundary, instead stays in the same place. The starting positions are shown by S and goal regions by G. [left] A 6×1 grid environment, wherein the robot can move left and right. Owing to the action uncertainty, determining whether the robot is in the goal region or in cells immediately adjacent is a crucial piece of information for this robot. [right] A 5×5 grid environment, wherein the robot can move left, right, downwards and upwards. For this problem, a plan exists that does not rely on any sensor.

Figure 6. These decision trees confirm the natural intuition of which actions or observations are more crucial. For example, in Figure 6, the root of the tree determines whether the given map enables the robot to distinguish between actions right and left or not. If not, any such map is destructive. This means that if the robot confuses going right with left, it will never be sure whether it has arrived in the goal region. Similarly for some observations, the tree checks whether the robot can distinguish being in different cells. For example, if positions 2 and 3 cannot be distinguished, where the boundary between the goal region and the rest of the environment lies, if positions 4 and 5 are also conflated, then such a map would be destructive. This is true because if only 2 and 3 are conflated, a correct plan still exists in which the robot can move to the right four steps to get rid of its uncertainty before proceeding to the goal. The tree in Figure 6, while trained with much less sample maps, gives us almost the same information. This shows that our method, even given a short time, can produce highly informative trees.

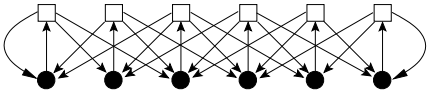


Fig. 4: The p-graph for the planning problem in Figure 3 left. Edge labels are omitted.

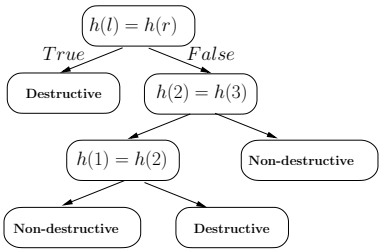


Fig. 5: An example decision tree made for the problem shown in Figure 3 left, when the size of the training set is 20.

In this experiment, we varied the training set size in increments of 2, and performed 100 trials for each training set size. For each trial, we measured the accuracy of the induced tree using a test set of 50 independently-generated random instances. We performed separate trials for the pairwise feature set, the monotone feature set, and a randomized feature set. Figure 7 shows the results. We observe that monotone and pairwise feature sets perform roughly equivalently, significantly outperforming the random feature set.

A similar experiment was done for the planning problem of Figure 3 right. An example of a decision tree learned for this problem is illustrated in Figure 8. As it can be seen, in the tree of Figure 8, if actions left and right or upwards and downwards are conflated, this results in destructiveness. In this problem, the robot can reach the goal without relying on any sensor (for example, by moving two steps upwards and two steps to the right). In Figure 9 the accuracy results versus different sizes of the training set for this problem, using the same experimental setup as for Figure 7, are shown. In this case, no clear trend to distinguish the three feature sets is evident.

The results so far, though only for a small selection of problem instances, are suggestive that the monotone feature set in particular performs approximately as well as the full pairwise feature set. In another experiment, we constructed a family of planning problems analogous to the 5×5 example in Figure 3 right, but with grid sizes ranging up to 13×13. We used both the full pairwise feature set and the monotone feature set to induce decision trees for these problems using a training set of size 50. The results, which are illustrated in Figure 10, show a clear advantage in scalability for the monotone feature set as the size of the input p-graph increases. This observation, combined with our earlier observation that reducing to the monotone feature set does not seem to have a significant impact on the accuracy of the induced trees, indicates that the monotone features are indeed successful in enabling our approach to handle larger scale problems.

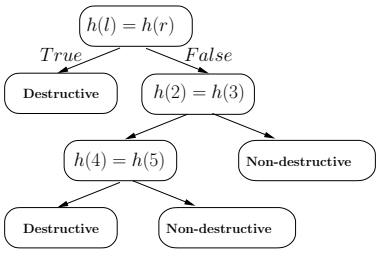


Fig. 6: An example decision tree made for the problem shown in Figure 3 left, when the size of the training set is 100.

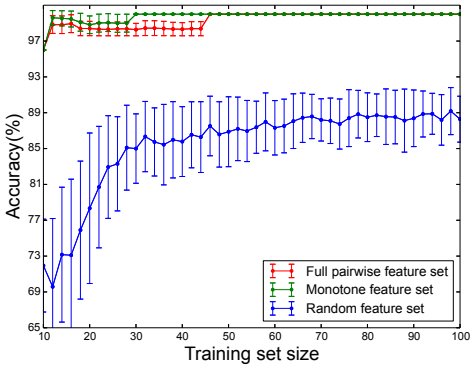


Fig. 7: Accuracy versus size of the training set for the planning problem of the grid environment in Figure 3 left, using different feature selection methods. The bars represent one standard deviation of uncertainty.

VII. Conclusion

The process of designing a robot that reliably can accomplish a task in an environment consists of many steps. In this paper, the questions we were trying to answer were: What degradation to sensors or actuators can be tolerated by the robot? What sets of resources (sensors and actuators) are minimal for accomplishing a specific job? How do you to find such a set? To this end, we used label maps to model modification in the set of sensors and/or actuators, and the associated notions of destructiveness and of a boundary were provided. To deal with the enormous number of possible label maps, we sampled over the space of all maps, and used a decision tree classifier to determine destructiveness of any given planning problem. The algorithm, upon completion, returns a tree where the nodes near to the root contain information about pairs of observations or actions that are most important in that when those elements are indistinguishable, the goal becomes unachievable. We reported the results from experiments were conducted on a number of planning problems. The data are promising, demonstrating that the method, is scalable beyond trivial problems, and is able to produce trees that are capable of classifying maps on a given problem with a high accuracy, even with few samples.

Potential future work includes developing a more intelligent way to produce sample maps to yield fast tree convergence. Using the definition of the refinement relation

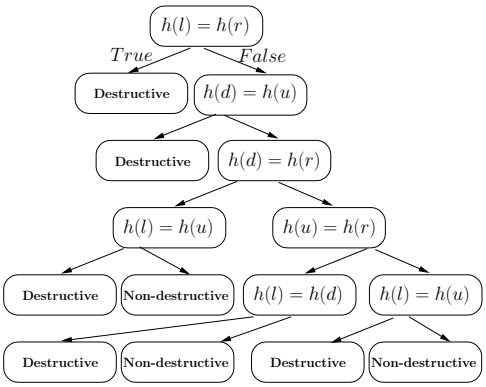


Fig. 8: An example decision tree made for the problem shown in Figure 3 right, from a training set of size 100. There are no nodes in the tree that check the possible confusion between a pair of observations. In other words, in this problem the robot requires no sensors to reach its goal.

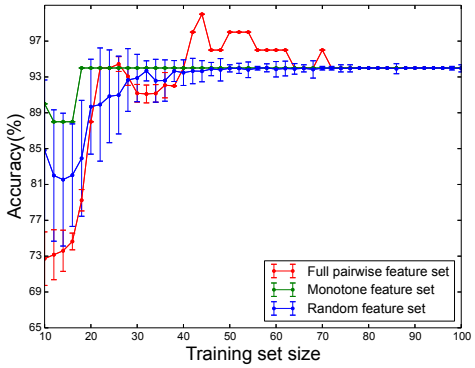


Fig. 9: Accuracy versus size of the training set for the planning problem of the grid environment in Figure 3 right, using different feature selection methods.

and the partial order over the space of all maps, starting from the identity map, one may sample maps that exactly lie on the boundary and, thus, will be more informative during the induction process. In addition, better feature selection methods could be used to improve accuracy and improve induction times. Another branch of future work could be to generalize the model we use to represent labels. Set-labelled p-graphs can be enhanced by more general, more expressive labels (for example, a tree-label where leaves are simple labels and internal nodes are logical operations) to close the gap between the abstract idea of plans expressed as p-graphs and their implementation on today's robots. Devising a method to get the set of all maps that lie on the boundary of non-destructiveness, and generalizing all these methods to continues maps is also future work.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. IIS-1527436, No. IIS-1453652 and No. IIS-1526862

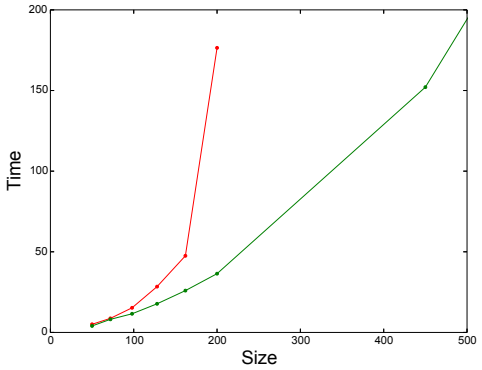


Fig. 10: The induction time (in seconds) versus the number of states, using all features (red) and monotone features (green). We performed this experiment on $n \times n$ grid environments where one of the center cells was the initial position and the bottom left and upper right cells were the goal region. For these $n \times n$ planning problems, there are $2n^2$ states in the corresponding p-graph.

REFERENCES

[1] Bruce Randall Donald and James Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 190–197, 1991.

[2] Bruce Randall Donald. On information invariants in robotics. *Artificial Intelligence*, 72(1-2):217–304, 1995.

[3] Michael A Erdmann. Understanding action and sensing by designing action-based sensors. *International Journal of Robotics Research*, 14(5):483–509, 1995.

[4] Ercan U Acar and Howie Choset. Robust sensor-based coverage of unstructured environments. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 61–68. IEEE, 2001.

[5] Pankaj K Agarwal, Anne D Collins, and John L Harer. Minimal trap design. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 3, pages 2243–2248. IEEE, 2001.

[6] Benjamín Tovar, Luis Guilamo, and Steven M LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Algorithmic Foundations of Robotics VI*, pages 425–440. Springer, 2004.

[7] Kenneth Y Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2-4):201–225, 1993.

[8] Michael A Erdmann and Matthew T Mason. An Exploration of Sensorless Manipulation. *IEEE Transactions on Robotics and Automation*, 4(4):369–379, August 1988.

[9] Jeff Wiegley, Ken Goldberg, Mike Peshkin, and Mike Brokowski. A complete algorithm for designing passive fences to orient parts. *Assembly Automation*, 17(2):129–136, 1997.

[10] Jason M. O'Kane and Steven M. LaValle. On comparing the power of robots. *International Journal of Robotics Research*, 27(1):5–23, January 2008.

[11] Steven M LaValle. Sensor lattices: A preimage-based approach to comparing sensors. Technical report, University of Illinois Urbana-Champaign, Department of Computer Science, September 2011.

[12] Andrea Censi, Erich Mueller, Emilio Frazzoli, and Stefano Soatto. A power-performance approach to comparing sensor families, with application to comparing neuromorphic to traditional vision sensors. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3319–3326. IEEE, 2015.

[13] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Constructing symbolic representations for high-level planning. In *AAAI*, pages 1932–1938, 2014.

[14] Vasumathi Raman and Hadas Kress-Gazit. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics*, 29(1):94–104, 2013.

[15] Andrea Censi. A Class of Co-Design Problems with Cyclic Constraints and Their Solution. *Robotics and Automation Letters*, February 2016.

[16] Fatemeh Zahra Saberifar, Shervin Ghasemlou, Jason M O'Kane, and Dylan A Shell. Set-labelled filters and sensor transformations. In *Robotics: Science and Systems*, 2016.

[17] Shervin Ghasemlou, Fatemeh Zahra Saberifar, Jason M O'Kane, and Dylan A Shell. Beyond the planning potpourri: reasoning about label transformations on procrustean graphs. In *Proc. International Workshop on the Algorithmic Foundations of Robotics*, 2016.

[18] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[19] Marcel Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI*, volume 87, pages 1039–1046. Citeseer, 1987.

[20] Michael A Erdmann. On the topology of discrete planning with uncertainty. in advances in applied and computational topology. In A. Zomorodian, editor, *Proc. Symposia in Applied Mathematics*, volume 70. American Mathematical Society, 2012.

[21] Matthew T Mason, Kenneth Y Goldberg, and Russell H Taylor. Planning sequences of squeeze-grasps to orient and grasp polygonal objects. *Planning*, 1:1–1988, 1988.

[22] Jason M O'Kane and Dylan A Shell. Concise planning and filtering: hardness and algorithms. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, pages 1–16, May 2017.

[23] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.