Every Action-Based Sensor

Grace McFassel and Dylan A. Shell

Department of Computer Science & Engineering,
Texas A&M University, College Station TX 77843, USA
gracem|dshell@tamu.edu

Abstract. In studying robots and planning problems, a basic question is what is the minimal information a robot must obtain to guarantee task completion. Erdmann's theory of action-based sensors is a classical approach to characterizing fundamental information requirements. That approach uses a plan to derive a type of virtual sensor which prescribes actions that make progress toward a goal. We show that the established theory is incomplete: the previous method for obtaining such sensors, using backchained plans, overlooks some sensors. Furthermore, there are plans, that are guaranteed to achieve goals, where the existing methods are unable to provide any action-based sensor. We identify the underlying feature common to all such plans. Then, we show how to produce action-based sensors even for plans where the existing treatment is inadequate, although for these cases they have no single canonical sensor. Consequently, the approach is generalized to produce sets of sensors. Finally, we show also that this is a complete characterization of action-based sensors for planning problems and discuss how an action-based sensor translates into the traditional conception of a sensor.

Keywords: Robot Design Problems, Abstract Sensors, Planning Problems

1   Introduction

In his paper *Understanding Action and Sensing by Designing Action-Based Sensors* [5], Erdmann[1] defines a class of abstract sensor that describes the information a sensor ought to provide a robot; his paper identifies a sort of canonical choice for such sensors. Summarizing that classic contribution to the literature, Donald [4] writes:

> In [5] Erdmann demonstrates a method for synthesizing sensors from task specifications. The sensors have the property of being "optimal" or "minimal" in the sense that they convey exactly the information required for the control system to perform the task.

But, as we will show, Erdmann's treatment may overlook certain sensor choices; indeed there may be multiple sensors which are equally "minimal", but only some of which have been considered in the past.

---

[1] For clarity when reading, we often refer to Erdmann by name when making reference to his theory of action-based sensors. Unless otherwise indicated, this is a reference to [5].

(a) A basic world, with two obstacles (shaded) and a single goal region G.

(b) A plan derived via backchaining from the goal.
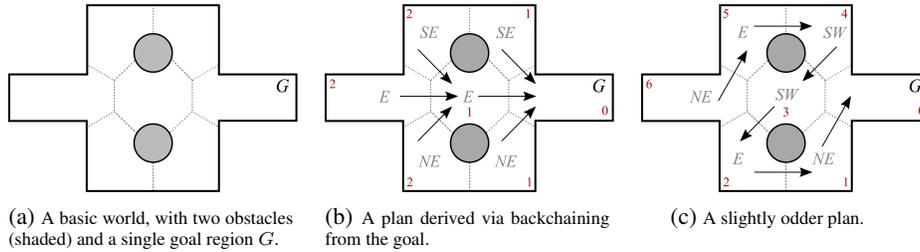
(c) A slightly odder plan.

Fig. 1: A world and two different kinds of plans on that world. Fig. 1b uses backchaining from the goal to form a plan, while Fig. 1c uses a Z-shaped path. This world and plans that solve it will be the basis for our discussion on the variety of plans and how this ultimately impacts sensor design. These plans can start in any region in the world, and each has a set of actions (in this case, singletons) that move them from state to state. Both plans terminate at G. The observations for each state are not included here, but are assumed to be distinct.

Generally, analyzing the information requirements of robotic tasks has yielded fundamental scientific insights in the past (cf. [1, 4, 6]). But making choices about sensors that are informed by information requirements is also important for practitioners, who need to balance considerations of cost, manufacturability, and reliability. Whether one's concern is purely theoretical or primarily practical, incompleteness is irksome.

Erdmann's sensors are defined using *progress measures*, which are real-valued functions on the state space of a planning problem that indicate how movement between states leads toward a goal. Given such a function, for each action, one labels regions of state space where that action makes progress, forming what are called *progress cones*. These regions must be distinguished sufficiently for the robot to determine which action to execute. This can be realized via *action-based sensors*, sensors that output actions guaranteed to make progress, which describe a subset of the progress cones containing the current state. As an abstraction of information attainment, such sensors do not specify which environmental features or associated technologies are actually used to compute (or evaluate) these functions. Erdmann formalizes the idea that information an agent needs is precisely and solely that which is needed to determine how to act now.

Action-based sensors embody the philosophy that sensors should be designed not to recognize states, only what actions must be taken to reach a goal. Utility in reaching goals is codified via progress measures and associated cones. These notions of progress are themselves computed from plans. The sequence goes like this: problems/tasks require plans to the solve them, plans give progress measures, measures give cones, and cones lead to sensors. But Erdmann focuses on backchained plans, a specific subclass of plans. It would seem that there could be broader sets of progress measures than he identifies. In fact, the situation is more dire, there are plans guaranteed to solve certain problems, but for which no Erdmann-like progress measure can be produced.

Figure 1a shows an example world. In this world, there are seven regions, each of which can be uniquely identified by the agent. Even in such a simple world, there are numerous ways an agent can get from any location to a goal, such as the two methods

presented in Figure 1. We will soon see that, despite the small world and simple plans, the question of how to develop action-based sensors is not quite so straightforward!

This paper extends the work done by Erdmann in several ways. First, we define a subset of planning problems and plans and show that within this subset, there are certain features, *crossovers*, that preclude the existence of progress measures. We precisely define these features and introduce a function, CLIP, that can transform a plan without a progress measure into multiple plans that do. We then show that these plans are capable of being used to define progress cones, which in turn can define action-based sensors. This obtains all action-based sensors, and we discuss how action-based sensors relate to more traditional concepts of sensing.

2 Preliminaries

The planning problem, or world for short, is a tuple $W = (V, V_0, V_{\mathrm{goal}}, Y, U, E)$ that consists of the following:

- The finite vertex set V, which is the set of states in the world.
- A set of initial states, $V_0 \subseteq V$, from where an agent may start.
- A set of goal states, $V_{\mathrm{goal}} \subseteq V$.
- A set of observations Y which label each vertex in V.
- A set of actions U.
- A set of edges E, each of which are labeled with $\{u_0, u_1, \ldots, u_k\} \subseteq U$.

An agent receives an observation $y \in Y$ which tells the agent where it is in the world. The agent then makes a choice of action that takes it (via an edge in E) from state to state, at which point it receives another observation and the process repeats. The agent begins in a state in V_0. Solving a planning problem demands that we develop a strategy for the agent that guarantees it will arrive in the goal region, i.e., some state in V_{goal}, when starting from any state in V_0.

A plan $P = (V, V_0, V_{\mathrm{term}}, U, Y, E)$ to solve a planning problem consists of:

- The finite vertex set V, which is the set of states in the plan.
- A set of initial states $V_0 \subseteq V$, from where an agent may start.
- A set of terminating states, $V_{\mathrm{term}} \subseteq V$.
- A set of actions U, subsets of which label each vertex in V.
- A set of observations Y.
- A set of edges E, which are labeled with $\{y_0, y_1, \ldots, y_k\} \subseteq Y$.

When discussing planning problems and plans simultaneously, we will use (P) and (W) to identify to which tuple an element belongs, e.g. $V(W)$, $V_0(P)$.

The preceding formalization of worlds and plans allows us to depict them as finite connected graphs. Figures 2 and 3 show this for our earlier example. Plans possess terminating states V_{term}, while worlds have goal states V_{goal}. A vital difference is that

Fig. 2: The example world from Fig. 1a as a connected graph. Vertices are labeled with observations **a–g**, while the actions include movement in the cardinal and intercardinal directions.

plan edges bear observations and vertices are labeled with sets of actions; world edges carry actions and vertices have a single observation. An agent following a plan tracks its plan state and may take any action in the set labeling the current state. After taking an action, it receives an observation from the world, and transitions to a new vertex, itself labeled with actions, and so on. Every plan's initial vertices V_0 are labeled with the empty set, as the agent obtains a first observation to establish information about the world before taking its first action. Only elements in V_0 and V_{term} have the empty set.

For a plan P to be a solution to planning problem W, the plan must satisfy a few constraints: it must be *safe*, in that it never attempts to take an action at a plan state if that action is unavailable at the current world state. Further, the plan must be ready to receive any observation that arises consistent with possible paths from the V_0 of the world. An *execution* is a sequence of alternating observations and actions, beginning and ending with an observation. They describe the interaction between a plan and world, giving rise to a path. Either this path must lead to states that are in V_{term} on the plan and in V_{goal} on the world, or the execution must be a prefix of a longer execution which does. If an execution is possible on both plan and world, it is a *joint-execution*. The plan must also be *finite on* the world, in that all joint-executions must have bounded length.

The *language* of a plan and world pair is the set of all joint-executions. Occasionally, we are interested in where a given execution will take us in the world. We *trace* an execution $s = y_0 u_1 y_1 u_2 y_2 \ldots$ over the world by beginning at the vertex in V_0 labeled

(a) A plan derived via backchaining from the goal.

(b) A slightly odder plan.

Fig. 3: Our two plans from earlier, now as connected graphs. Arrows from nowhere leading to vertices are used to indicate an agent that waits for an initial observation, and then transitions to a state in the plan consistent with that initial observation. Starting edges with observations have been rendered in a lighter color to make the overlay of plans clearer.

with y_0 and following the edges consistent with the actions in the execution. In the event of non-determinism in actions leading to multiple possible states, the following observation in the execution clarifies the resulting state in the world.

The world state reached by this method is the vertex in the world obtained by tracing a sequence s on W, designated $\mathcal{V}_s^W$. (We will also consider tracing on a plan, and it is defined analogously.)

2.1   Scope

Consistent with Erdmann's treatment, in this paper, we consider planning problems that start from any state ($V_0 = V$) and which have a single goal state ($|V_{\text{goal}}| = 1$)[2]. In addition, also consistent with Erdmann, we examine planning problems that are fully observable. This is because the method we use to define action-based sensors coarsens the original observation set. Partially observable planning problems implicitly contain information about how the world is divided by a sensor; if, in the input, states cannot be distinguished, then there is no hope for a sensor that will be able to, either. Thus, to enumerate the full set of sensors, the input should be fully observable.

Some of the definitions and theory discussed throughout have overhead for handling more complicated planning problems than those just defined. Though we maintain definitional generality, restricting our set of planning problems simplifies discussion significantly. We show later that this subset is sufficient to capture all action-based sensors. Relaxing these restrictions often results in problems that result in objects that are not quite action-based sensors in Erdmann's sense or which require additional interpretation to be meaningful, and are therefore beyond the scope of this paper.

3   What It Means to Make Progress: the Progress Measure

To determine how plans make progress toward a goal, we start with defining what it means to make progress. Erdmann uses a framework of progress measures to develop progress cones. Given a task, to get from planning problems via progress to sensors, Erdmann [5] prescribes:

> " 1a. Determine a sequence of actions that accomplishes the task.
> 1b. Define a progress measure on the state space that measures how far the task is from completion, relative to the plan just developed.
> 1c. For each action, compute the region in state space at which the action makes progress. "

The first step requires one to construct a sequence of actions, and subsequently in his paper Erdmann uses a very specific kind of plan when discussing progress measures — those obtained via backchaining from the goal. However, plans created using

---

[2] In particular, knowledge of the starting location provides information in the form of context to incoming observations, allowing an agent to gather information it might not otherwise have.
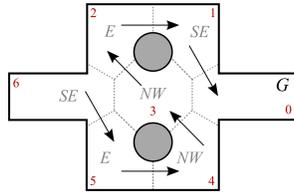
(a) The mirror image of the Z-shaped plan: the S-shaped plan.
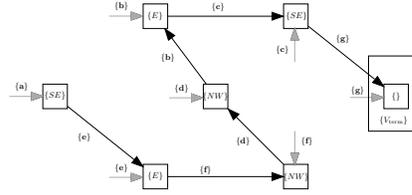
(b) The new plan, as connected graph.

Fig. 4: A new plan, similar to the one seen in Figure 1c. Although the S-shaped plan takes the same action as the Z-shaped plan at numerous states in the world, there are some states for which they are moving in opposite directions from each other.

backchaining yield a unique progress measure corresponding to the fastest strategy which Erdmann calls "very special". Our agenda is to broaden this set of plans, and doing so has implications for progress measures. In particular, we require a little more nuance, which manifests as two separate definitions in what follows.

Definition 1 (execution progress measure). *A* progress measure over executions *on a solution P to a planning problem W is a function* $\phi : 2^{V(W)} \to \mathbb{R}^+$ *such that:*

a) $\phi(V) = 0 \implies V \subseteq V_{\text{goal}}$;
b) *at least one* $V \subseteq V_{\text{goal}}$ *satisfies* $\phi(V) = 0$;
c) *for any two joint-executions p and q, if p is a prefix of q, then* $\phi(\mathcal{V}_p^W) > \phi(\mathcal{V}_q^W)$.

The execution progress measure applies to sets of vertices in the world. All sets that take the value of 0 are required to have only goals within them. There must also be at least one goal with value 0. We restrict joint-executions, requiring that if one is a prefix of another, its value must be strictly greater than. In this way, the executions of the plan visit states in the world such that the resulting progress measure is strictly decreasing.

Definition 2 (vertex progress measure). *A* progress measure over vertices *on a solution P to a planning problem W is a function* $g : V(W) \to \mathbb{R}^+$ *such that* $\phi_g(V) := \max_{w \in V} \{g(w)\}$ *is an execution progress measure.*

The intuition here is to give a measure on singleton states and require that we get an execution progress measure when it is lifted, in a natural way, to sets. For the purposes of this paper, the distinction between execution and vertex progress measures is irrelevant as we focus on small set of plans (see Section 2.1) due to space limitations. The overhead of two separate definitions hints toward the generalization, so we have deemed to retain the distinction.[3] In light of this, we write *execution progress measure* or *vertex progress measure* as just *progress measure*, and context will resolve any ambiguity when necessary.

---

[3] In this paper, we are considering fully observable planning problems and plans which may be at only one state in the world during any point of execution, for which an execution progress measure and vertex progress measure are equivalent. However, there exist plans in which an agent may be in multiple potential world states, for which both the execution progress measure and vertex progress measure are required to define progress-making actions.

Progress measures can be seen in the red numerals in Figures 1b and 1c. Progress measures are defined in terms of the plan's actions on the world, which can lead to a measure in which making progress leads away from the goal (in terms of distance) before reaching it. Such an example can be seen in Figure 1c, exemplifying the difference between a progress measure and a distance metric from the goal.

3.1   Lack of Uniqueness in Progress Measures and Crossovers

Figure 4 shows the mirror twin of a familiar plan. These two plans have some states where they take the same action, and some where they differ, creating different progress measures. There is no reason to prefer one of these plans over the other. We could even have a plan that considers both of the routes such as the plan in Figure 5, which chooses one of the two paths at random and commits to that path once it has been selected. However, an issue arises when we attempt to develop a progress measure for this plan. The issue stems from the fact that, though the plan informs our definition of progress, the progress measure is based on states in the world.

Figure 5's plan contains executions that go in opposite directions from each other; for example, one path visits the state labeled **d** before the state labeled **e**, while another path visits the state **e** before **d**. While the plan is not structured such that the agent could actually loop infinitely between states **d** and **e**, the progress measure, considering only the corresponding world states, has the impossible task of satisfying $\phi(\mathbf{d}) < \phi(\mathbf{e}) < \phi(\mathbf{d})$.
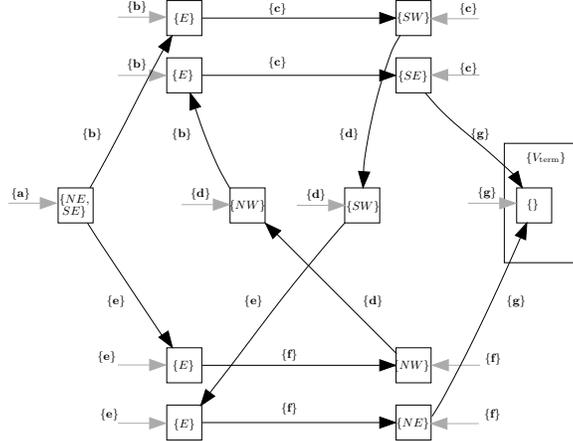

Fig. 5: The S-shaped and Z-shaped plans, presented together. This plan has numerous crossovers within it. The lack of a global ordering on when states are visited is what defines a crossover.

Progress measures fail to exist when there are contradictory requirements on the values that the execution progress measure must take. We refer to this issue as the *crossover conflict*, or simply as the existence of *crossovers*, due to the fact that the lack of progress measure extends from the fact that plan executions "cross over" each others' paths when traced over the world. In this way, crossovers can be thought of as

loops induced on the world by the plan. A plan can distinguish between multiple visits to a single world state by having multiple distinct plan states. However, the progress measure's definition considers all potential paths to and from a world state when defining values. Therefore, executions that visit states in differing orders create these loops, resulting in a malformed progress measure.

Definition 3 (crossover conflict). *A plan P on a planning problem W has a* crossover conflict *if it has two executions s_1, s_2 in the language of the plan L_P such that:*

1. *s_1 and s_2 both, during their execution on P, visit the states S and S' in W, and*
2. *s_1 requires an execution progress measure where $\phi(S) > \phi(S')$, while s_2 requires an execution progress measure where $\phi(S') > \phi(S)$.*

To identify crossovers, the relationship between plan states, plan executions, and world states must be made explicit. One method for this is through enumeration of a plan's language and tracing that language on the world. In Section 4, we discuss the use of a graph to identify crossovers and to avoid the need to enumerate the entire language of the plan. We are interested in identifying crossovers as their presence is the necessary and sufficient condition for the non-existence of a progress measure.

Theorem 1. *A progress measure exists iff there is no crossover within the plan.*

  Proof:

  $\Longleftarrow$ **A Crossover Implies no Progress Measure Exists.** The previous discussion has shown that the existence of a crossover induces an unsatisfiable condition on the progress measure.

  $\Longrightarrow$ **The Lack of a Progress Measure Implies a Crossover Exists.** Consider a plan P that solves a planning problem W, and which lacks a progress measure. Then, by definition of the execution progress measure, one of the following must be true:

  (a) $\phi(V) = 0, V \not\subseteq V_{\text{goal}}$.
  (b) There is no $V \subseteq V_{\text{goal}}$ where $\phi(V) = 0$.
  (c) There exist joint-executions p and q with p a prefix of q, and $\phi(\mathcal{V}_p^W) \leq \phi(\mathcal{V}_q^W)$.

  Think of the progress measure not in terms of a function, but instead as an ordering on the states of the world. At least one goal must come last in the ordering, taking the value of 0. For executions, the requirement of prefixes having a higher measure than sequences which they precede imposes an ordering on those states, as well.
  If we start trying to fix the progress measure, we first assign the goal (the final element in the ordering) to value 0 and increment up as we go earlier in the order. This resolves the issues presented by (a) and (b), should they exist.
  Assume we try to correct (c) in such a way. If we are able to do so, obeying the induced ordering and assigning values to the states reached by p and q such that they no longer fulfill the condition of (c), then a progress measure does in fact exist. However, if it does not, then, because we were assigning values to the ordering of states based on back-tracking from the goal, we must have seen the state reached by p in the ordering before the state reached by q and assigned it a value accordingly. If p is a prefix, that means the state reached by p is visited by an execution both before *and* after it visits the state reached by q. Therefore, there is an unsatisfiable requirement of $\phi(\mathcal{V}_p^W) \leq \phi(\mathcal{V}_q^W) \leq \phi(\mathcal{V}_p^W)$, which is a crossover.

□

Crossovers therefore are the cause of failure in a plan that does not produce a progress measure, impeding our ability to craft action-based sensors. Recall in our example that the plan without a progress measure is created by combining two plans that do. In fact, the choice of action is what permits crossovers in the plan, but not all choices in the plan lead to crossovers. In Section 4, we present an algorithm that identifies such choices and generates plans with progress measures when given a plan without.

4   Removing Crossovers and Enumerating Progress Measures

Given a plan P that solves a planning problem W and which has no progress measure, we now discuss a method to produce the set of all plans which can be derived from this initial plan, which have progress measures, and which are also solutions to the planning problem. To make precise what we mean when we say one plan is *derived from* another, we define a set of actions from the world called the *operative action* set.

Definition 4 (operative action set). *For a plan P that solves W, we define the* operative action set *of P as a function $\mathfrak{u}_P : V(W) \to 2^{U(W)}$ such that $\mathfrak{u}_P(v)$ includes an action u_k if and only if P and W have a joint-execution $j = y_1 u_1 y_2 \ldots u_{k-1} y_k$ for which: j arrives at v when traced on W; action u_k is a label on an outgoing edge from v; j arrives at a state labeled with u_k when traced on P.*

Definition 5 (derived plan). *A plan P' is* derived from *another plan P if the operative action set of P' is contained in the operative action set of P, that is for all $u \in U(W)$, $\mathfrak{u}_{P'}(u) \subseteq \mathfrak{u}_P(u)$.*

The operative action set for a plan associates each world state with a subset of that state's outgoing actions — those that are potentially used by the plan. Crossovers imply a cyclical dependency between world states, and therefore to identify them we need to find the correspondence between plan states and world states and clip away these cycles. A solution is to take the entire set of operative actions and generate all possible combinations, rejecting options that aren't solutions to the planning problem or which do not have progress measures. This is a naïve approach because we don't need to consider every possible edge in the plan to remove crossovers, but only those edges that are involved in the crossover itself. For edges that are not involved in any crossover, we may take any subset of them so long as we abide by our constraints. Therefore, we propose an algorithm to generate representatives for the plan that show all possible ways of breaking crossovers without regard to the other edges.

To accomplish this, we present the algorithm CLIP. It has two parts: first, given a plan and a planning problem, it creates a new graph structure based on the executions of the plan and the corresponding world states. We call this structure the *Plan-World Interaction Graph*, or simply the *I-Graph*.

The I-Graph is a graph consisting of three layers: the initiating layer, the plan layer, and the world layer. The initiating layer is a single vertex. The edges of the initiating layer transition to the plan layer. They are labeled with observations that the agent may receive upon beginning execution. Each vertex in the plan layer is labeled with a pair, the first element being a subset of plan states and the second being the single world state to which they all correspond. This layer is called the plan layer because, from this layer, the outgoing edges are actions: this is the layer of the I-Graph for which the plan makes a choice. These edges go to the world layer, which corresponds to the world's choice. A single action from a state in the world may result in several different observations depending on the outcome. The world 'chooses' which of these observations is obtained. The edges from this layer go back to the plan layer to a new set of plan and world states. An example appears in Figure 6.
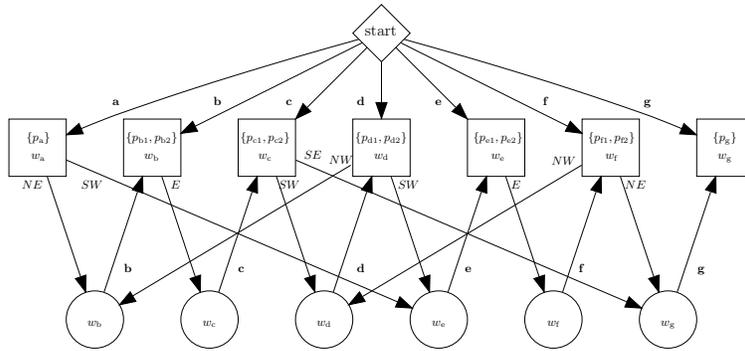

Fig. 6: An I-Graph for the plan seen previously in Figure 5. Plans may have multiple vertices that correspond to the same state in the world. The I-Graph merges these into a single vertex.

By finding where multiple plan states act on the same world state, the algorithm constructs a representation of the operative action set on the world. In addition, because plan states that were distinct correspond to a single vertex in the I-Graph, loops that previously existed only in terms of the progress measure now become explicit.

Lemma 1. *The set of all actions for I-Graph I generated from plan P has the same operative action set as P on the original planning problem W.*

Proof: The construction associates world states by tracing P's executions.   □

Our goal, and the function of the second part of CLIP, is to use this I-Graph to develop plans with progress measures. We define the *comes-before* relation, which indicates which world states precede each other during execution of the original plan P. Crossovers are now easily identified as loops in the graph of the I-Graph, and we can both determine which world states are involved in crossovers and how many crossovers exist. For each crossover found, the loop in the I-Graph must be clipped. However, there

---

Algorithm 1: Construction of the Interaction World

Require: Inputs: World W, Plan P
  I-Graph = New Graph()
  association_queue = {P.init_states, W.init_states}
  while association_queue is not empty **do**
    current_state = association_queue.dequeue()
    plan_states = current_state[0]
    world_states = current_state[1]
    for each p in plan_states **do**
      for each outgoing_edge in p **do**
        if outgoing_edge has equivalent action from world_states on world_edge **then**
          new_plan_states.add(outgoing_edge.target)
          new_world_states.add(world_edge.target)
        end if
        if new_world_states is a set already in the plan as q **then**
          add edge to I-Graph: current_state, outgoing_edge, q
        else
          add new vertex to I-Graph: new_world_states
          add edge to I-Graph: current_state, outgoing_edge, new_world_states
        end if
        association_queue.enqueue({new_plan_states, new_world_states})
      end for
    end for
  end while
  return I-Graph

---

are many ways to clip a cycle, and cycles may even overlap with each other. We wish to enumerate all possible ways to resolve these crossovers.

CLIP constructs a search tree starting from the original I-Graph. For each crossover found, CLIP generates a set of candidate edges. Candidate edges are those that transition between world states in the loop and come from a world state that has more than one outgoing action. This second requirement is because, should we remove all outgoing actions from a world state, any resulting plan cannot reach the goal from that state and therefore is not a solution.

The search tree considers the powerset of the candidate edge set for removal. For each set of edges from the powerset, we create a new child where these edges have been removed from the I-Graph and move on to the next loop. The empty set is included in this search as loops may share edges or states in common, and therefore the choice of removing no edges by a single loop effectively defers the choice of which edges to remove to later. Because CLIP only removes edges, any I-Graph that has isolated a world state is invalid and is removed from consideration. To improve the search, nodes in the search tree are checked to see if they are invalid before CLIP generates their children.

We now examine the output of CLIP. CLIP produces numerous subgraphs of the I-Graph, each of which has its own method of resolving the crossovers. We use these to define a set of plans: the *representative plan* for an I-Graph is the plan which has the same operative action set as the I-Graph used to generate it.

Theorem 2. *All plans generated by* CLIP *using a plan P are derived from P, have progress measures, and solve the planning problem W.*

Proof:

1. **All plans generated by** CLIP **are derived from the plan P.**
   CLIP only removes edges from an input plan, and cannot add actions to output plans that are not part of the input. CLIP creates an I-Graph, which via Lemma 1 has the same operative action set as P, and then generates plans from subgraphs. Therefore, any plan CLIP generates must have an operative action set that is equivalent to P, or a subset.
2. **All plans generated by** CLIP **have a progress measure.**
   By the comes-before relation, CLIP verifies that no loops in the world exist before acceptance. As lack of looping is indicative of a lack of crossovers and therefore sufficient to indicate existence of a progress measure, all plans generated by CLIP have a progress measure.
3. **All plans generated by** CLIP **solve the planning problem.**
   By definition, before accepting any solution, CLIP calculates the comes-before relation. If, for any world state, that state does not 'come before' at least one goal state, CLIP rejects it. CLIP also rejects any plans with loops still present. Therefore, any plan CLIP accepts is a solution. □

We call the output plans of CLIP the *representatives* of the set of all solutions, so named because any plan in this set of desired solutions that is not generated by CLIP directly can be derived from a representative itself.

Theorem 3. *All plans derived from* CLIP*'s representatives are in the solution set.*

Proof: We define plans derived from CLIP's results as any plans constructed from a subset of edges from a result produced by CLIP. Excepting such plans that are not a solution to the planning problem, a plan with some of its edges removed will remain a solution. In addition, as removing action edges cannot induce a loop on a plan that does not have one, the resulting plan keeps a progress measure, ensuring that it is also part of the solution set. □

Theorem 4. *Every plan in the solution set can be derived from a representative plan.*

Proof: Assume that there is a plan P' that is not generated by CLIP nor derived from CLIP's solutions. Then P' must solve the original planning problem W, use actions

found only in the operative action set of P, have a progress measure, and not be a representative produced by CLIP or derived from these results.

We define two sets: E^{loop} and E^{static}. E^{loop} is the set of edges in the world that are part of the operative action set of all loops in the I-Graph. By definition, CLIP considers E^{loop} as candidates for removal. E^{static} is the set of all other action edges in the operative action set. As P' uses the operative action set that is also used to generate the solutions of CLIP, it must differ from any given plan provided by CLIP in E^{static} or E^{loop} (or both).

If P' differs in E^{static}, it can be in one of two ways: either it contains an element not in any representative produced by CLIP, or it is a subset of any given representative's E^{static}. If it is a subset, then P' is actually derived from that representative. If it contains an element not in a representative, then that element is not from the operative action set, and P' is not truly in the solution set, as CLIP does not remove any edges from E^{static}. Therefore all representatives generated by clip contain the entire set of E^{static} from the original plan P.

If P' differs from E^{loop}, it can be in one of two ways: if it has an extra edge, then it either contains an action not from the operative action set or it still contains a loop that causes its language with the world to not to be finite. If it is smaller than any representative, than it implies CLIP does not enumerate all possible values of the loop edges. As CLIP enumerates all loop edge possibilities (through generating the powerset), it must enumerate all possible values of the loop edges, so this is a contradiction.   □

The input plan therefore yields numerous representatives, all of which have progress measures and which can be used to generate the entire set of plans of interest. Having resolved the issue of plans without progress measures, we next turn to the question of how to use progress measures to obtain sensors.

5   Translating Progress Measures into Sensors

Once we have obtained a plan with a progress measure, we want to use it to link actions with how to make progress. We achieve this through defining a *progress cone*, a term inherited from Erdmann. Every action $u \in U$ has an associated progress cone. This progress cone is a set of observations. At any state in the world labeled with an observation in this set, the action u makes progress toward the goal (transitioning from a higher-valued state to a lower-valued state) according to the progress measure.

Definition 6 (progress cone). *For a planning problem W and plan P with a progress measure ϕ, the* progress cone of an action $u \in U(W)$ *is the largest subset of $Y(W)$, $\{y_1, y_2, \ldots, y_k\}$ where u makes progress under ϕ, from all states labeled with some y_i.*

Two views of the cones are possible. The first, which is natural from the preceding definition, maps from actions to states (or, equivalently, observations); the second asks

which actions make progress at a state (or given an observation). Since both views are useful, we consider the progress cone to be a relation between observations and actions so that each observation has an associated set of actions that make progress.

Definition 7 (cone relation). *For a planning problem W and plan P with a progress measure ϕ, the* cone relation $\mathbf{C} \subseteq Y(W) \times U(W)$ *contains (y, u) if there exists a progress cone for the action u containing y. We will also write $y \underset{\mathbf{C}}{\sim} u$, when $(y, u) \in \mathbf{C}$.*

As the planning problem must be solved from any location in the world, any observation y that is used as a label of a state in $V(W)$ must have at least one action u where $y \underset{\mathbf{C}}{\sim} u$. This forms a covering over the set of observations $Y(W)$.

For an observation, there are potentially many progress-making actions. However, only one action is needed for any given y to guarantee that the agent will eventually arrive at the goal. We can define a class of functions that, for each observation y, return a single action u, transforming the covering into a collection of partitions. We call these functions *singleton action-based sensors*.

Definition 8 (singleton action-based sensor). *A function $f : Y(W) \to U(W)$ is a* singleton action-based sensor *if $y \underset{\mathbf{C}}{\sim} f(y)$ for every y.*

The connection between singleton action-based sensors and real sensors is not immediately apparent. A "traditional" sensor can be represented as a function $s : V(W) \to Y(W)$, taking world states as inputs and returning some observation.

To bridge the gap we define a new set of observations $Y' = \{y'_u \mid u \in U(W)\}$ by making a correspondence of each element to an action, as indicated by the subscript. For an element $y \in Y(W)$, $y \mapsto y'_u$ if $f(y) = u$ according to a singleton action-based sensor. Recall that in our framework, each element in Y mapped to a single state in the world, so we can think of the elements of Y as a kind of stand-in for the world states of W. $W \to Y \to Y'$ is therefore equivalent to $W \to Y'$, and takes in states in the world and maps them to this new set.

The process above transformed a covering into a partition through use of a function. However, perhaps we would like multiple (or even all) possible progress-making actions for a single observation. To achieve this, we define *permissive action-based sensors*.

Definition 9 (permissive action-based sensor). *A function $f : Y \to 2^U \setminus \{\varnothing\}$ is a* permissive action-based sensor *if, for every $u \in f(y)$, $y \underset{\mathbf{C}}{\sim} u$.*

The relationship of this object to more traditional sensors is less clear than for a singleton action-based sensor. We can construct a Y' as before, where now each element y' corresponds to some subset of U, but the semantics of actions within multiple different sets becomes a matter of interpretation.

6   Completeness

We have presented the limitations of the existing theory of action-based sensors and extended it to cover a broader group of plans. As this paper attempts to fill a gap in Erdmann's treatment of action-based sensors, there is the question of whether the proposed method is itself comprehensive. We claim that the method presented here captures all action-based sensors obtainable for any given plan P that solves a planning problem W satisfying the criteria in Section 2.1.

We Capture All Action-Based Sensors. *Proof:* Assume there is some action-based sensor that is not obtained by the method described above. This action-based sensor maps from the set of observations Y to sets of actions $2^U \setminus \{\varnothing\}$. This sensor must be usable to solve the planning problem. Therefore, this sensor must give an action that proceeds toward the goal from each state in the world. This induces an ordering on the states, which means that a progress measure can be created on the world using this sensor. As this progress measure gives an action from each state in the world, it prescribes a plan that is capable of starting from any location. But then it is included in the set of plans that we consider. Therefore, this action-based sensor is obtainable. $\square$

7   Implications and Conclusion

Our motivation for revisiting[4] action-based sensors stems from an interest in what sensing, fundamentally, *is.* Often we take sensors for granted: as a distance sensor, or wall sensor, and so on. But as Brooks and Matarić [2] note: "The data delivered by sensors are not direct descriptions of the world. They do not directly provide high level object descriptions and their relationships." How we use the word "wall" with easy abandon! These mental categories are ingrained so deeply as to have a pernicious influence on our thinking.

In conceiving his theory, Erdmann asked the question of what sensors are *for.* The action-based sensor, then, relates what a robot should do with what it needs to perceive. The approach conceptualizes sensors as abstractions which entirely sidestep issues with the representation of information to provide what is required: what action to take next. His definition appeared to give the utmost leeway in its requirements, being most re-laxed or unconstrained so the set of sensors seems to be maximally inclusive — forming a sort of 'free object' for sensors. It is hardly surprising, then, that little work has sought to expand directly upon Erdmann's highly-original paper, for it looks to be the final word on the subject.

Characterizing the space of sensors is particularly relevant for approaches to au-tomate robot design. Work such as [3] addresses the problem of satisfying numerous constraints simultaneously, where the question of how to express both constraints on sensing and their utility to completing the desired task is crucial. CLIP produces numer-ous sensor options, some of which may be more realizable than others. By comparing

---

[4] Or perhaps resurrecting?

these sensor specifications to libraries of actual components, practitioners can make informed choices in terms of sensing information that is useful to the agent. The question of suitability now becomes one of intersecting constraints.

One caveat to this is that progress measures, by their nature, do not make use of state. Our example plan with crossovers chooses a path at the start of execution, and encoding of that decision is maintained within the plan structure itself. The additional states are required for this plan to succeed. Rather than thinking of crossovers as simply loops to untangle, they imply that execution of a given plan requires some internal memory in order to make progress toward the goal. The idea of sensors that maintain information of state in order to give relevant actions goes well beyond what we typically consider when discussing a sensor. The authors are interested in pursuing how to define and make use of these "stateful" sensors for more complex behavior. In addition, the existence of these plans and their sensors implies a structure in the complexity of plans and the resulting sensors.

References

[1] Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978.

[2] Rodney Brooks and Maja Matarić. Real robots, real learning problems. In *Robot learning*, pages 193–213. Springer, 1993.

[3] Andrea Censi. A mathematical theory of co-design. *arXiv preprint arXiv:1512.08055*, 2015.

[4] Bruce Donald. On Information Invariants in Robotics. *Artificial Intelligence — Special Volume on Computational Research on Interaction and Agency, Part 1*, 72 (1-2):217–304, January 1995.

[5] Michael Erdmann. Understanding Action and Sensing by Designing Action-Based Sensors. *The International Journal of Robotics Research*, 14(5):483–509, October 1995.

[6] Jason M. O'Kane and Steven M. LaValle. Comparing the power of robots. *The International Journal of Robotics Research*, 27(1):5–23, January 2008. doi: 10. 1177/0278364907082096.