Accelerating the Construction of Boundaries of Feasibility in Three Classes of Robot Design Problems

Shervin Ghasemlou and Jason M. O'Kane

Abstract—This paper aims to improve the practical scalability of automated tools to assist in designing robots. Such problems rapidly become intractable because the underlying design space is immense. We consider a specific type of design tool addressed in prior work, which constructs a representation of the destructiveness boundary in the space of robot designs. This prior work showed that a legible representation, specifically a decision tree, of this boundary can illuminate which elements of a sensing or actuation system are most important for enabling the robot to complete its task. In that context, the robot's interaction with the world is represented as procrustean graph, and the space of robot designs is represented by the space of label maps that rewrite the labels on that graph. In this paper, we expand upon those results by showing how domain knowledge can enable such tools to find solutions to more complex problems within a reasonable time frame.

Specifically, we propose three different scenarios, expressed as constraints on the p-graph and on the label maps, under which the learning algorithm to identify the destructiveness boundary can converge quickly to high accuracy results for problems at larger scales than the prior, general-purpose algorithm. The conditions for each of these scenarios are easily verifiable and the set of problems that fall under each is rich enough to encompass several interesting problems. Experimental results demonstrate the effectiveness of the proposed methods.

I. INTRODUCTION

Roboticists nowadays are able to build robots that have high degrees of autonomy. However, in the majority of robotic applications, the automation that the robotic system provides is heavily dependent on a design that itself is not done in an automated fashion. Thus, we are interested in algorithms that can assist human roboticists with the process of designing autonomous robots.

Algorithmic robot design and co-design, being in its infancy, requires the roboticist to deal with several important issues. How the design problem is modelled, and how the relationship between different designs is defined are among these issues. The former needs to capture all the problem's elements accurately, and the latter has to offer a good means for design algorithms to navigate in the space of all designs effectively. In the space of all possible designs, we are primarily interested in those designs that enable the robot to complete its task. We model this situation by starting with an idealized robot model, and reasoning about various kinds of reductions in that robot's abilities. In this context, it is valuable to know about those designs that do not destroy



Fig. 1: An example in which a UGV localizes itself using the signals it gets from three different transmitters, in order to move from its initial position (highlighted in a circle) to the goal region (highlighted in a rectangle). Two of the transmitters produce signals that are powerful enough to reliably cover the entire region. The third transmitter is reliable only within some radius. To determine which observations are most important for this problem (that is, those that most strongly determine whether the transmitter is adequate for the task), we only need check the observations in which the signals received from the third transmitter are outside the reliable range. Applying this domain knowledge can reduce the number of costly destructiveness tests needed to identify the destructiveness boundary.

(i.e. are not destructive of) the robot's ability to complete the assigned task.

To that end, we are interested in understanding the *boundary* in the space of robot designs between non-destructive realizations of a model (i.e. those for which the robot is able to complete its task) and destructive ones (for which the robot cannot complete the task). However, because the space of robot designs is enormous, explicitly computing such a boundary can easily become intractable for problems of any appreciable size. In fact, even the problem of finding a single 'simplest' design on the boundary is known to be NP-hard [17].

In our prior work [10], we proposed a sampling-based method to tackle this problem. We utilized the concept of

The authors are with Department of Computer Science & Engineering, University of South Carolina, Columbia, South Carolina, United States {sherving, jokane}@cse.sc.edu

label map [17] to capture the idea of modification in the set of sensors and actuators a robot is equipped with. Instead of trying to directly compute the destructiveness boundary, that existing method seeks to identify observations and actions that determine destructiveness. See Figure 1. This is done by sampling the space of all maps, testing the destructiveness of the sampled maps, and then inducing a decision tree model using those results.

In the current work, based on this idea, we make the scale of some of the interesting planning problems a codesign algorithm can solve substantially larger. To do this, we propose three different classes of problems, and their corresponding approaches. Depending on the structure of the problem and the type of possible changes that can take place on the suite of actuators and sensors of a robot, the subset of the space the algorithm needs to sample from could get substantially smaller. This is the first reason why domain specific information makes the learning process faster. The second reason comes from the fact that the samples extracted from this subspace are richer in terms of information they provide, which results in a faster convergence of the decision tree.

There are two main contributions in this work. First, after providing an overview of the related work and preliminaries in Section II and Section III, in Section IV we propose three different scenarios where the knowledge of the domain could be successfully incorporated into the learning process. We show how this knowledge should be used in extracting the features in the sampling phase. Second, we show the effectiveness of our approach by presenting the results of our experiments for all of these scenarios in Section V.

II. RELATED WORK

The questions about sufficiency of the simple collections of sensors and actuators to complete various tasks, and of the limits of a robotic system's power have been addressed in numerous studies [1], [2], [6]–[10], [12], [15], [20], [21], using a variety of models. The model we use here, namely procrustean graphs (p-graphs), describes the interactions between the robot and the environment it operates in as a sequence of actions executed by the robot and observations received by the robot from the environment. The definition of p-graph encompasses those of many other models, including combinatorial filters [13], the minimalist manipulation plans of Erdmann, Mason, Goldberg, and Taylor [8], [12], [14] and universal plans [19].

In a study by O'Kane and LaValle [15], the relative powers of various robotic systems are compared. The authors present and utilize the concept of *dominance* in navigating between the space of designs. This concept is closely related to the concept of refinement relation, presented in our previous work [10]. In another set of closely related studies [4], [5], Censi suggests that one could construct a catalog of components (sensors and actuators), and then search over compositions from this catalog—somewhat surprisingly, this is tractable for certain classes of components. In the current work, we use procrustean graphs [11], [18] to encode planning problems and plans that solve them. The notion of plan here refers to a p-graph that guides the robot within the planning problem to achieve its goal. We use the concept of label maps [17] to model possible modifications to a robot's set of sensors and actuators, and compare these modification by means of a refinement relation [10].

III. PRELIMINARIES AND BASIC DEFINITIONS

The primary objective in this work is to compute, given a model of the task the robot should complete, a legible representation of the boundary between feasible and infeasible robot designs for that task.

To describe that problem more formally, in this section we present the concept of procrustean graphs and how we use them to model planning problems and plans. We also present the concepts of label maps that we use to model alterations to a procrustean graph, and a refinement relation that we use to navigate between different designs. These are condensed versions of detailed definitions that appear in our prior studies [11], [18].

A. Procrustean Graphs, Planning Problems and Plans

We model the world using a *procrustean graph* (*p-graph*) *G*. A p-graph is an edge-labelled bipartite directed graph which represents the robot's interactions with its environment. This interaction occurs 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. The states of the p-graph are partitioned into action states and observation states. For each action state, outgoing edges are labelled with a set of actions. Likewise, for each observation state, outgoing edges are labelled with a set of observations. We denote the set of all vertices in a p-graph by V, and the space of all possible observations and actions by Y and U, respectively. Each p-graph has an initial set of states, denoted V_0 .

A *planning problem* is a p-graph *G* equipped with a subset of states V_{goal} , called the *goal region*. A *plan* is a p-graph *P* along with a set of states V_{term} called the *termination region*. We say a plan (*P*, V_{term}) *solves* a planning problem (*G*, V_{goal}), if the plan, executed on the planning problem, can handle every observation it can receive from the world, and the planning problem can handle every action it receives from the plan, and if when the plan terminates, the current state of the planning problem should be within its V_{goal} . An example of a planning problem along with its p-graph representation and a plan that solves it are illustrated in Figure 2.

B. Label Maps and Destructiveness

We model the modifications to a robot's suite of sensors or actuators as changes to the p-graph representing the problem. A change in this suite could be expressed by transforming sets of actions and observations in the p-graph to new ones. We do so using the concept of label maps.



Fig. 2: [Top left] A grid environment with initial node *S*, goal region *G* and a set of obstacle cells. The robot can move in two directions, *up* and *right* and has a goal sensor where observation θ means the robot is not in the goal region and observation 1 indicates that the robot is in the goal region. [Top right] The problem of moving from *S* to *G* in the grid is encoded as a p-graph. Circle nodes are observation nodes and square nodes are action nodes. [Bottom] A plan that solves it.

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 observation map h_v and an action map h_u .

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). Applying a map h on a planning problem's p-graph is the way we model modifications. We are interested in determining whether a modification is destructive on a planning problem. That is, we want to see if applying the map precludes the existence of a plan to solve the planning problem. The following definition makes this precise.

Definition 1 (strongly destructive). In planning problem (G, V_{goal}) , if for every plan (P, V_{term}) that solves it, $(h(P), V_{\text{term}})$ cannot solve $(h(G), V_{\text{goal}})$, then we say h is strongly destructive.

To decide if there exists a solution for a planning problem (G, V_{goal}) , one can first convert the p-graph to its equivalent state determined expansion [11], and then run a backchaining algorithm on G. The algorithm establishes a solution incrementally by identifying the states in the planning problem from which we can guarantee to reach the goal, starting from V_{goal} .

C. Non-destructiveness boundary

Our goal is, given a planning problem, 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 give a definition of what this boundary is, we need the following definition.

Definition 2 (refinement relation). Given two maps, h: $U_1 \cup Y_1 \rightarrow U'_1 \cup Y'_1$ and $g: U_2 \cup Y_2 \rightarrow U'_2 \cup Y'_2$, 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. That is, 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. Now we can define the boundary of nondestructiveness.

Definition 3 (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 on (G, V_{goal}) 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.

Our objective is to construct a legible description of this boundary. The tool we use for generating a description of this boundary is a decision tree that classifies, based on some features, whether an input map is destructive or not. Decision trees are able to give insight into what actions and observations are the ones that determine on which side of this boundary a particular design lies. The tree induction process is described in Section V.

IV. TREE INDUCTION USING DOMAIN KNOWLEDGE

Our method for inducing a decision tree, as a legible description of the non-destructiveness boundary, consists of several steps. Feature extraction is one of the key elements of this process. Label maps, the entities we use to model modifications to sets of sensors and actuators, come with far too much information to be used as learning data. Instead, for each label map, we extract some features of it. Each feature vector will be tagged by the result of a destructiveness test, before inducing the decision tree.

In this section, we describe three scenarios and discuss how these scenarios are verified, and how we extract features for each map under each scenario. Formally, we define *classes of design problems* $\mathbb{D} = (P, M)$, in which P is a set of planning problems and M is a set of maps that can be applied on the planning problems in P. Within such a class, each instance is a *design problem*, consisting of a planning problem $(G, V_{\text{goal}}) \in P$, and the set of maps M, which is shared between all instances in the class. The idea of our approach is to generate, for one such design problem, a decision tree representation of the destructiveness boundary. In each of the following scenarios, we introduce a class of design problems and describe the defining elements of each.

A. Partially Affected Problems

A class of design problems $\mathbb{D}_{pap} = (P, M)$ contains *partially affected problems* if, for any $(G, V_{goal}) \in P$ with action space U and observation space Y, there exists a nonempty proper subset of events $E \subset Y \cup U$, such that for every map $m \in M$ and every element $e \in Y \cup U - E$, we have m(e) = e. In this case, the maps in M can only affect events in $Y \cup U - E$, which we call the *affected space*.

The intuition behind this class of problems may be seen in Figure 1. We are particularly interested in those partially affected problems where $|Y \cup U - E|$ is much smaller than $|Y \cup U|$, so the number of destructiveness tests will be small.

In this category, the feature vector for each given map *h* is computed by vector $F = [X_{pq}]$, for all $p, q \in Y \cup U - E$, $p \neq q$, in which X_{ab} is defined as:

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

The idea is to check if any of the observations or actions in the affected area are conflated with any other one. There is no need to check conflation between all pairs in $Y \cup U$.

For example, if for a map *h* and p-graph *G*, the affected space is $Y = \{y1, y2, y3, y4\}$ (that is, there are no affected actions), then the feature vector will be be calculated by computing each element in $[X_{y1y2}, X_{y1y3}, X_{y1y4}, X_{y2y3}, X_{y2y4}, X_{y3y4}]$. Each 1 in the feature vector indicates that the corresponding pair of actions (or observations) is distinguishable, and each 0 indicates they are indistinguishable.

B. Disjoint Events Problems

A class of design problems $\mathbb{D}_{dep} = (P, M)$ is called a class of *disjoint observation problems* if the following applies: for all $(G, V_{goal}) \in P$ and $m \in M$ the following set of conditions is true:

- 1) each observation appears only one time in *G*, as part of the label of some outgoing edge, and
- 2) each map $m \in M$ is injective on all observations.

Similarly, meeting the same conditions for action vertices, the class will be called a class of *disjoint action problems*. Refer to these two classes collectively as *disjoint events problems*.

For each given map h, in this case there is no need to extract features from the injective component of the map. This is true because applying an injective map does not bring any ambiguity to the corresponding set of vertices, and therefore, no destruction can happen. For the other component, we extract the feature vector for each map based on the structure of the p-graph G. If the given problem is a disjoint observation problem, for each map m we extract feature vector $F = [X_{pq}]$ where X_{pq} appears in F if there exists an action vertex v, for which p and q are possible actions that can be executed at v. The feature vector will be extracted similarly for disjoint action problems. The function X_{ab} is the same as previous section.

As an example, see Figure 3, which shows a part of the p-graph of a disjoint observation problem (G, V_{goal}) . The individual features for this part of the graph are X_{u1u2} , X_{u1u3} , X_{u2u3} , and X_{u4u5} for any given map $m \in M$.

C. Uncertain Events Problems

We define the *uncertainty degree* of an action in a p-graph G as the largest number of different outcomes that can occur for that action, across all of the action states. The definition is similar for the uncertainty degree of observations. The



Fig. 3: A part of the p-graph for a disjoint events problem. Incoming edges to the action states and the next states of the observation edges are omitted.



Fig. 4: A part of the p-graph for an uncertain events problem. Incoming edges to the action states and the next states of the observation edges are omitted.

largest uncertainty degree for all actions and observations of a p-graph is called its *uncertainty level*, denoted by K_G .

A class of design problems $\mathbb{D}_{K\text{-uep}} = (P, M)$ is considered a class of *uncertain event problems*, if for any $(G, V_{\text{goal}}) \in P$ with action space U and observation space Y, the uncertainty level K_G of G is at most K.

The idea is that the result of executing some action (or reading some observation) in a planning problem could be uncertain, but with limited possible outcomes. See Figure 11. A p-graph can capture this idea by having "undetermined" states where more than one outgoing edge is labelled with the same action or observation. Assuming that such a planning problem is solvable, we want to see if applying a map on it is destructive. To do so, after applying the map, we only need to check the states that these uncertain action or observations lead to, and see if their outgoing edges are labelled with conflated actions or observations.

For a given map *m*, we extract the feature vector $F_a = [X_{pq}]$ where X_{pq} appears in *F*, if and only if there exists an action vertex $v \in G$ and an action *u* such that *v* leads to vertices $w \in G$ and $w' \in G$ by action *u*, and *w* has an outgoing edge labelled with observation *p* and *w* has an outgoing edge labelled with observation *q*. Feature vector F_o is similarly calculated for observation vertices. Merging F_a and F_o creates the feature vector *F* that we use. X_{ab} is the same as the previous scenarios.

As an example, consider Figure 4. The individual features for a map that is applied on *G*, for this part of the p-graph are X_{y1y2} , and X_{y3y1} .

D. Decision Tree Induction

The different classes of design problems introduced in this section will be used in our experiments to induce a decision tree for the non-destructiveness boundary. To this end,



Fig. 5: The induction time, in seconds, versus the number states, using the general-purpose Naïve feature extraction method (red) and the method for partially affected problems (green). This experiment is done for the problems analogous to the one in Figure 1.

we use the Classification and Regression Trees algorithm (CART) [3]. The idea is that, for any given planning problem, we generate sample maps from the space of all maps for that problem. Then we extract features for each map, apply destructiveness test on all of these maps, and categorize each instance based on the outcome of the test. These instances will be used as the input to CART. As the output, CART gives a decision tree. In this decision tree, each internal node is labelled by a pair of actions or a pair of observations and each leaf is marked 'destructive' or 'non-destructive'. The expectation is that, in the tree, the more crucial pairs will show up in the higher levels.

V. EXPERIMENTAL RESULTS

For each class of design problems introduced in the previous section, we have performed a series of experiments, to evaluate their effectiveness in handling larger scale problems, compared to the general-purpose, naïve method from our prior work [10]. The experiments were performed on an Intel Core i7 machine with 32 gigabytes of RAM. To induce trees, we use scikit-learn package [16], which includes an implementation of CART. In all experiments, the depth of each induced tree is limited to 7 levels.

In the general-purpose method, one feature is extracted for each pair of distinct actions and each pair of distinct observations, regardless of the structure of the p-graph it is applied on. In other words, for a given map *h* over *n* actions and *m* observations, the feature vector is $F = [x_{u_iu_j}, x_{y_ky_\ell}]$ where $i, j \in \{1, ..., n\}, i \neq j$, and $k, \ell \in \{1, ..., m\}, k \neq \ell$. X_{ab} is the same as the one in our scenarios. For example, given a map *h*, with h(a) = a', h(b) = a', h(c) = c', h(d) = d', the feature vectors for *h* will be $[0 \ 1 \ 1 \ 1 \ 1]$.

A. Partially Affected Problems: Choosing an Antenna

The first experiment is performed on a p-graph representing the planning problem shown in Figure 1. Assuming the reference frame is at the bottom left corner of this subregion, the transmitters are positioned at $(x_1, y_1) = (1, 15)$, $(x_2, y_2) = (21, 1)$, and $(x_3, y_3) = (26, 16)$ where the distances are measured in meters. To understand how the choice of



Fig. 6: An induced tree for the problem shown in Figure 1. The observation tuples show the distance from transmitter 1, transmitter 2, and transmitter 3 respectively.

an antenna in the design process can influence the system, we assume that any change that can happen to this planning problem can only affect a portion of the observations, in particular, the observations in which the signals received from the third transmitter (far right) are beyond 25 meters. Sensor readings may get confused with each other outside this range. We can encode such changes in this problem as label maps such that the action component of each map is the identity map, and except for the observations from the affected subspace of observations, the observation map also acts like the identity map. These settings qualify this problem as an instance of \mathbb{D}_{pap} . Therefore, we can extract the feature vector for any such map using the method described in the previous section for \mathbb{D}_{pap} . We have limited the robot's set of actions to four moves: up, down, left, and right, where the actions are uncertain, i.e, each action may move the robot for 1 meter in the desired direction either one or two times. We have induced decision trees for a series of problems analogous to this problem with planning problem sizes ranging from 64 to 478 states, using both the naïve general-purpose feature extraction method and the method for partially affected problems. We measured how long it takes for the tree to reach 80% accuracy, by incrementally varying the size of the training set, using both feature extraction methods. The results, which are illustrated in Figure 5, show that using the domain knowledge improves the scalability. By way of example, an induced tree for this problem when the representing p-graph consists of 478 states is shown in Figure 6. The tree reflects what observations outside the reliable range of the third transmitter are more crucial to reach the goal in this case. The nodes on this tree show what observations within the affected space matter. The two observation on the root of the tree belong to a point in the goal region, $(d_1, d_2, d_3) = (11.40, 17.26, 25.06)$, and a point 1 meter below the goal region, $(d_1, d_2, d_3) = (12.37, 17.12, 25.55)$. In the second level of the tree, the same observation in the goal region, $(d_1, d_2, d_3) = (11.40, 17.26, 25.06)$, and another one within 2 meters below the goal region, $(d_1, d_2, d_3) =$ (13.34, 17.03, 26.08), appear on a node. These two nodes indicate that, among all possible pairs of observation in the affected region, only conflating the observations that lie in the goal region and the ones within a 2 meters distance from it matter. All other possible conflations are non-destructive. This insight implies, for example, that the robot designer should choose an antenna capable of reliably detecting the



Fig. 7: A rover inside a crater on Mars. To get recharged efficiently, the robot needs to get out of the crater from either the bottom left corner or the top right one, highlighted in two triangles.



Fig. 8: The induction time, in seconds, versus the number of states, using the general-purpose feature extraction method (red) and the method for disjoint events problems (green). This experiment is done for a set of problems analogous to the one in Figure 7.

third transmitter at this distance. Or, instead, it might be a good decision to use another cheap, low range source of signal along with the current choice of antenna, where this new signal source is placed close to the region where the observations from the third transmitter are unreliable.

B. Disjoint Events Problems: Determining the Crucial Actions of A Planetary Rover

The second experiment is performed on a p-graph representing the planning problem shown in Figure 7. In this example, a planetary rover must travel to one of the two goal regions to recharge. From a robot design perspective, this example solely focuses on the effects of possible conflation in the action space on the design process. Therefore, we consider changes where only actions may get conflated. The observations here determine the position of the robot, and the observation maps are assumed to be injective. Therefore, this problem can be considered as an instance of \mathbb{D}_{dep} . Similar to the previous example, we have limited the robot's set of actions to four moves: up, down, left, and right. However, here we assume the actions are certain: each action moves the robot for some *d* meters only once in the desired direction. The corresponding decision trees are induced for a set of



Fig. 9: An induced tree for the problem shown in Figure 7. Action here are up: u, down: d, left: l, and right: r.



Fig. 10: A robot equipped with a range sensor. The robot's goal is to move towards the wall and stop in the goal region (highlighted in a purple rectangle) before hitting the wall. The goal region lies between 20 and 40 centimeters from the wall. The robot's reference point lies at the front side of the robot. Some parts of the environment are always slippery, e.g., covered with ice (highlighted in blue) and make the robot's forward action uncertain, i.e., the robot does not know how long has it moved. This part lies within 100 and 140 centimeters from the goal region.

analogous problems, with the number of states in the pgraph varying from 50 to 2048 states, using the generalpurpose method feature extraction and also the method for disjoint events problems. Similar to the previous experiment, we measured how long it takes for the tree to reach a certain degree of accuracy, here 95%, by incrementally varying the size of the training set, using both methods. The results, illustrated in Figure 8, show that taking the domain knowledge into account can substantially decrease the induction time. An example induced tree when the number states in the p-graph is 800 is shown in Figure 9. Similar to the previous example, by confusing some actions, for example left and right, the robot will become unable of reaching the goal region. The interesting observation though, from a robot designer's point of view, lies in the symmetry of the planning problem reflected in the induced tree as well: conflation in actions left and up is not destructive unless the robot confuses down and right as well, and vice-versa. The robot can always go to one of the goal regions, for example by moving right and then up enough times. If right and up are conflated, then this can be done by moving left and then down enough times.

C. Uncertain Events Problems: Choosing a Range Finder

The last experiment is performed on a p-graph encoding the planning problem illustrated in Figure 10, in which a mobile robot attempts to park near a wall. To entirely focus



Fig. 11: The induction time, in seconds, versus the number of states, using the general-purpose feature extraction method (red) and the method for uncertain events problems (green). This experiment is done for a set of problems analogous to the one in Figure 10.



Fig. 12: An induced tree for the problem shown in Figure 10. Observations here are range finder readings in centimeters.

on the choice of range sensor the roboticist needs to make, we limit the set of actions to only one: move forward. The forward action is uncertain at the states that represent the part of the environment that is highlighted in blue. In this subregion, each forward movement may move the robot one, two, three, or four times, each time 20 cm in the forward direction, making the uncertainty level of the pgraph equal to 4. Therefore, the problem can be considered a \mathbb{D}_{4-uep} . Similar to our previous experiments, a decision tree is induced for this problem with different number of states varying from 600 to 1500 states. We used the generalpurpose method's feature extraction performance along with the method for uncertain events problems. Similarly, we measured how long it takes for the tree to reach a certain degree of accuracy (95%), by incrementally changing the size of the training set, using both methods. The results, showing the advantage of using the proposed method, are presented in Figure 11, and an induced tree for when the number of states in the corresponding p-graph is 1150 is shown in Figure 12. There is no action node on the tree due to our assumption that the only action movement is moving forward.

The insight that this tree provides helps the designer to decide if the range sensor that the robot is equipped with is sufficient for the planning problem. It can be seen that the range sensor should not confuse being in the goal region, 20 centimeters from the wall, with being 40 or 60 centimeters from it. A suitable range sensor for this planning problem needs to be reliable within these ranges.

VI. CONCLUSION

In this paper, our goal was to improve the practical scalability of a certain class of automated design tool, by incorporating domain knowledge, to help roboticists in designing robots. Specifically, we introduced three classes of design problems, and described the conditions for utilizing each. We performed a series of experiments for several problems. The results clearly demonstrate the advantage using domain knowledge in algorithmic robot design problems.

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

REFERENCES

- E. Acar and H. Choset, "Robust sensor-based coverage of unstructured environments," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.
- [2] P. K. Agarwal, A. D. Collins, and J. L. Harer, "Minimal trap design," in Proc. IEEE Int. Conf. on Robotics and Automation, 2001.
- [3] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.
- [4] A. Censi, "A Class of Co-Design Problems with Cyclic Constraints and Their Solution," *Robotics and Automation Letters*, Feb. 2016.
- [5] —, "A mathematical theory of co-design," arXiv preprint arXiv:1512.08055, 2015.
- [6] B. R. Donald and J. Jennings, "Sensor interpretation and task-directed planning using perceptual equivalence classes," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1991, pp. 190–197.
- [7] B. R. Donald, "On Information Invariants in Robotics," Artificial Intelligence, vol. 72, no. 1–2, pp. 217–304, Jan. 1995.
- [8] M. Erdmann and M. T. Mason, "An Exploration of Sensorless Manipulation," *IEEE Transactions on Robotics and Automation*, vol. 4, no. 4, pp. 369–379, Aug. 1988.
- [9] M. A. Erdmann, "Understanding action and sensing by designing action-based sensors," *International Journal of Robotics Research*, vol. 14, no. 5, pp. 483–509, 1995.
- [10] S. Ghasemlou, J. M. O'Kane, and D. A. Shell, "Delineating boundaries of feasibility between robot designs," in *Intelligent Robots and Systems* (IROS), 2018 IEEE/RSJ International Conference on. IEEE, 2018.
- [11] S. Ghasemlou, F. Z. Saberifar, J. M. O'Kane, and D. A. Shell, "Beyond the planning potpourri: reasoning about label transformations on procrustean graphs," in *Proc. International Workshop on the Algorithmic Foundations of Robotics*, 2016.
- [12] K. Y. Goldberg, "Orienting Polygonal Parts without Sensors," Algorithmica, vol. 10, no. 3, pp. 201–225, 1993.
- [13] S. M. LaValle, "Sensing and Filtering: A Fresh Perspective Based on Preimages and Information Spaces," *Foundations and Trends in Robotics*, vol. 1, no. 4, pp. 253–372, Apr. 2012.
- [14] M. T. Mason and K. Y. Goldberg and R. H. Taylor, "Planning Sequences of Squeeze-Grasps to Orient and Grasp Polygonal Objects," in Symp. on Theory and Practice of Robots and Manipulators, 1988.
- [15] J. M. O'Kane and S. M. LaValle, "On comparing the power of robots," *International Journal of Robotics Research*, vol. 27, no. 1, pp. 5–23, January 2008.
- [16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal* of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
- [17] F. Z. Saberifar, S. Ghasemlou, J. M. O'Kane, and D. A. Shell, "Set-labelled filters and sensor transformations," in *Proceeedings of Robotics: Science and Systems*, Ann Arbor, MI, Jun. 2016.
- [18] F. Z. Saberifar, S. Ghasemlou, D. A. Shell, and J. M. O'Kane, "Toward a language-theoretic foundation for planning and filtering," *The International Journal of Robotics Research*, 2018.
- [19] M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Domains," in *Proc. Int. Joint Conference on AI*, 1987.
- [20] B. Tovar, L. Guilamo, and S. M. LaValle, "Gap navigation trees: Minimal representation for visibility-based tasks," in *Algorithmic Foundations of Robotics VI*. Springer, 2004, pp. 425–440.
- [21] J. Wiegley, K. Goldberg, M. Peshkin, and M. Brokowski, "A complete algorithm for designing passive fences to orient parts," *Assembly Automation*, vol. 17, no. 2, pp. 129–136, 1997.