Journal Title
XX(X):1–20

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Reactivity and statefulness: Action-based sensors, plans, and necessary state

Grace McFassel and Dylan A. Shell[1]

Abstract

Typically to a roboticist, a plan is the outcome of other work, a synthesized object that realizes ends defined by some problem; plans qua plans are seldom treated as first-class objects of study. Plans designate functionality: a plan can be viewed as defining a robot's behavior throughout its execution. This informs and reveals many other aspects of the robot's design, including: necessary sensors and action choices, history, state, task structure and how to define progress. Interrogating sets of plans helps in comprehending the ways in which differing executions influence the interrelationships between these various aspects. Revisiting Erdmann's theory of action-based sensors, a classical approach for characterizing fundamental information requirements, we show how plans (in their role of designating behavior) influence sensing requirements. Using an algorithm for enumerating plans, we examine how some plans for which no action-based sensor exists can be transformed into sets of sensors through the identification and handling of features that preclude the existence of action-based sensors. We are not aware of those obstructing features having been previously identified. Action-based sensors may be treated as standalone reactive plans; we relate them to the set of all possible plans through a lattice structure. This lattice reveals a boundary between plans with action-based sensors and those without. Some plans, specifically those that are not reactive plans and require some notion of internal state, can never have associated action-based sensors. Nevertheless, action-based sensors can serve as a framework to explore and interpret how such plans make use of state.

Keywords

Abstract sensing, a theory of; Stateful/Reactive plans; Structured plans; Action-based sensors; Plan state semantics

1 Introduction

In his venerable paper *Understanding Action and Sensing by Designing Action-Based Sensors*, Michael Erdmann[1] defines a class of abstract sensor that describes the information a sensor ought to provide a robot; his paper identifies a type of canonical choice for such ideal sensors. Summarizing that classic contribution to the literature, Donald (1995) writes:

> Erdmann (1995) 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.

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 defined via *progress measures* and associated *progress 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.

Erdmann's work focuses on a subclass of all plans, those created from backchaining, which yield "special" sensors. The backchained plans in his work codify the fastest way to reach the goal from any starting location. This naturally leads to the question of what would happen if one applied this method to other types of plans, as it would seem that doing so would open up a much larger family of action-based sensors. We shall show that this is indeed the case.

Part of our motivation for exploring new families of sensors is that generally, analyzing the information requirements of robotic tasks has yielded fundamental scientific insights in the past (cf. (Blum and Kozen 1978; Donald 1995; O'Kane and LaValle 2008)). Construction of Erdmann's "minimal" sensors assumes that all information needed for task completion can be sensed in the environment. If the information available is insufficient, then the robot must make use of *state*. Through extension of the problem to other plans, we see that the required memory of the robot and the abilities of its sensors are linked in a way that different action-based sensors give a new, unique way to explore.

Moreover, 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 (Censi 2015; Zhang and Shell 2020). Considering additional plans helps make the theory of action-based sensors more applicable for

Department of Computer Science & Engineering, Texas A&M University, College Station TX 77843, USA

Corresponding author:
Grace McFassel, Department of Computer Science & Engineering, Texas A&M University, College Station TX 77843, USA.
Email: gracem@tamu.edu

roboticists by, for instance, allowing one to model some limits imposed by physical or technological constraints.

The theory's incompleteness, in not being able to obtain all action-based sensors, is irksome. This is true, whether one's concern is primarily theoretical (previously overlooked sensors that have an equal claim to being "minimal") or is purely practical (sensors that can respect design constraints). The first part of this article, thus, is concerned with identifying further action-based sensors through considering the full set of plans for a given planning problem.

This analysis-oriented treatment differs from the typical approach to plans, which adopts a synthesis standpoint that asks how to find plans to realize some ends. Instead, the concept of a plan is used as the basis for the design of sensors. We present an algorithm, CLIP, that transforms a plan for which we cannot define an action-based sensor into a set of plans for which we can. We extend our earlier theoretical work (McFassel and Shell 2020), and associated algorithmic methods and our implementation, to allow consideration of partially observable planning problems. Such cases lead to the identification of certain plans that are guaranteed to solve some given planning problem, but for which no Erdmann-like progress measure can be produced.

Such plans are the aforementioned plans that require state. To discuss the relation between these "stateful" plans and action-based sensors, we first define a lattice structure with which to organize the set of all plans. Action-based sensors are then placed within the lattice as a type of reactive plan. The action-based sensors to which a plan is related then becomes a basis for defining "stateful" sensors, further broadening the original concept of action-based sensors and allowing for one to examine state requirements alongside information requirements.

The approach followed in this paper is, after some broader context and preliminary formalization of plans and planning problems generally (Sections 2 and 3), to reexamine Erdmann's classic theory of action-based sensors through that lens (Sections 4 and 5). Because the generalized treatment we offer (especially with partial observability) helps make the theory more practical, we discuss relationships to problems of sensor design (Section 6) and also LaValle's Sensor Lattice concept (Section 7). Having done that, the paper pivots (Section 8) so that action-based sensors, now in the guise of reactive plans, become a way to improve our understanding the space of plans (Sections 9–10), culminating in an example (Section 11), before concluding (Section 12).

2 Related Work

Our motivation for revisiting 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 (1993) 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 easily we say that a sensor can detect "walls"! These mental categories are ingrained so deeply as to have a pernicious influence on our thinking.

In conceiving his theory, Erdmann (1995) 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.[2] His definition appeared to give the utmost leeway in its requirements, being most relaxed 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.

This task-focused approach to sensor design puts at the forefront of consideration the close link between the robot's desired operation and the sensors needed to accomplish it. Rather than plans being solved for, given a certain robot, a robot is designed to be well-suited to completing its task. Taking such an approach allows for the incorporation of restrictions, such as defining planning problems (and therefore obtaining plans) which exclude certain movement or sensing. Robots can then be designed based on these limitations on how it should behave or what can be sensed.

One approach to designing 'well-suited' robots is to simultaneously optimize controller and body design. Biologically-inspired work, such as Banarse et al. (2019) and Lipson and Pollack (2000), employ iterative evolutionary methods in the search of optimality, in each case with change driven by a pre-defined fitness metric. Pervan and Murphey (2021) take a different approach, beginning with either the sensor or actuator space, and then minimizing the other, also taking into consideration the question of design limitations, in which a designer may have a particular set of sensors or actuators to choose from. Overall, the goals of these works are similar to our own: understanding the impact of robot design on sensing requirements and plan complexity.

Zardini et al. (2021), in scaling-up and extending previous work (Censi 2016), approaches the link between plans and the resulting robot requirements through a co-design problem which integrates controller dynamics into the problem. This work includes the controller and performance bounds within the set of what to optimize. The result is that the system is then designed closely with its intended behavior. Though the co-design problem is not directly the same as our search for sensing and state requirements, identifying these requirements places bounds on what components will satisfy the constraints.

To explore how changes in sensing capabilities affect a given problem, a structure for comparing sensors to each other is valuable. The sensor lattice described by LaValle (2019) is one approach to this, defining a lattice which contains the set of all possible partitions of a given state space. This framework allows for precise comparison of how sensors may be stronger or weaker than each other, or if one sensor can be used to solve a problem instead of another. A longer discussion of his sensor lattice and its relationship to the present work appears in Section 7. We show that his sensor lattice also contains the set of action-based sensors within it, and use it as an inspiration for a lattice with which to compare plans by their executions.

At the extreme end of reducing what information is stored and what is sensed, reactive plans and action-based sensors are particularly valuable for robots with hard restrictions on

their resources. Work by O'Kane and Shell (2017) explores methods of searching for minimal plans, which are valuable for these resource-restricted systems.

Ultimately, the close link between what a robot must do and how it must be designed to do it impacts a broad spectrum of problems within the robotics community. Extending Erdmann's work to a larger family of plans allows for the concept of action-based sensors to be applied where it previously could not be. This provides designers and theorists with another tool for analysis of information requirements, particularly with the theory now being applicable to plans for which the robot must keep state.

3 Preliminaries

In this paper, the environment and robot inhabiting it are described in terms of two symmetric structures: planning problems and plans. The former defines both the world and task, while the latter defines the robot and its operation. Mutual interaction between the world and the robot determines whether the plan solves the planning problem. Operating in discrete time, a robot receives observations and chooses an action to take. This action is then performed upon the world, which has its own structure that decides the action's outcome. This outcome then determines what observations are next received by the robot. Both can be conceived of, speaking intuitively about casualty, as instances of "choice." Thus, there is a back-and-forth between the choice the robot makes (the action selected to be executed), and the choice the world makes (the action's outcome and subsequent observation). A bipartite graph called a *procrustean graph*, or *p-graph*, will be used to formalize these aspects next. Definitions 1–4 are slightly less general versions of those from Saberifar et al. (2019); we refer the reader to that original reference for more comprehensive discussion.

Definition 1. p-graph (Saberifar et al. 2019). *A* procrustean graph (p-graph) $P = (V, V_{\mathrm{init}}, Y, U, E)$ *is a finite edge-labeled bipartite directed graph in which:*

1. *a finite vertex set V can be partitioned into two disjoint subsets, called the* action vertices $\mathrm{V_u}$ *and the* observation vertices $\mathrm{V_y}$*, with* $\mathrm{V} = \mathrm{V_u} \cup \mathrm{V_y}$*,*
2. *a non-empty set of vertices (*$\mathrm{V_{init}} \subseteq \mathrm{V_y}$*) are designated as* initial vertices*,*
3. *each edge $e \in E$ originating at an observation vertex is labeled with a set of observations* $\mathrm{Y(e)} \subseteq \mathrm{Y}$ *and leads to an action vertex,*
4. *each edge $e \in E$ originating at an action vertex is labeled with a set of actions* $\mathrm{U(e)} \subseteq \mathrm{U}$ *and leads to an observation vertex.*

With a p-graph, we can model both planning problems and plans. A planning problem (or, as we write interchangeably, *world*) models goal attainment tasks that require the robot to arrive in some condition in the world.

Definition 2. planning problem. *A* planning problem $W = (V, V_{\mathrm{init}}, Y, U, E, V_{\mathrm{goal}})$ *is a p-graph W equipped with a* goal region $V_{\mathrm{goal}} \subseteq V(G)$.

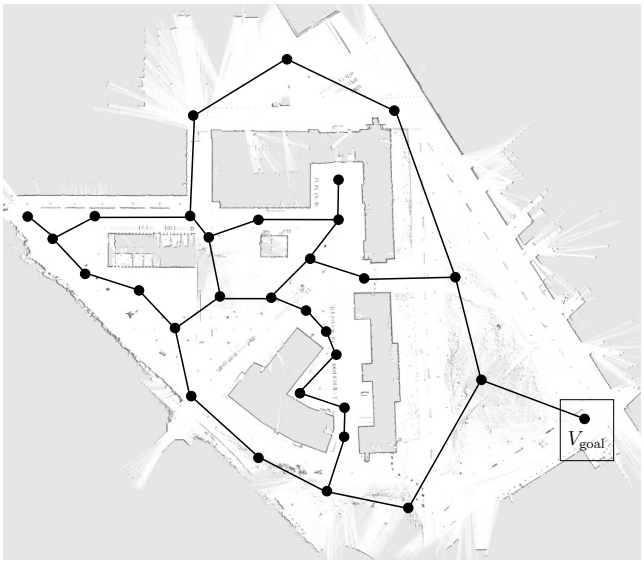


Figure 1. Initial working example: A mobile robot navigates in a planar environment amongst obstacles. For simplicity, a generalized Voronoi graph is superimposed over the top; the robot can start anywhere and we desire that it navigate toward the region represented by the rightmost vertex. (This is part of the University of Freiburg campus, with thanks to Cyrill Stachniss and Giorgio Grisetti for making the dataset available.)

A plan prescribes actions for particular circumstances in order to solve planning problems. As it is a directed graph, it is potentially governed by internal state, encoded directly into its branching structure.[3]

Definition 3. plan. *A* plan $P = (V, V_{\mathrm{init}}, Y, U, E, V_{\mathrm{term}})$ *is a p-graph P equipped with a* termination region $V_{\mathrm{term}} \subseteq V(P)$.

Since both planning problems (i.e., worlds) and plans involve p-graphs, we have used (W) and (P) respectively to help designate to which p-graph some thing belongs, e.g. the vertices $V(W)$, or $V(P)$. Although it was not strictly necessary given the scoping implicit in the definitions above, we will persist with this convention.

The preceding two definitions have exactly identical form: a p-graph and a set of vertices. The p-graph describes dynamics, while the set of vertices describe some notion of termination. A termination semantics (made precise in Definition 4) ties these two objects together, and differs slightly between plan and planning problem. It is expressed in terms of an *execution*, a sequence of alternating observations and actions, and (for this paper) always beginning with an observation, or is the empty sequence. We trace execution $s = y_0 u_1 y_1 u_2 \dots y_n$ over a p-graph Q by beginning at some vertex in $V_{\mathrm{init}}(Q)$, and following an edge labeled with some set containing the observation y_0 in the execution, and proceeding by following an edge labeled with a set containing action u_1, and so on for the whole sequence. If an execution is possible on both plan and world, it is a *joint-execution*. Joint executions describe the dynamic intermeshing of the plan and the world. We desire that either such a sequence must lead to vertices 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.

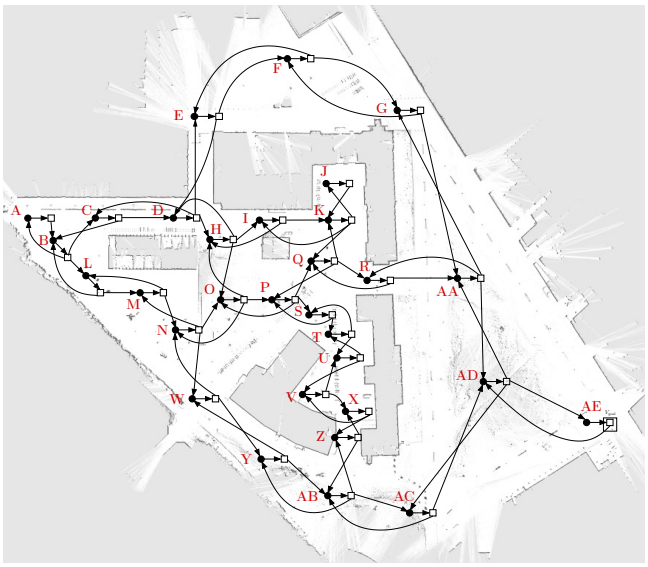


Figure 2. The example world from Figure 1 as a p-graph with goal region. Here, we consider every observation vertex (filled circle) to be a possible initial configuration. Observation vertices give distinct observations as output, while the actions include movement in the cardinal and intercardinal directions. Labels showing these have been omitted for clarity of the figure.

Definition 4. solves. *A plan* (P, V_{term}) solves *the planning problem* (W, V_{goal}) *if:*

1. P *is* finite on W. *The length of all joint-executions must be bounded.*
2. P *is* safe *on* W. *In tracing executions,* P *must never have an action leaving a plan vertex if there is no outgoing edge with that action at the vertex reached in the world. Additionally, the plan must always be ready to receive observations as can arise from possible executions on* W*, starting at* $V_{\text{init}}(W)$*, with actions chosen via* P.
3. P *is* live. *Every joint-execution* $y_0 u_1 \cdots y_k$ *or* $y_0 u_1 \cdots x_k$ *of* P *on* W *either reaches a vertex in* V_{term}*, or is a prefix of some execution that reaches* V_{term} *and, moreover, all the joint-executions reaching a vertex* $v \in V_{\text{term}}(P)$*, when traced on* W *must reach a vertex* $w \in V_{\text{goal}}(W)$.

Occasionally we will be interested in the vertices reached by some given execution. Then one traces the execution following the process outlined above, and those vertices reached by tracing sequence s on Q will be designated $\mathcal{V}_s^Q$. When tracing any such sequence, if, as one proceeds, at most one edge can be traversed, the p-graph is *deterministic*. Otherwise it is *nondeterministic*. The *language* of a p-graph Q (or plan and world), denoted $\mathcal{L}(Q)$, is the set of all executions.

3.1 Example

We give an extended example to make the preceding definitions more concrete. Figure 1 depicts part of the University of Freiburg campus via a map constructed using a robot equipped with a laser rangefinder (Stachniss and Grisetti 2010), and available as part of the Robotics Data Set Repository (Howard and Roy 2003). Shown superimposed is an (approximate) generalized Voronoi graph (Choset and Burdick 1995) that has been constructed to help reduce complexity and aid with discussion of the space. We will assume for the purposes of this example that the Voronoi graph vertices correspond to locations that our hypothetical robot can clearly and reliably distinguish from each other, e.g., using unique range signatures. We will suppose that our robot can start anywhere in the environment and we wish for it to reach the position represented by the rightmost vertex.

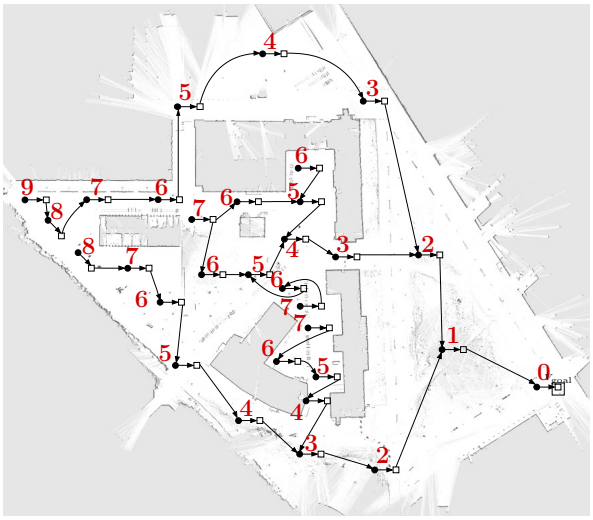
The p-graph depicted in Figure 2 shows this scenario modeled as a planning problem. In this example, each point on the original figure has become an observation vertex (shown as filled circles), which has an associated action vertex (shown as white squares). At each observation vertex, the transition for the robot is determined by what observation is received from the world. Owing to the ability to reliably distinguish locations, here edges departing any circular vertex will bear a label directly corresponding to that vertex. At each action vertex, the robot can select an action that labels any of the outgoing edges. For this example, they are the cardinal and intercardinal directions, although sophisticated motion primitives might also be involved.

Figure 3a and Figure 3b show different kinds of plans for reaching the rightmost vertex (the goal). These plans specify a certain subset of actions available from the original planning problem—each selects actions for the robot to take at different locations within the world. Figure 3a shows a plan derived through backchaining from the goal, while Figure 3b shows a more arbitrary plan. Looking at these plans, the backchained plan's routes are significantly shorter.

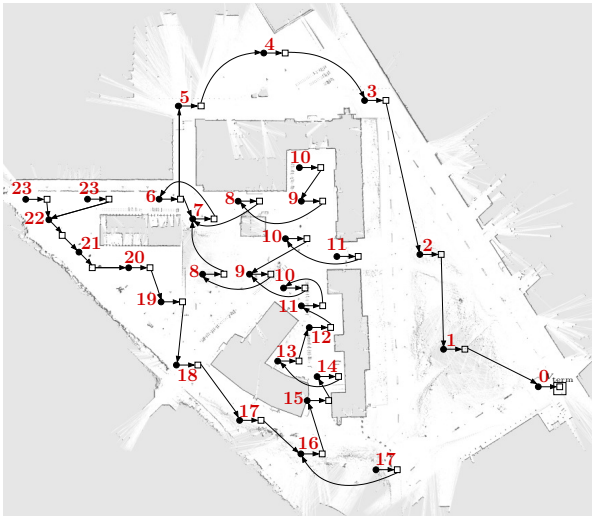
3.2 Generality, Scope, and Problem Variations

Quite apart from cluttering the diagram, the use of separate action and observation vertices in the previous example seems unnecessary. So too, the fact that plan's actions are not simply shown as direct prescriptions on the planning problem itself; perhaps selecting and highlighting a subgraph of Figure 2 could have sufficed to describe Figure 3a? But the preceding definitions are rather more general, and that generality will be put to use. The definitions permit, for instance, the plan to track "beliefs" about potential world vertices that are consistent with the execution sequence (for instance, consider a nondeterministic world p-graph with nondeterminism arising from aliased sensor readings, unreliable action outcomes, or both). Later, important conditions will be examined in detail when the resemblance between plan and planning problem is limited. The example above emphasizes familiarity, but the added generality will come to the fore as we examine more exotic plans which yield additional action-based sensors.

Without further forestalling, we add some detail here by touching particularly on considerations of observability. In Figure 2, the p-graph has observation and action vertices which form a direct correspondence with the original points on the graph in Figure 1. For this example, each observation vertex yields a distinct observation, allowing the robot to localize itself directly throughout its execution via the last observation received. Full observability, as here where observations uniquely map to vertices in the world, simplifies matters in searching for (and expressing) a plan, because



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



(b) A different set of choices that also always reach the goal.

Figure 3. Two plans, as p-graphs with termination regions, that solve the planning problem in Figure 2. Progress measures (Section 4) are labeled in red.

plans need only prescribe actions on the basis of vertices. This is the familiar understanding of full observability in regard to planning problems when considering plans as objects that we (or our algorithms) seek.

As we shall see, action-based sensors partition the world into regions which can be conflated without risk of task failure. Plans can yield action-based sensors and when the underlying planning problem is fully observable, this observability expresses a different aspect, though still in the nature of a simplification. Instead, it places no *a priori* limits on what the robot could in principle distinguish. When an action-based sensor conflates regions of the world as partitions, it represents a degree of acceptable additional partial observability. To apply the theory when there are known limitations owing to features of the environment, or because of other considerations (e.g., technological, practical resource limits, or design constraints), then partial observability can be used to specify particular perceptual limits. Modeling the constraints of existing sensors results in a different interpretation: the existence of an action-based sensor then implies that the existing sensor can enable the robot to complete the task without additional information.

We will further discuss impact of partial observability in defining progress measures and deriving action-based sensors. Here we have emphasized that it is a mistake to believe that because the preceding example is fully observable, the theory which follows does not consider partial observability. In fact, action-based sensors are directly concerned with giving a degree of non-destructive partial observability. Starting in Section 8, a subclass of plans for which no action-based sensors exist, and for which none can be derived, is considered. This non-existence can be directly tied to the requirement of additional information which a sensor cannot provide, and hence to complete the task, plans will require internal state.

4 The Progress Measure

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 the information a robot needs is precisely and solely that which is needed to determine how to act now.

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 (1995) 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 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 5. 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, and 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. In this way, the executions of the plan visit vertices in the world such that the resulting progress measure is strictly decreasing.

Definition 6. 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 $\phi_g(V) \coloneqq \max_{w \in V} \{g(w)\}$ is an execution progress measure.*

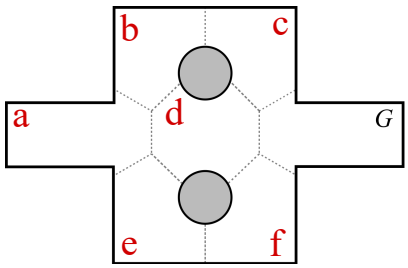
The intuition here is to give a measure on singleton vertices and require that we get an execution progress measure when it is lifted, in a natural way, to sets. When discussing the existence and properties of either kind of progress measure, we will simply use *progress measure*; when necessary, we disambiguate between an *execution progress measure* or *vertex progress measure*, or provide context to resolve any ambiguity.

Progress measures can be seen in the red numerals in Figures 3a and 3b. 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 increasing physical straight line distance) before reaching it. Such an example can be seen in Figure 3b, exemplifying the difference between a progress measure and a metric of physical distance from the goal region.

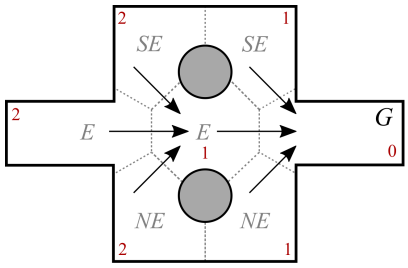
4.1 Progress Measures and Crossovers: Lack of Uniqueness and Existence

For clarity in the following section, we will put aside the planning problem in Figure 2 to consider a minimal example. Figure 4 shows a small, seven-vertex planning problem with a single goal G, as well as three plans which are able to solve it from any starting region in the world. The three plans here give three different progress measures; Erdmann's concentration on backchaining is but one choice.

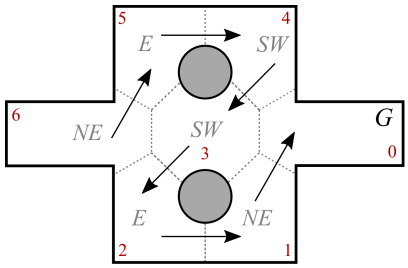
The pair of plans in Figures 4c and 4d are worth contrasting. These two plans have some vertices where they take the same action, and some where they differ. There is little intrinsic reason to prefer one over the other, and we could even have a plan that considers both of the routes such as the plan in Figure 5, which chooses one of the two routes arbitrarily (via nondeterminism) and commits to that route once it has been selected. Even within this small example, the plans will construct contradictory progress measures, and so the plan which combines them has no progress measure at all. This stems from the fact that, though the plan informs our definition of progress, the progress measure is ultimately defined on vertices in the world.



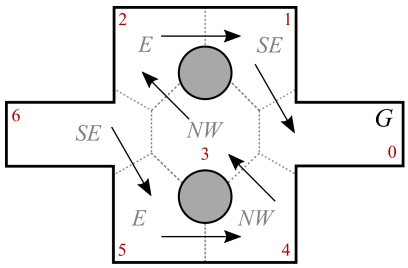
(a) A small world.



(b) A solution derived via backchaining.



(c) A plan which solves the planning problem of reaching the goal, G, from any vertex in the world.



(d) Another solution.

Figure 4. A small world with goal vertex G, along with three plans that solve it. As we will see, this is sufficient to demonstrate the existence and handling of crossover conflicts. For each plan, the red integers are a progress measure.

Figure 5's plan contains executions that go in opposite directions from each other; for example, the 'Z' route (which represents the movement according to Figure 4c) visits the location **b** before **d**; however the 'Σ' motion (akin to Figure 4d) visits **d** before **b**. For the p-graph to represent a solution, the finiteness requirement means it must be structured such that the robot could never actually cycle infinitely. Indeed, it meets this requirement. But the progress measure, considering only the corresponding world vertices, has the impossible task of satisfying $\phi(\{b\}) < \phi(\{d\}) < \phi(\{b\})$.

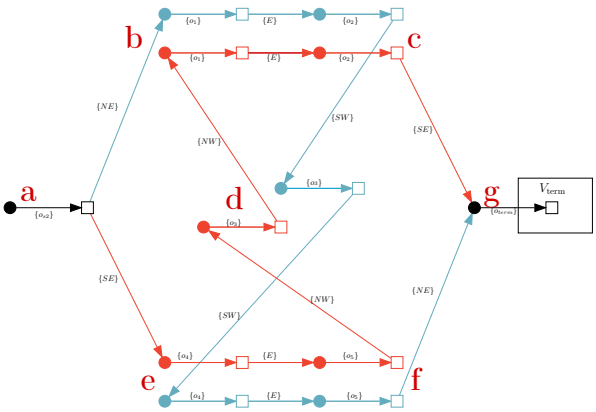


Figure 5. A single plan in p-graph form that solves the problem of reaching the goal in Figure 4a. The letters **a**–**g** here serve to expose the correspondence to locations in the world. It includes both the 'Z' route of Figure 4c and the 'Σ' one of Figure 4d. Every (circular) observation vertex is in V_{init} and, at the start of its execution, the robot receives an observation that can be traced to no more than two possible observation vertices. Once an action is chosen (the choice is resolved arbitrarily), the robot is ether committed to the blue or the red subgraphs. This plan induces numerous crossovers, arising from the lack of a global ordering on when vertices are visited.

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 a *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. Crossovers can be thought of as cyclic dependencies induced on the world by the plan. If a plan is a solution (like Figure 5) then the set of joint executions (or, equivalently, the intersection of its language with that of the world) must be finite. To re-visit a vertex in the world, such a plan must can make use of multiple vertices to distinguish executions. But, the progress measure's definition considers all potential paths to and from a world vertex. Therefore, executions that visit vertices in differing orders create a cyclic dependency.

Definition 7. crossover conflict. *A plan P that solves a planning problem W has a* crossover conflict *if there are two distinct joint-executions s_1, s_2 such that:*

1. *s_1 and s_2 both visit the world vertices v and v', and*
2. *s_1 requires an execution progress measure where $\phi(\{v\}) > \phi(\{v'\})$, while s_2 requires an execution progress measure where $\phi(\{v'\}) > \phi(\{v\})$.*

Crossovers are the primary cause of failure in a plan that does not produce 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 $\phi(\mathcal{V}_p^W) \leq \phi(\mathcal{V}_q^W)$.

For a progress measure it is enough to conceive of an ordering on the vertices of the world. At least one goal must come last in the ordering. For executions, the requirement of prefixes having a higher measure than sequences which they precede imposes an ordering on those vertices, as well.

To attempt to fix the progress measure, we first assign a goal (the final element in the ordering) value 0 and increment up as one goes 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 vertices reached by p and q such that they no longer fulfill the condition of (c), then a progress measure does in fact exist. Otherwise, however, because we were assigning values to the ordering of vertices based on back-tracking from the goal, we must have seen the vertex reached by p in the ordering before the vertex reached by q and assigned it a value accordingly. If p is a prefix, that means the vertex reached by p is visited by an execution both before *and* after it visits the vertex 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.

The presence of crossovers is the necessary and sufficient condition for the non-existence of a progress measure, and their existence impedes our ability to craft action-based sensors. To identify crossovers, the relationship between plan vertices, plan executions, and world vertices must be made explicit, ideally without directly enumerating the plan's language. Next, we introduce an algorithm, based on a graph product construction, that is useful in this regard.

5 Removing Crossovers and Enumerating Progress Measures

Given a plan that solves a planning problem 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 8. operative actions. *For a plan P that solves planning problem W, an action u_k is an* operative action *if there exists 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 $e = y_1 u_1 y_2 \ldots u_{k-1} y_k$ for which:*

- *e arrives at $w \in V(W)$ when traced on W;*
- *action u_k is a label on an outgoing edge from w;*
- *e arrives at $p \in V(P)$ when traced on P;*
- *action u_k is a label on an outgoing edge from p.*

For a given world, intuitively, the operative action set at any vertex in that world consists of only those actions that the robot (using its plan) may actually end up taking during its execution. When a plan doesn't induce any progress measures, we will consider alternate plans with the property that they select the same actions in the same places in the world as that original plan. Operative actions formalize this:

Definition 9. derived plan. *Given world W, 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 $v \in V(W)$, $\mathfrak{u}_{P'}(v) \subseteq \mathfrak{u}_P(v)$.*

Next, we generate all possible plans without crossovers from some other plan, but using only actions from the operative action set. The algorithm we describe next, named CLIP, has two parts: first, given a plan and a planning problem, builds an intermediate data-structure (a graph) through which all of the crossovers can be identified. Secondly, it enumerates resolutions to these crossovers. For edges that are not involved in any crossover, we may take any subset of them so long as we ensure it is a solution. Thus, as an object on which various combinatorial operations act, this graph implicitly represents a large number of plans.

5.1 The Plan-World Interaction Graph

CLIP starts by constructing a type of product graph we call the *Plan-World Interaction Graph*, or simply the *I-Graph*. It is formed by corresponding plan and world vertices, while also encoding information about the operative action set in an organized fashion. In the original plan, different executions, possibly separated and obscured by the plan's structure, may interact so as to obstruct the existence of progress measures. These various instances are collapsed within the I-Graph, which records when multiple plan vertices map to the same world vertex, transforming crossovers into explicit cycles.

The I-Graph is itself a p-graph, but its structure is defined by the joint-executions of the plan and world. Algorithm 1 provides pseudo code for the incremental construction of the I-Graph. To know which plan vertices correspond to certain world vertices at different points in execution, the I-Graph starts as a single vertex that corresponds to the initial

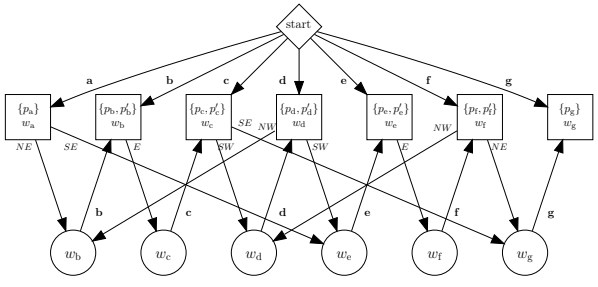


Figure 6. An I-Graph for the plan seen previously in Figure 5. Plans may have multiple vertices that correspond to the same vertex in the world. The I-Graph merges these into a single vertex.

Algorithm 1: Constructing the Plan-World I-Graph

Data: World $W = (V_W, V_{\text{init}W}, Y_W, U_W, E_W, V_{\text{goal}})$,
Plan $P = (V_P, V_{\text{init}P}, Y_P, U_P, E_P, V_{\text{term}})$
Result: I-Graph $I = (V_I, V_{\text{init}I}, Y_I, U_I, E_I, V_{\text{term}})$

Procedure I-GRAPH CONSTRUCTION(W, P) **:**
  igraph ← New PGraph()
  igraph_vtx ← New Observation_Vertex(I-Graph)
  world_vtxs, plan_vtxs ← $V_{\text{init}W}$, $V_{\text{init}P}$
  I-GRAPH SEARCH(world_vtxs, plan_vtxs, igraph, igraph_vtx)
Function I-GRAPH SEARCH(world_vtxs, plan_vtxs, igraph, igraph_vtx) **:**
  if no outgoing edges for world_vtxs, plan_vtxs **then**
    return I-Graph
  if vertices are type "observation" **then**
    world_tgts = New Set()
    plan_tgts = New Set()
    foreach world vertex v_W **do**
      foreach outgoing edge from v_W **do**
        observation ← outgoing edge label y
        add edge.tgt to world_tgts
        foreach out edge from plan_vtxs **do**
          if plan edge has label y **then**
            add plan edge.tgt to plan_tgts
  else
    world_tgts = New Set()
    plan_tgts = New Set()
    foreach plan vertex v_P **do**
      foreach outgoing edge from v_P **do**
        action ← outgoing edge label u
        add edge.tgt to plan_tgts
        foreach out edge from world_vtxs **do**
          if world edge has label u **then**
            add world edge.tgt to world_tgts
  if world_tgts corresponds to existing vertex $v_I \in V_I$ **then**
    if world_tgts $\subseteq V_{\text{goal}}(W)$, plan_tgts $\subseteq V_{\text{term}}(P)$ **then**
      add v_I to $V_{\text{term}}(I)$
    if vertices are type "observation" **then**
      label_set ← outgoing obv. from $world\_vtxs$
    else
      label_set ← outgoing act. from $plan\_vtxs$
    for each label n in label_set **do**
      backlabel ← label
      add edge (igraph_vtx → v_I, with label) to igraph
      I-GRAPH SEARCH(world_tgts, plan_tgts, igraph, v_I)
  else
    if vertices are type "observation" **then**
      label_set ← outgoing obv. from world_vtxs
      new_vtx ← New Observation_Vertex(IGraph)
    else
      label_set ← outgoing act. from plan_vtxs
      new_vtx ← New Action_Vertex(IGraph)
    if world_tgts $\subseteq V_{\text{goal}}(W)$, plan_tgts $\subseteq V_{\text{term}}(P)$ **then**
      add new_vtx to $V_{\text{term}}(Igraph)$
    for each label n in label_set **do**
      backlabel ← y/u
      add edge (igraph_vtx → new_vtx, with label y/u) to igraph
      I-GRAPH SEARCH(world_tgts, plan_tgts, igraph, new_vtx)
  returnI-Graph

vertices of both the plan and the world. Each vertex in the graph will have a pair (v_P, v_W) — for this initial vertex, $(V_{\text{init}}(P), V_{\text{init}}(W))$ — indicating what vertices within the plan and world it corresponds to. This single observation forms the initiating layer of the I-Graph, modeling the moment execution begins and the robot has yet to receive any observations.

The rest of the graph is constructed by tracing the possible executions of the plan, alternatingly giving priority to the plan and world when determining structure. For each observation vertex, the current set of world vertices determines the outgoing edges. The plan is safe on the world, and therefore for each observation that might occur from any of these outgoing edges, the plan has at least one corresponding edge labeled with the same observation from each vertex within the set under consideration. For each of these edges, an action vertex is added to the I-Graph, consistent with the world and plan vertices reached by that observation.

In the examples discussed above, a single observation completely localized the robot to one vertex in the world. However, one observation need not isolate the robot to a single vertex within the plan as there may be multiple ways (e.g., the blue and red routes in Figure 5) from which our robot may have chosen arbitrarily. For the plan in Figure 5, the initial observation is (for most cases) consistent with up to two possible plan vertices.

As per the earlier discussion of partial observability, this can also be true within the world itself: information in an observation may correspond to multiple places within the world. In such cases, v_W is a set of vertices, just as v_P is in the example shown here.

For each action vertex in the I-Graph, unless it is a terminating vertex, each corresponding plan vertex in v_P has outgoing edges with actions the robot can take. (It is also possible that vertices in V_{term} have their own outgoing actions, indicating the *option* to terminate.) These actions are added as outgoing edges, leading to a new set of observation vertices for the plan and world. If a vertex within v_P does not have that action as an outgoing edge from itself, it is dropped from the current plan sets, as our robot has committed to a certain part of the plan.

Again at an observation vertex, the process repeats until we have explored all possible executions of the plan. Should the execution return the robot to a set of world vertices already encountered, the edge connects back to the existing vertex. If new plan vertices are associated with this set of world vertices, they are appended to the existing list of associated vertices.

The result is a graph with three layers: the initiating layer, the plan layer, and the world layer. The I-Graph for the running example is shown in Figure 6. The initiating layer is our single starting vertex that corresponds to all possible starting locations. (Although illustrated with a diamond, this is simply to highlight that this observation vertex is differs from the others.) The plan layer consists of all action vertices, and derives its name from the fact that actions are the part of execution dictated by the plan. Its departing edges go to the world layer. Although there may be multiple observations a robot may receive, the various edges correspond to the world's choice of observation to provide. The edges from this layer return to the plan layer.

The I-Graph mimics the structure of the world and its connections, but is restricted by the operative action set of P.

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. Beginning the construction, the algorithm starts with the set of initial world vertices $V_{\text{init}}(W)$ and the set of initial plan vertices $V_{\text{init}}(P)$. The set of all outgoing observations from the vertices in $V_{\text{init}}(W)$ represent observations the robot can receive initially. As the plan is presumed to be safe on the world, each vertex in $V_{\text{init}}(P)$ has an outgoing edge labeled with the possible observations.

Therefore, for each possible observation y, the algorithm obtains the result of tracing a joint-execution consisting of that single observation as a new set of world and plan vertices: $(\mathcal{V}_y^P, \mathcal{V}_y^W)$.

For u_k to be an operative action at vertex $v_w \in V(W)$, there must exist a joint execution e arriving at v_w and also $v_p \in V(P)$, where both v_w and v_p have an outgoing edge labeled with u_k. The two sets of vertices just reached by the algorithm, $(\mathcal{V}_y^P, \mathcal{V}_y^W)$, comprise action vertices. As shown in lines 19–28 in Algorithm 1, the outgoing edges for each action vertex in $\mathcal{V}_y^P$ are considered, and their targets collected into a set. As the plan is presumed to be safe on the world, an outgoing action from a vertex $v \in \mathcal{V}_y^P$ indicates that each vertex in $\mathcal{V}_y^W$ has an outgoing edge with the same label. These edges indicate the operative action set for each world vertex within our current set under the joint-execution y, by definition.

Now, for each outgoing action u, following the edges labeled with u leads to the set of vertices obtained by tracing yu on the plan and world: $(\mathcal{V}_{yu}^P, \mathcal{V}_{yu}^W)$. As y is an element in the larger set $W(Y)$, and u is an element in the set $P(U)$, each set $(\mathcal{V}_{yu}^P, \mathcal{V}_{yu}^W)$ is a subset of all vertices which can be reached by joint-executions of length two. As the algorithm has considered all possible observations that could be obtained from the world, as well as all possible actions the plan could take, it has visited all vertices resulting from any joint-execution of length two.

For each y, u pair, the algorithm passes $(\mathcal{V}_{yu}^P, \mathcal{V}_{yu}^W)$ to a recursive call of itself, upon which the same process is repeated. As the incoming actions do not impact which outgoing edges may be obtained, $(\mathcal{V}_{yu}^P, \mathcal{V}_{yu}^W)$ can be treated within the iteration in the same way as the initial set, without regard to the prior actions or observations received.

The algorithm continues until the joint-executions end, which they must as they have bounded length, and further they end in termination (i.e., in $V_{\text{term}}(P)$). Thus, the algorithm obtains the operative action set for all sets of world vertices visited. As each vertex within the constructed I-Graph corresponds to a pair of sets $(\mathcal{V}_e^P, \mathcal{V}_e^W)$, and all actions found by the algorithm are appended as outgoing edges from the vertex associated with that pair, the I-Graph captures the full operative action set of the original plan P.

□

A subset of the I-Graph vertices are also designated as terminating vertices, just as for plans. During construction,

if the set of plan vertices for the current execution are all within $V_{\text{term}}(P)$, and the corresponding world vertices are all within $V_{\text{goal}}(W)$, then the vertex is included in $V_{\text{term}}(I)$. Termination is not required for *every* set of plan vertices that reaches this vertex, as a robot may pass through goal vertices before actually stopping. However, it must guarantee goal achievement when it eventually does terminate.

This not only handles cases in which the robot is unsure if it has reached the goal or not, but also permits cases in which the robot may bypass one goal to reach another. In such a case, the robot's plan has two trajectories: one which goes to the goal vertex and terminates, and another that passes through that vertex to reach another goal further along. In the I-Graph, both will be terminating vertices, but the first will have outgoing edges, providing the robot the ability to terminate the execution *or* continue.

Though the I-Graph is a p-graph augmented with a set of terminating vertices, and is constructed from a plan, it is not necessarily a plan itself. It satisfies the same safety requirements as a plan, but crossovers from the source plan manifest as cycles, which may mean that the set of joint-executions with the world is not finite. Resolution of all these cycles within the I-Graph will result in a plan that is derived from the original plan P and which both solves the planning problem W and has a progress measure. (Later we will show that the resolution of the crossovers demands goal vertices be reached, retaining the requirement that our plans have no dead ends.)

5.2 The Comes-Before Relation

The essence of the proof of Theorem 1, in dealing with progress measures, only used an ordering property and then constructed a function from that ordering. (The definitions for progress measures are written in terms of functions to closely retain Erdmann's original form.) Next, we describe a relation put on the action vertices of the I-Graph, which helps determine whether those vertices can be ordered. We focus on the I-Graph's action vertices because actions are the sole means by which the robot exercises control and hence has the power to split apart crossovers.

We call this relation the *comes-before relation*, denoted $K \subseteq V_u(I) \times V_u(I)$. For action vertices v_1 and v_2 within the I-Graph, we say v_1 comes-before v_2 (written $\langle v_1, v_2 \rangle \in K$) if, after an action taken from v_1, the observation vertex reached by that action has v_2 as a target. Also, that K is transitive: $\langle v_1, v_2 \rangle \in K$ and $\langle v_2, v_3 \rangle \in K$ means $\langle v_1, v_3 \rangle \in K$.

The I-Graph itself is used in initial construction; once the transitive closure of K is computed, this gives, for each action vertex of all vertices that it precedes. There two points of note: firstly, the nature of the I-Graph means that its vertex set is constantly changing during its construction. Additionally, the I-Graph's p-graph form means that it is similar to the plan and world from which it derives; as mentioned previously, this means that the resolution of crossovers results in a new plan that is already expressed using the same structure as the original plan.

However, once the vertices and edges of the I-Graph are fully established, experience has shown that an adjacency matrix representation provides simple, compact way of representing the relation between vertices. It also affords computationally attractive optimization (e.g., obtaining the transitive closure via linear algebraic methods).

Also, for the purposes of resolving crossovers, we consider each vertex within the I-Graph, and not the sets of vertices they correspond to in the world. As a result, a vertex v_1 in the I-Graph 'comes before' another vertex v_2 if there is a path in the I-Graph between two vertices which correspond to v_1 and v_2, respectively.

When dealing with fully observable planning problems, this matrix has a one-to-one correspondence to world vertices, and resolving the crossovers present in the I-Graph yields a vertex progress measure. In some instances of partially observable problems, it is possible for the same vertex in the world to correspond to multiple vertices in the I-Graph. In these cases, resolution of the crossovers in the I-Graph may yield only an execution progress measure on sets of vertices, as opposed to a vertex progress measure on individual vertices in the world. The impact of this is discussed starting in Section 8, with important implications following.

5.3 Resolving Crossovers

When the I-Graph collapses down all executions, conflicting orderings on world vertices that only existed conceptually now present as cycles within the I-Graph itself. Resolving crossovers now becomes a matter of "clipping" away some of the edges that constitute these cycles, which is the function of the second part of CLIP.

Each cycle within the I-Graph yields a set of edges, which can be further reduced. For each set, we consider a subset called the *candidate edges*. Edges are disqualified from being a candidate for removal if the algorithm can determine in advance that their removal will result in a non-solution.

If the planning problem is both fully observable and requires that the robot is capable of starting from anywhere, then edges are excluded from candidacy if their removal would leave a non-goal vertex with no outgoing edges, which would leave the robot stranded at the corresponding world vertex.

In all other problem cases, removing edges and stranding certain vertices in the I-Graph may only result in a change in executions. Therefore, the only edges which can be immediately disqualified are those that are the sole outgoing edge from an action vertex at the start of execution, as the robot must have choice in action from all starting locations. The algorithm also removes cycles that are fully contained within larger cycles in the I-Graph, so that the same sets of vertices are not considered multiple times.

In our implementation, since even moderate sized problems can give a search space that will be very large, the user also has the option of manually resolving some cycles before the main search. As the user often has additional insight into the problem structure (for instance, understanding how choices taken in one cycle may impact desired choices in future cycles), this knowledge can be used to help the algorithm search a reduced and more relevant space.

CLIP constructs a search tree starting from the original I-Graph, using the sets of candidate edges for each crossover. The search tree considers the powerset of the candidate edge set for removal. For each set of edges from the powerset, we

add to a list of proposed edges to remove from the original I-Graph. We include the empty set within this set, as it is possible for cycles within the graph to have edges or vertices in common, and therefore the choice of a cycle to remove no edges effectively defers the choice of which edges to remove to later.

In addition to limiting the set of edges considered, CLIP also prunes branches of the search tree once it is known that they will have no solutions. As CLIP only removes edges, if the I-Graph's initial vertex cannot reach the goal, this is an irreversible disconnection and it will continue to fail to be a plan given any additional removals. Therefore, the validity of each currently proposed set of removed edges is checked to see if the current node in the tree still permits the initial vertex within the I-Graph to reach the goal before CLIP generates its children.

5.4 The Output of CLIP

We call the output plans of CLIP the *representatives*, or individually a representative or *representative plan*. CLIP produces numerous subgraphs of the I-Graph, each of which has a method of resolving all crossovers that is different from the other outputs. Each representative plan in the output set is both a plan derived from the I-Graph by CLIP, and has an operative action set that is a subset of that of the I-Graph used to generate it.

These output plans are representatives of the full set of all solutions in the sense that they capture all possible ways of resolving the crossovers. Any plan in the full set of solutions that is not directly generated by CLIP can be derived from a representative.

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 the operative action set of P, or a subset.

2. **All plans generated by CLIP have a progress measure.**

   By the comes-before relation, CLIP verifies that no cycles in the world exist before acceptance. As the lack of cycles is indicative of a lack of crossovers, by Theorem 1 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 vertex, that vertex does not 'come before' at least one goal vertex, CLIP rejects it. CLIP also rejects any plans with cycles still present. Therefore, any plan CLIP accepts is a solution.

□

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. Unless removal of an edge results in the breaking of all paths from an initial vertex to the goal, edge removal will not result in a plan no longer being a solution. In addition, as removing action edges cannot induce a cycle 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 cycles 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 in 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 cycle that causes its language with the world to not to be finite. If it is smaller than any representative, then it implies CLIP does not enumerate all possible values of the cycle edges. As CLIP enumerates all cycle edge possibilities (through generating the powerset), it must enumerate all possible values of the cycle edges, so this is a contradiction.

Therefore, P' must be derived from some representative in order to be a valid plan; any derivation other than from the representative set violates our requirements for plans to solve. □

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. As the executions of the representatives are derived from the

original plan, they may contain executions that are not included within the original language of the input, but are a subset of its set of operative actions. Having resolved the issue of plans without vertex progress measures, we next turn to the question of how to use progress measures to obtain sensors.

6 Translating Progress Measures into Sensors

Now that we have obtained a plan with a progress measure, we can use it to link actions with how to make progress. This is achieved through the notion of a progress cone. Every action $u \in U$ has an associated progress cone, which is a set of observations. At any vertex in the world labeled with an observation in this set, the action u makes progress toward the goal (transitioning from a higher-valued vertex to a lower-valued vertex) according to the progress measure.

Definition 10. 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 vertices with an outgoing edge 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 observations; the second asks which actions make progress given a certain 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 11. 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 $(y, u) \in \mathbf{C}$.*

Given the set of observations $Y(W)$ for a planning problem, any observation y that is used as an edge label of a vertex in $V(W)$ must have at least one action u where $y \underset{\mathbf{C}}{\sim} u$. This forms a covering over $Y(W)$.

For an observation, there are potentially many progress-making actions. However, only one action is needed for any given y to guarantee that the robot 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 12. 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 may not be immediately apparent. A "traditional" sensor can be represented as a function $s : V(W) \to Y(W)$, taking world vertices 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. Each element in Y maps to a set of vertices in the world, and so we can think of the elements of Y as a stand-in for the identifiable regions of W. $W \to Y \to Y'$ is therefore equivalent to $W \to Y'$, and takes in vertices in the world and maps them to this new set.

The process above transforms 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 13. permissive action-based sensor. *A function $f : Y(W) \to 2^{U(W)} \setminus \{\varnothing\}$ 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. Some recent work has examined covers as models of sensors that have imperfections, such as noise or cross-talk (Zhang and Shell 2020, 2021). It is curious that permissive action-based sensors appear to be more powerful owing to the choices inherent in the cover, not less powerful.

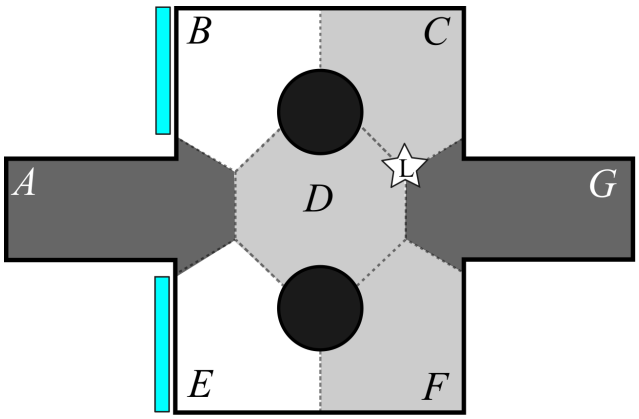
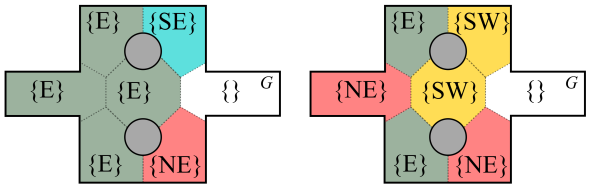


Figure 7. A world in which conflation between vertices precludes Erdmann's sensors being functional. Vertices **C** and **D** both are in mid-range lighting and can see the landmark L, making them indistinguishable. For such a sensing setup, the partition required by the backchained plan cannot be realized.



(a) Partition consistent with the backchained plan in Figure 4b, and its resulting progress measure.

(b) Partition consistent with the plan in Figure 4c and its resulting progress measure.

Figure 8. Two partitions consistent with a progress measure, implementing an action-based sensor. Figure 8a, derived from the backchained plan in Figure 4b, cannot be used to solve the planning problem in Figure 7, while 8b, derived from the plan in Figure 4c, can.

6.1 A Concrete Example

We illustrate some of preceding definitions by adding to the example of Figure 4a. Figure 7 shows our example environment in a bit more detail. Windows (in aquamarine blue) allow light to enter and result in varying light levels throughout the space (in shades of gray). A landmark (L) can be seen from vertices C, D, and G, but not elsewhere.

Figure 8 shows two singleton action-based sensors, each of which maps the observable regions of the world to single actions. Each of the two partitions is consistent with the progress measure of the plan from which they were derived. The action-based sensor given in Figure 8a is derived from backchaining, and is an action-based sensor that would be found with Erdmann's existing methods of obtaining them. The second action-based sensor given in Figure 8b is derived from the plan in Figure 4c. This plan is only considered when we open up the concept of progress measures and action-based sensors to a wider set, as done in this work.

Now, return to the detail in Figure 7 where, due to constraints on what sensors are available, a robot can only recognize brightness levels and the presence of a nearby landmark. Such a robot cannot distinguish vertices C and D. This presents a problem for the action-based sensor obtained from a backchained plan, which requires these vertices be distinguished. The action-based sensor realized in Figure 8b, however, takes the same actions in vertices C and D, and therefore describes a partition of the world that can be realized with this setup. The action-based sensors defined by the extended family of plans we consider allow one to better respect the constraints that designers may actually face.

7 The Sensor Lattice

A single plan may result in multiple action-based sensors, each of which may have multiple partitions that are consistent with it. These partitions draw parallels to the virtual sensors discussed in LaValle (2019). In his model, sensors are defined as a function from the world to sensor readings; for a given sensor value, its preimage is a set of locations in the world that map to that sensor reading. Each sensor then yields a partition over the entire world, based on which locations give which sensor readings.

The sensors-as-partitions are then arranged in a lattice structure based on the fineness of their partitions, with coarser partitions towards the bottom and the partition that can uniquely identify every location in the world as the supremum. The previous example, in Section 6.1, showed two partitions (Figs. 8a, 8b) from two plans. The partitions obtained from their action-based sensors are naturally included within the full possible sensor lattice, which contains all such possible partitions of the state space.

LaValle (2019) addresses the question of state requirements indirectly. Environments that change over time are modeled through adding time as an additional component in their state space. The sensor readings are then functions of both space and time. In the case of trajectory tracking, the history of states and times in which they were visited becomes the state space upon which the sensor acts.

His treatment of time as a separate variable is a substantial difference from any of the ideas we have considered here. Still, the concept of modifying the state space to accommodate how different observations are received over the course of the execution is reminiscent of I-Graphs with execution progress measures, where action depends on the current execution. However, implementing execution progress measures requires more information than just the history of states in the world that the robot has visited. Rather than the state space reflecting only external changes to the world, the sensor must also change based on what actions have been taken by the robot. The requirement of task achievement means that the sensor must be a function both of the world and of the robot's internal state. Additionally, unlike trajectory tracking, the robot does not require its full history to provide the required information, but a particular subset of it.

To convert these execution progress measures into sensors, Definition 12 is insufficient. We must include the history of relevant actions as a sort of *context* that informs what actions are given by the action-based sensor.

Definition 14. contextual sensor. *A function* $g : (Y(W) \times F) \to (U(W) \times F)$ *is a* contextual sensor *if, for all* $y \in Y$*:*

1. *for a given* context f, $g(y \times f) \underset{\mathbf{C}}{\simeq} u$, *or*
2. $g(y \times f) \underset{\mathbf{C}}{\simeq} \{\varnothing\}$,

and *the resulting execution language is safe on the planning problem* W *from which the relation* $\mathbf{C}$ *is derived.*

(We use f for a certain context as opposed to c for two reasons: first, $\mathbf{C}$ is already used to indicate the progress cone relation; second, we will later see that each of these contexts relates to a kind of graph we call a *flower graph*.)

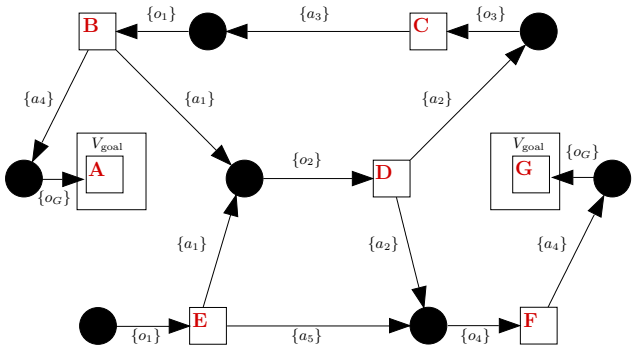
As the context varies over time, it may be that the function g is not defined for all possible pairs of (y, f), but only those y which the robot may encounter under the given context f. The result is multiple partially-defined action-based sensors. The resulting sensor requirements for the group of sensors are the intersection of each action-based sensor's individual requirements.

While the intersection of these partitions exists within LaValle's lattice, the lattice has no way to capture this notion of state. The following sections discuss precisely how execution progress measures lead to the need for context, as well as a method with which to express these stateful sensors.

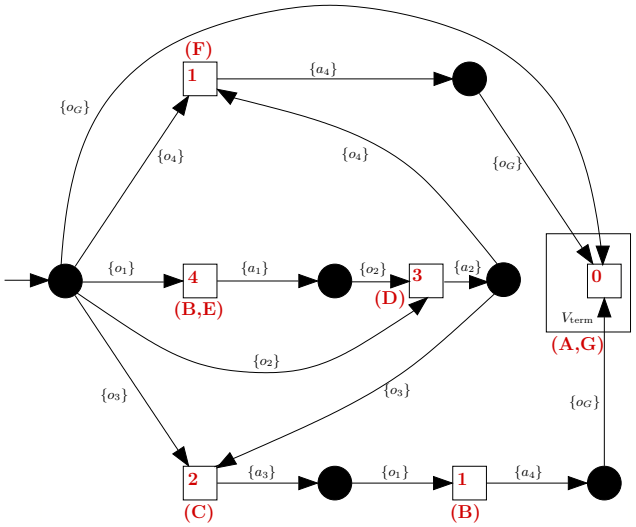
8 Execution Progress Measures, and What They Mean for Sensors

A new example of a planning problem and plan can be seen in Figure 9. Although a robot within this small world is capable of reaching the goal, even from an unknown initial state, it must first take actions that are safe in multiple locations to disambiguate its position in the world. Although we can create an execution progress measure on the sets of world vertices (shown in red), no vertex progress measure exists. Therefore, we cannot define an action-based sensor for this plan either, as there is no mapping between observations in the world and actions to take.

In this example there exists no one-to-one mapping between the plan and world vertices. Such mappings are useful conceptually.



(a) An (approximately) hourglass-shaped world. If we can start at any observation vertex, then a starting observation of o_1 could indicate that the robot is either in B or E.



(b) A plan which solves the planning problem. In the case the o_1 is the first observation received, the robot must take actions to disambiguate its state before it can reach a goal.

Figure 9. An example planning problem, and a plan which is non-homomorphic.

Definition 15. homomorphic solution (Saberifar et al. 2019). *For a plan P that solves planning problem W, consider the relation $R \subseteq V(P) \times V(W)$, in which $(v, w) \in R$ if and only if there exists a joint-execution on P and W that can end at v in P and in w in W. A plan for which this relation is a function is called a* homomorphic solution.

We consider all plans for which this relation is not a function to be *non-homomorphic*. These plans present unique challenges when designing action-based sensors, and we find that for many of them no action-based sensor may exist at all.

We see in Figure 9 that though the robot may revisit the same world state, it occurs under different contexts, which specify different actions. Therefore, we can define a contextual sensor, but not an action-based sensor.

Although the idea that some robots require some kind of memory or internal state to function is not new, we refer the reader once again to the plans in Figure 3. These plans can be converted into action-based sensors, but in their original form are homomorphic, taking after the world's structure. Therefore, while certain structure within plans encodes information necessary to task completion, other structure has no impact on the robot's ability to complete

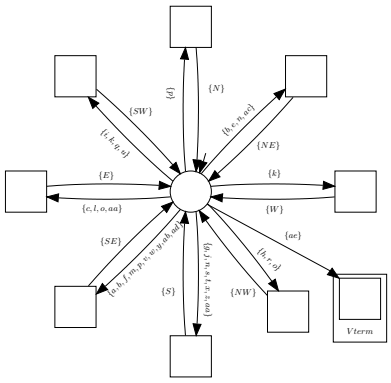


Figure 10. A permissive action-based sensor which solves the planning problem given in Figure 2, presented in flower graph form.

the task. (We return to examine non-homomorphism in more detail in Section 10.2.)

By changing our perspective on action-based sensors, we can begin to bring these plans into our existing framework. Rather than treating action-based sensors purely as a method with which to define sensors, we focus on their ability to act as reactive plans. An example of an action-based sensor presented as a reactive plan can be seen in Figure 10. This representation is the aforementioned flower graph, so called due to its appearance.

While the contextual sensors are not *quite* action-based sensors, but have reactive components that must be connected through transitions. These transitions permit switching between virtual sensors, creating a kind of "stateful sensor". We call these structured plans *vine graphs*, as they consist of small flower graphs connected to each other by chains of observations and actions.

Before exploring vines in more detail, we must first make a detour to explore how the diversity of plans affects both the diversity of sensors and the requirements of stored state. This requires a systematic way to compare plans. With that in place, we can then identify the structure within plans that indicates when the robot switches from one flower to another — the previously mentioned context — as well as how we can use the concept of action-based sensors as a base with which to construct these plans that incorporate state.

9 Expanding the Languages: The Plan Lattice

Depending on the structure of the world, there may potentially be an infinite number of plans. In order to talk precisely about the role of structure and what is meant by our use of the word 'state', we must first develop a way in which we can talk about plans in a comparative sense. The *plan lattice* is a structure that, for a given planning problem, affords comparsion of plans to one another.

Though form influences function of plans, we discard their graph structure and examine the language of each plan's joint-executions. If it is possible to generate the same language of joint-executions with different plan structures (for example, a plan and its action-based sensor), these plans should be treated as equivalent.

However, the list of joint-executions alone is insufficient to compare plans with each other. Although they may have the same language, a plan that reaches a world state and terminates has different behavior from another that reaches that world state and does not. We complement the language with an additional symbol, $\dashv$, to indicate that a given execution ends with the plan terminating.

Definition 16. cessation language. *The* cessation language *of a plan P supplements $\mathcal{L}(P)$ by duplicating executions arriving at $V_{\text{term}}(P)$, but marking those as such by appending a '$\dashv$' symbol. Formally,*

$$\mathcal{L}_{\dashv}(P) = \mathcal{L}(P) \cup \left\{ s\dashv \mid s \in \mathcal{L}(P), \mathcal{V}_s^P \subseteq V_{\text{term}} \right\}.$$

Recall that the symbol $\mathcal{V}_s^P$ denotes the non-empty set of vertices that one arrives at after tracing an execution s on P. This allows for us to determine if a plan contains the full language of another, including all of the executions for which it terminates. We call this the *plan subsumption relation*.

Definition 17. plan subsumption relation. *For two plans P_1 and P_2 we say that P_1* is subsumed by *P_2, denoted $P_1 \preceq_{\dashv} P_2$, if $\mathcal{L}_{\dashv}(P_1) \subseteq \mathcal{L}_{\dashv}(P_2)$.*

9.1 The Plan Lattice

A plan lattice will be constructed modulo a particular planning problem, but since we generally keep a fixed planning problem (W) in mind, we won't hesitate to call it *the* plan lattice. The language of a plan considered as a standalone graph may differ from the language that results from interaction on W. Thus, we rationalize the set of all possible plan languages with respect to a given planning problem W, reducing each language to its joint-executions.

This resulting subset of languages should consist of those which are safe on W. While they must be valid *plans* for the planning problem they are not required to be solutions as defined in Definition 4, and as a result the subset includes languages with executions of infinite length and those that terminate outside of V_{goal}.

Given that the termination symbol $\dashv$ is not included in W's language, we extend W's language for the sake of definition, permitting any string in W to end with the termination symbol $\dashv$.

$$\mathcal{L}_{\dashv}(P) \cap (\{x\dashv | x \in \mathcal{L}(W)\} \cup \mathcal{L}(W)).$$

The lattice is then constructed based on plan subsumption.

The $\subseteq$ relation on languages implies reflexivity and antisymmetry, as languages contain their full set as a subset, and two languages cannot subsume each other unless they have the same language. The join of any two languages $\mathcal{L}_{\dashv}(P), \mathcal{L}_{\dashv}(P')$ is their union, $\mathcal{L}_{\dashv}(P) \cup \mathcal{L}_{\dashv}(P')$. The requirement of transitivity follows, as all $P \preceq_{\dashv} P' \preceq_{\dashv} P''$ implies that $\mathcal{L}_{\dashv}(P) \subseteq \mathcal{L}_{\dashv}(P') \subseteq \mathcal{L}_{\dashv}(P'')$. The meet of any two languages is their intersection, $\mathcal{L}_{\dashv}(P) \cap \mathcal{L}_{\dashv}(P')$.

Figure 11 shows an example plan lattice.

At the bottom of the lattice is the plan that does nothing: the ϵ-language plan. Above that is the family of plans that all possess a vertex progress measure, as well as the language of their action-based sensors. Eventually we reach a boundary, beyond which action-based sensors cannot be derived directly from a plan. Above this boundary lie plans with crossovers. All of these plans can be expressed as vine graphs, however only some of them can be reduced by CLIP into families of action-based sensors. At the top of the lattice are non-solutions.

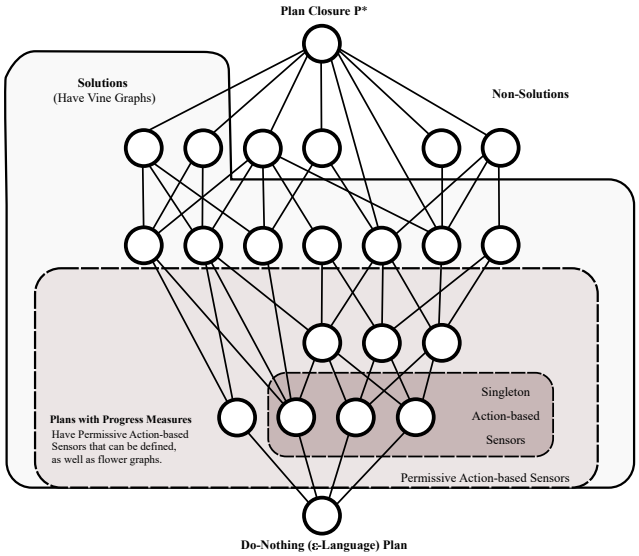


Figure 11. A sketch of a Plan Lattice.

We call the maximal element of the plan lattice the *plan closure*, borrowed from Zhang et al. (2018).

Definition 18. plan closure (Zhang et al. 2018). *Given a world (W, V_{goal}) and a (potentially infinite) set of solutions for W $\{P_1, P_2, , P_3, \dots\}$, there exists a p-graph $P^\star$, which we term the* plan closure*, such that*

$$\mathcal{L}(P^\star) = \mathcal{L}(P_1) \cup \mathcal{L}(P_2) \cup \mathcal{L}(P_3) \dots.$$

Using the plan lattice, we can bring action-based sensors, as well as their upcoming stateful counterparts, into a single construction. At the level of their languages, this allows for us to examine how the combinations of certain executions induce requirements of state, as well as in what instances memorization is actually required. In terms of plans, the set of action-based sensors that are subsumed by a plan are the representatives that would be provided by CLIP; these are the subsets of the original plan language for which no crossovers occur. Through the structure of the lattice, we can see not only that some plans require state while others do not, but also how those requirements come about as well as what kind of behaviors can be modified or discarded to cross between that boundary of reactivity and statefulness.

10 What Lies Beyond Flower Graphs

Figure 10 presents a flower graph form of a plan that solves the problem presented in Figure 1; the current location of the robot in the world has no bearing on what action should be taken next. If we consider a plan such as that in as in Figure 5, the robot's current observation is insufficient for us to determine what actions will make progress. Only though also knowing where the robot is within the plan, which is specific to the route the robot previously committed to (blue or red, in that example), can we determine the robot's future actions.

This observation is the basis for what we consider *state*.

10.1 Defining State, Defining Vines

If state can arise implicitly from plan structure, then it can also be expressly used to encode execution history for our own needs. This raises the question of how to capture only the information needed to execute the full language of the plan. For example, a plan's graph could branch off two different ways depending on if a door is open or closed. While some history is vital for the robot to keep to determine what actions to take in the future, other information can be discarded without impact.

Plans which can be transformed into a reactive plan with an action-based sensor were keeping information about their execution that was unnecessary, while plans with only execution progress measures rely on their structure to encode information such that they can achieve the goal.

10.2 The Impact of Non-homomorphism

Non-homomorphism generally arises from uncertainty in the robot's model of the world. This may be due to uncertainty in the robot's physical components, such as sensors and actuators, but may also arise due to the structure of the plan itself. Even if we can determine a robot's precise location within the world given its joint-execution history, a single vertex within the plan may correspond to different world vertices each time it is visited.

This changing relation between plan vertices and world vertices is captured by the I-Graph. It is also possible for the I-Graph itself to create a non-homomorphic plan, even if the input was homomorphic to the world. By definition, the initial vertex of the I-Graph corresponds to all starting vertices in the world. The empty string ϵ would then potentially end in any starting location in the world, making the plan non-homomorphic. This trivial case aside, this transformation can also occur when obtaining an action-based sensor. After CLIP operates on the I-Graph, the algorithm attempts to reduce the I-Graph down into a reactive plan (as action-based sensor). As a result, the existence of the vertex progress measure means that world vertices that have the same incoming observations will be conflated within the final graph, resulting in a non-homomorphic plan.

10.2.1 Ambiguity and Dynamism in Plans

To discuss the various ways in which plans can be non-homomorphic, we extend the relation presented in Definition 15. This extension includes not only the correspondence between how a plan vertex and world vertex can be reached with a joint-execution, but also *which* joint-executions reach those pairs of vertices in P and W. If we disregard s in this definition, we see that this relation again becomes the one defined in Definition 15.

Definition 19. triple relation. *For a plan P and planning problem W, we can define a relation $T \subseteq V(P) \times V(W) \times (\mathcal{L}(W) \cap \mathcal{L}(P))$, where a tuple $(v, w, s) \in T$ if and only if there exists a joint-execution s on P and W that can be traced to a vertex v in P and a vertex w in W.*

To explain how different types of non-homomorphism arise and are handled, we present two new definitions. The first captures that in nondeterministic plans, the same joint-execution may result in the robot reaching different vertices in the world. We say that plan vertices with this relation are *ambiguous*.

Definition 20. ambiguity in plans. *A vertex v in plan P is* ambiguous *if there exists a joint-execution s and vertices w, w' in planning problem W, where $w \neq w'$, such that both (v, w, s) and (v, w', s) are in T.*

Ambiguity is a statement about our ability to determine the outcome of a single joint-execution, all else being equal. Such nondeterminism may arise from sensor noise, or from lack of precision in the robot's actuation. Imprecision in location can also lead to the possible world vertices of the robot moving from set to set.

However, it's also possible that the same vertex in the plan will map to different vertices in the world depending on the current joint-execution.

Definition 21. dynamism in plans. *A vertex v in plan P is* dynamic *if there exist joint-executions s, s' and world vertices w, w' in planning problem W, where $s \neq s'$ and $w \neq w'$, and both $(v, w, s) \in T$ and $(v, w', s') \in T$.*

Plan vertices with a changing relation to which vertices in the world they correspond to are *dynamic*. The plan in Figure 9b models a version of dynamism. If the robot receives the observation o_G at the start of execution and immediately terminates, the vertex within V_{term} could correspond either to the goal A or the goal G. However, if the robot receives any other observation at the start of its execution, it will (eventually) localize to a single location within the world. At that point, when it reaches the goal, the execution could tell us for certain whether the goal reached was A or G. (Although the definition of dynamism uses single vertices such as w, w', the principle also applies over sets, as seen here.)

Although these definitions are properties of individual vertices, we can broadly refer to plans as ambiguous or dynamic. Additionally, plans can be both ambiguous and dynamic, just ambiguous or just dynamic, or neither.

When creating an I-Graph, ambiguity results in some vertices in the I-Graph mapping to sets of world vertices as opposed to single vertices. This is the partially observable case discussed throughout Section 5, which can easily lead to requirements of state.

Dynamism, however, disappears during the I-Graph creation. The I-Graph's purpose is to remove the constraints of the plan structure through relating plan vertices to world vertices. As dynamism results purely from plan structure, we see that while the same plan vertex may correspond to multiple vertices in the I-Graph, this has no impact on the final I-Graph itself.

Progress measures and action-based sensors can be defined for some non-homomorphic plans, and Section 8 describes the requirements for action-based sensors to exist. For those plans for which action-based sensors cannot be defined, we now further extend CLIP to recognize what information must be retained, and use this knowledge to reconstruct those plans as vine graphs.

10.3 From Execution Progress Measures into Vines

The final part of our algorithm defines a new vine graph based on an execution progress measure. For each instance of a given observation, the algorithm obtains a list of actions from the I-Graph. If the set of actions differs depending on the instance, then the observation is one which the robot must keep track of.

This graph is constructed similarly to the method used to create the I-Graph. Starting with an initial vertex for the vine and the initial vertex for the I-Graph, the algorithm will trace the I-Graph's possible executions in order to construct the new vine graph.

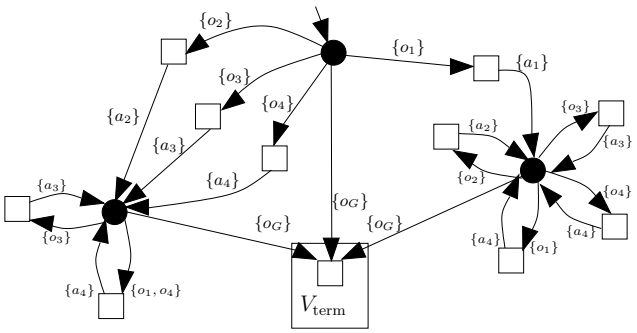
Given a set of outgoing edges from an observation vertex within the I-Graph, one of two cases may occur. Either the set of outgoing observations includes one that the robot must remember, or it does not. In cases such as the plan in Figure 9b, the robot may receive the observation o_1 at the start of execution, which requires it to take a different action than when o_1 is encountered at any other point in the plan. Considering the definition of a contextual sensor in Definition 14, the plan starts in a context C_1, where o_1 corresponds to action a_1. Upon receipt of o_1, a new flower graph is created, reached by the execution $o_1 a_1$. This flower has its own context, C_2. However, if o_1 is not encountered as the first observation at the start of execution, then for subsequent o_1 encounters the robot should take the action a_4. Therefore, the other observations (collectively) transfer to a new flower of their own, a context C_3, representing that the robot needs to remember that an observation *was not* encountered.

In the case that none of the outgoing observations in a set are ones that need to be memorized, they can be added to the vine in the same way a normal action-based sensor is constructed, with the observation going to an action vertex that loops back to the observation.

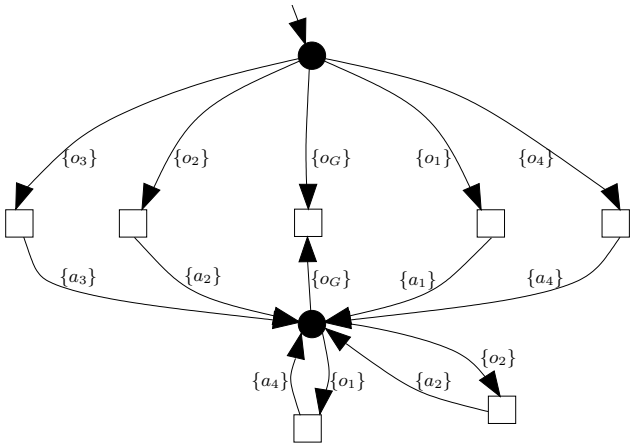
In addition to this branching, it is also possible for the robot to stop tracking information once it is no longer valuable. For example, for both contexts C_2 and C_3, o_1 will always correspond to the same action throughout the rest of execution. Therefore, the list of observations to be tracked is updated after branching. If the action vertex reached by an instance of the observation does not have conflicting actions with any other instances that may follow it, that observation is removed from the list.

Regardless of whether the path branches or not, or if a new action vertex is created or not, the search continues on the I-Graph vertices reached by these observations and actions until all executions have finished. After this, the various goal vertices are then merged into a single vertex. The end result is a graph consisting of several individual flowers, each connected to others by a transitional observation and action. An example of the vine graph for the plan in Figure 9b can be seen in Figure 12a.

Keen readers may notice that the p-graph in Figure 12a is not the smallest possible representation of this plan. This plan can be represented with only two observation vertices, as show in Figure 12b. One of these flowers corresponds to the start of execution, while the other represents that after that initial observation, no additional information needs



(a) The resulting vine given by the CLIP algorithm.



(b) A small vine that makes use of only two observation vertices.

Figure 12. Two vine graphs for the plan in Figure 9b.

to be kept. After the initial o_1, all other instances of o_1 have an action in common, and therefore do not need to be distinguished.

While in this instance the two-flower graph could be obtained by checking that all further instances of the observation agreed on actions before the branch, instead of after, it should be noted that in general the question of obtaining a concise plan for a general planning problem W is NP-complete. O'Kane and Shell (2017) explain this further in their discussion of how filter reduction differs from minimization of a deterministic finite automaton[4]:

> "...we do not require the reduced filter to produce identical results for every observation string in Y, but only on those observation strings in $\mathcal{L}(F)$. In practice, this means that the reduced filter may generate colors for observations strings [*sic*] that are not in the language induced by the original graph, which allows I-states to be "merged" even when their outgoing edges differ."

To translate that terminology: the filter is represented with F, the set Y remains the set of all observations, and I-states can be thought of as an individual observation vertex. Although p-graphs and filters are slightly different structures, they share the complexity that arises from the difference between their structure and their interaction language. An example of this concept applied to p-graphs is the action-based sensors themselves, which have an infinite language when considered alone, but which have a finite language when reduced to their joint-executions with the world.

(Though beyond the scope of this work, interested readers are referred to O'Kane and Shell (2017) for further detail and presentation of heuristic algorithms for concise planning.)

Despite not necessarily being the minimal graph, the resulting vines yield useful information about the plan execution. Each branch out from a flower into two more indicates an additional bit of memory required by the robot to track the decision. The resulting vine structure can be examined for various features, such as if certain choices lead to future requirements for memory when compared to other executions, in which case elimination of certain actions can reduce overall memory requirements. The changes in sensor requirements that result from previous choices also become more apparent within the vine's structure, which may be used to inform design choices.

10.3.1 Turning Flowers into Vines In addition to our ability to create vines for objects that only possess execution progress measures, we can also use vines to combine sets of action-based sensors. In the case of a crossover, CLIP will produce two (or more) action-based sensors, each of which captures part of the original language of the plan from which they were derived. In certain cases, this behavior may be undesirable; as such, recovering the full set of executions requires construction of a vine.

By definition, each action-based sensor disagrees with all others children on at least one action. Separation is therefore required if one does not want to re-introduce crossovers.

There are a variety of ways to implement this; an action-based sensor could be selected non-deterministically from the set at the start of execution, or a vine graph could be constructed based on a concept of "least commitment", maintaining the largest possible set of action-based sensors until the robot must eliminate ones inconsistent with its current history of actions. The desired behavior and resulting structure is up to the designer.

11 From Action-based Sensors to Vines: Another Example

Throughout this paper, we have claimed that the structure of the action-based sensor and vine graphs can be highly valuable in answering questions of design. The example in Subsection 6.1 shows an instance in which broadening the family of action-based sensors allows for accommodation of unusual constraints on the environment. As we will now show, the fact that vines and action-based sensors can both be found by CLIP means that designers can also model how small changes to the world and available sensing information impact the operability and requirements of a robot over time.

As a final example, imagine that the robot we have been designing is a rover being launched to another planet. For obvious reasons, once the robot is in service it will not be able to be repaired or upgraded. In addition to ensuring that our robot is capable of sensing all information needed to remain operational and perform science, as designers we would also like for it to have as long an operational life as possible — for example, by retaining some functionality even in the face of partial component failure.

As this system will naturally have a large number of components, we will focus on a single (simplified) subsystem: power management. Our hypothetical robot has several sensors that monitor its power consumption and current battery levels, and can temporarily shut down systems that are drawing too much power. Additionally, our robot has solar panels that it uses to charge these batteries. To assist the positioning of these panels, the robot has several other tools: sensors for light direction and current, which detect the position of the sun and how quickly the batteries are charging. It also has a set of motors, which it can use to fine-tune the position of its solar panels, to maximize light exposure.

As numerous other systems will be vying for power and chip space, we would like the controller for managing the robot's charging to be as simple as possible. While continually pointing the solar panels towards the sun may gather the most possible current, it restricts the bearing of our robot and requires constant adjustment of the panels. Instead, the power system examines the energy demands and the current battery level. If power demands are high, or if the battery is low, the robot can prioritize adjusting the solar panels to maximize the incoming current, and throttle other systems to keep the battery from dying. If the battery is at a high level, or power demands are low, the robot can divert power from systems for adjusting the panels and their charging circuitry to other systems for exploration of the planet.

While this power management must be performed for as long as the robot functions, we can still describe this problem as a task with a goal. Each time our robot checks on the state of its power systems, we can define its goal state based on the power load. If the load is balanced with the incoming current, or if the battery is high and the load is small, the system can terminate. If the system is unbalanced, the robot should make adjustments until the power consumption is stable.

Figure 13 shows an example action-based sensor for this behavior. We consider that our observation set consists of two readings; first, the current load of the system may be high, low, or balanced when compared to the charge current, designated $l_{\text{HIGH}}, l_{\text{LOW}}, l_{\text{BAL}}$. We also know the general level of the battery: high, low, or critically low, designated $b_{\text{HIGH}}, b_{\text{LOW}}, b_{\text{CRIT}}$. A single observation $y \in Y$ is a pair from these two sets, indicating the current state of the system. (We also assume that the observation set includes the ability to sense the input current, $current_{\text{HIGH}}$ and $current_{\text{LOW}}$, which are also included in any given observation, but for clarity in the figures they are omitted until relevant.)

For actions, our robot may move the solar panels to face the sun, lay the solar panels in a neutral (flat) position, or turn non-vital systems off. (We assume that, if not being held in an off state by the power system, the robot may turn non-vital systems on as needed.) This set is designated U : $\{u_{\text{move\_to\_sun}}, u_{\text{lay\_flat}}, u_{\text{nonvital\_OFF}}\}$. (As part of moving the panels, we assume the robot can tilt its panels left and right, and so these actions are also included in U. These individual actions will become relevant later in the example.)

Our robot faces many potential failures in space. In particular, the above example uses an action to move the panels towards the sun — which presumes that our robot can always identify that direction, and that this sensor is working. Although not part of the explicitly defined observation set, this observation is used to drive whatever routine moves the panels towards light. If this ability fails, then the robot must

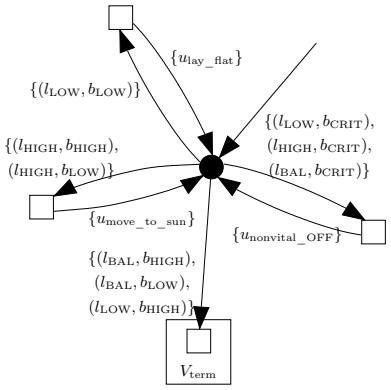


Figure 13. An action-based sensor for a system with the goal of balancing power consumption for a robot. This plan is finite on the world; even if the load is not balanced, as systems are turned on and off, eventually the battery level will be high and the load will be low. If an I-Graph was created for this, our world states would correspond to input currents, battery levels, and loads. The progress measure would therefore be defined as the choices in action that prioritizes either a balanced load, or a high battery level with minimal load.

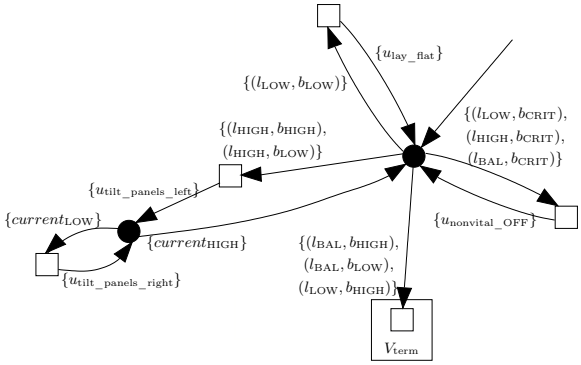


Figure 14. A vine graph demonstrating behavior that adapts to a broken light sensor. We exclude the panel current, $current_{\text{HIGH}}$ and $current_{\text{LOW}}$, from edges where they are not relevant to the action taken, and the same is done with the load and battery observations when they are not relevant to the resulting choice in action.

develop new behaviors for when it has high loads on the system. In such a case, although the sun cannot be directly sensed, its direction can be generally inferred from what angle yields the highest input current. Therefore, our robot now must use a multi-step process to maximize this input. Beginning with a maximum tilt in one direction, it can then adjust the tilt of the panel in the other direction. The output current should rise as it faces a light source, and decrease as it moves away.

Figure 14 shows an example vine graph for this behavior. Observations which would have triggered the robot to adjust its panels towards the sun now transition to a new observation vertex. Slightly abusing our observation labeling, we ignore the current load and battery level and focus only on the input current. Assuming the robot is not keeping past information, the robot should return to normal behavior once the current reaches a certain level. This behavior also makes use of actions for adjusting the panels, $u_{\text{tilt\_panels\_right}}$ and $u_{\text{tilt\_panels\_left}}$.

Another way in which the system might fail is that the motors for adjusting the angle of the solar panels may fail. If they fail at an angle, the robot will have to physically turn to face the sun. This will result in a similar vine as the one before, although the robot will turn towards a light instead of adjusting panels to hit a desired current, but will also impact other systems on the robot. The requirement of facing in a certain direction to charge the batteries means that other actions, such as exploration and movement, are restricted by the battery level and the direction in which it must face.

Although we cannot service our robot once it leaves Earth, we can predict ahead of time what kind of problems may occur, and how this changes the sensor readings and desired responses of the robot. As a result, we can construct vine graphs that suggest new behaviors to compensate, and identify how those behavioral changes may result in impacts on other parts of the system.

12 Conclusion

Having extended the work done by Erdmann we can now obtain not only the sensors that correspond to the backchained solution, but the complete set of all action-based sensors. Given a library of components, this set of sensors can guide practitioners in determining what is simultaneously realizable, respectful of environmental constraints, and sufficiently powerful to accomplish the task. Design and selection of sensors then becomes just a question of intersecting constraints.

However, we have seen that sensing alone may be inadequate, leaving our robots to explore, to learn, and to memorize the information they need. Through modeling plans as directed graphs, the required internal state is encoded directly into their structure. Identifying what precisely this state is, how it manifests in stateful plans, and what properties these stateful plans have, is valuable in understanding how limitations in one aspect of the design (sensing) can be compensated for by other components of the system.

By identifying only the information that must be retained, we gain additional insight into the plan's requirements, and can express these requirements through vine graphs, which extend action-based sensors.

From the perspective of the theorist, vine graphs and action-based sensors provide a useful tool for analyzing information and state requirements of systems. By comparing closely related plans within the plan lattice, the effect of minor changes in behavior on the system can be easily explored. Additionally, the structure of the plan lattice allows for understanding of where requirements for internal state begin, and where sensing alone becomes insufficient to solve the task.

From a designer's perspective, the action-based sensors and vine graphs allow for direct comparison of designs. Given a planning problem, the sensing limits imposed by different sensors that are available to the designer can be applied to the planning problem and its resulting solutions. The resulting action-based sensors can be compared to see if the different sensors require different behaviors, or if the required change in behavior to implement them is suitable for the desired task. If a vine graph is required, then the variation

in sensors can be used to compare the resulting complexity of the plans in terms of their state.

We have also seen that the ability to compare the application of different sensors allows for exploration of additional questions, such as if changes to the same sensor over time result in a planning problem that can no longer be solved, or if the changes can be planned around through the use of more complicated behaviors from the robot.

It is our belief that the tools presented in this work provide a new way for roboticists to explore the tricky connections between desired behavior and required sensing, as well as how choices in either of those areas can have significant impact on the options for the other.

Acknowledgements

This work was supported, in part, by the National Science Foundation through awards IIS-1453652 and IIS-1849249, and from a graduate fellowship provided to Texas A&M University by the 3M Company and 3M Gives.

Notes

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 Erdmann (1995).
2. More than any other robotics work of which we are aware, Erdmann's theory embodies the perspective of Pragmatism, in the William James and John Dewey sense of the term.
3. The word 'state' has been used with extreme care thus far and only in precisely one form, namely to mean something memorized, tracked, or remembered; within the p-graph we have called the elements vertices; other words like 'some condition' and 'particular circumstances' have been deliberately used to avoid the conceit of declaring something to be 'state' and taking its existence so seriously as to believe it an objective concept.
4. A filter (O'Kane and Shell 2017) is a directed graph that is somewhat similar to a p-graph, but which consists only of observations on its edge transitions and yields an output "color" dependent on the given state. The complexity of p-graph plan reduction follows from similar structure.

References

Banarse D, Bachrach Y, Liu S, Lever G, Heess N, Fernando C, Kohli P and Graepel T (2019) The Body is Not a Given: Joint Agent Policy Learning and Morphology Evolution. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099, p. 1134–1142.

Blum M and Kozen D (1978) On the power of the compass (or, why mazes are easier to search than graphs). In: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. IEEE, pp. 132–142.

Brooks R and Matarić M (1993) Real robots, real learning problems. In: *Robot learning*. Springer, pp. 193–213.

Censi A (2015) A mathematical theory of co-design. *arXiv preprint arXiv:1512.08055* .

Censi A (2016) Monotone co-design problems; or, everything is the same. In: *2016 American Control Conference (ACC)*. IEEE, pp. 1227–1234.

Choset H and Burdick J (1995) Sensor based planning. I. The generalized Voronoi graph. In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 2. pp. 1649–1655 vol.2. DOI:10.1109/ROBOT.1995.525511.

Donald BR (1995) On information invariants in robotics. *Artificial Intelligence* 72(1-2): 217–304.

Erdmann M (1995) Understanding action and sensing by designing action-based sensors. *The International journal of robotics research* 14(5): 483–509.

Howard A and Roy N (2003) The robotics data set repository (radish). URL http://radish.sourceforge.net/.

LaValle SM (2019) Sensor Lattices: Structures for Comparing Information Feedback. In: *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*. pp. 239–246.

Lipson H and Pollack J (2000) Automatic design and manufacture of robotic lifeforms. *Nature* 406: 974–8. DOI:10.1038/35023115.

McFassel G and Shell DA (2020) Every Action-Based Sensor. Springer Proceedings in Advanced Robotics. Springer, Cham. ISBN 978-3-030-66722-1, pp. 176–193. DOI:https://doi.org/10.1007/978-3-030-66723-8_11.

O'Kane JM and LaValle SM (2008) Comparing the Power of Robots. *The International Journal of Robotics Research* 27(1): 5–23.

O'Kane JM and Shell DA (2017) Concise planning and filtering: hardness and algorithms. *IEEE Transactions on Automation Science and Engineering* 14(4): 1666–1681.

Pervan A and Murphey TD (2021) Algorithmic Design for Embodied Intelligence in Synthetic Cells. *IEEE Transactions on Automation Science and Engineering* 18(3): 864–875. DOI:10.1109/TASE.2020.3042492.

Saberifar FZ, Ghasemlou S, Shell DA and O'Kane JM (2019) Toward a language-theoretic foundation for planning and filtering. *The International Journal of Robotics Research* 38(2-3): 236–259.

Stachniss C and Grisetti G (2010) Freiburg campus. Accessed June 20, 2021. Available at http://hdl.handle.net/1721.1/62286 in *The Robotics Data Set Repository (Radish)*.

Zardini G, Censi A and Frazzoli E (2021) Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis. *arXiv e-prints* : arXiv:1810.03873.

Zhang Y and Shell DA (2020) Abstractions for computing all robotic sensors that suffice to solve a planning problem. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8469–8475.

Zhang Y and Shell DA (2021) Lattices of sensors reconsidered when less information is preferred.

Zhang Y, Shell DA and O'Kane JM (2018) What does my knowing your plans tell me? *arXiv e-prints* : arXiv:1810.03873.