Advances in Cognitive Systems (2017) 1–12 Submitted 3/2017; published 5/2017

Generating, Executing, and Monitoring Plans with Goal-Based Utilities in Continuous Domains

Pat Langley PATRICK.W.LANGLEY@GMAIL.COM
Institute for the Study of Learning and Expertise, Palo Alto, CA 94306 USA

Dongkyu Choi DONGKYUC@KU.EDU
Department of Aerospace Engineering, University of Kansas, Lawrence, KS 66045 USA

Mike Barley MBAR098@CS.AUCKLAND.AC.NZ
Ben Meadows BMEA011@AUCKLANDUNI.AC.NZ
Department of Computer Science, University of Auckland, Private Bag 92019, Auckland 1142 NZ

Edward P. Katz E.P.KATZ@IEEE.ORG
College of Computer and Information Sciences, Northeastern University, San Jose, CA 95138 USA

Abstract

Like humans, autonomous agents must operate in continuous physical settings which involve competing objectives that change over time. The PUG architecture addresses these issues by combining relational and quantitative descriptions of states, associating numeric utilities to symbolic goals, and using mental simulation to evaluate alternative plans. In this paper, we report extensions to PUG that interleave plan generation with execution and monitoring. We describe each of these processes, how they interact, and how they lead to plan revision when anomalies arise. We demonstrate the extended architecture's behavior in a continuous physical domain, including four classes of scenarios in which unexpected developments lead the agent to revise its plans before continuing execution. In closing, we review work on related topics and discuss avenues for future research.

1. Background and Motivation

We are interested in developing and understanding autonomous agents that support extended operation over time and space. Autonomy requires an agent to respond adaptively to its situation, decide which goals to adopt and which actions to take, and choose how to allocate its resources. Such an intelligent agent depends not only on the ability to generate plans, but also to execute them in the environment, monitor their progress, and replan when necessary. The integrated nature of extended autonomy suggests that we approach it from a cognitive systems perspective.

Consider a robotic agent that pursues a variety of goals during an extended planetary mission in a setting like that depicted in Figure 1. The robot (R1) starts the mission will a full tank of fuel and carrying three sensors. Other objects in the environment include target sites (T1 to T5), samples (S1 and S2), fuel depots (F1), and danger areas (D1). Goals include depositing sensors at target sites, collecting samples, and avoiding dangerous areas. Missions involve multiple goals of

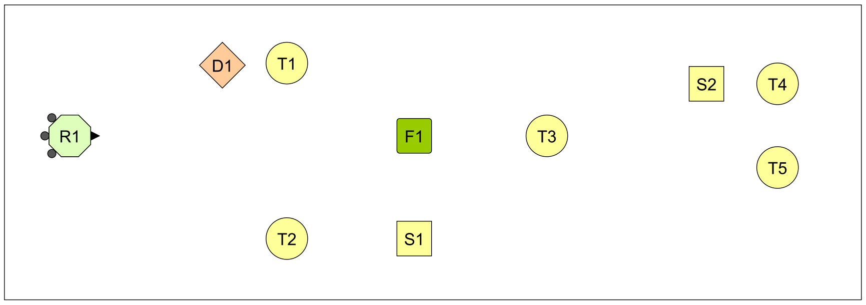

Figure 1. A sample robotic mission scenario that includes one robot (*R1*), five target sites (*T1* to *T5*), three sensors (attached to the robot), two samples (*S1* and *S2*), one fuel depot (*F1*), and one danger area (*D1*). The robot cannot detect all objects in the environment at the outset.

this sort, some of them incompatible, which forces the agent to select some over others. The agent must reason not only about symbolic relations, but about numeric features of the environment, only portions of which may be sensed at a time. As the robot moves through the landscape, new objects become visible, suggesting new goals that interact with the existing plan. We desire a computational theory that integrates inference, planning, execution, and monitoring in such contexts.

In this paper, we extend a framework described elsewhere (Langley et al., 2016) that addresses planning in such continuous and conflicting domains. We review the theory's main claims, along with the representational and processing assumptions of PUG, an architecture that adopts them. After this, we propose additional theoretical tenets that extend these ideas to plan execution, monitoring, and replanning. Next we report augmentations to PUG that implement the new postulates and incorporate the earlier planning mechanism as a module. We demonstrate the extended architecture's behavior on a variety of scenarios that show its ability to detect and recover from surprises during execution. We conclude by discussing links to prior research and plans for future work.

2. Review of the PUG Architecture

In a recent paper (Langley et al., 2016), we reported a computational account of planning and inference in continuous domains that involve conflicting objectives. The new theory revolved around four basic postulates:

- Numeric utilities are associated with symbolic goals;
- Both goals and their utilities are conditioned on belief states;
- A combination of relational structures and numeric attributes describe these states; and
- Planning uses quantitative simulation to evaluate and select operators during heuristic search.

The framework incorporated ideas from earlier efforts, but it also combined them in novel ways to provide a unified account of planning in continuous, multi-objective domains, such as the planetary mission scenario that we outlined earlier.

In the previous paper, we also described PUG (*Planning with Utilities and Goals*), a cognitive architecture designed around these theoretical tenets. Table 1 presents examples of four types of knowledge structures on which PUG relies. Compositional rules define concepts that combine elements into higher-level entities, such as a *vector* that denotes a spatial relation between two objects, including numeric attributes like *distance* and *angle*. Specialization rules define categories to which objects, whether primitive or composite, should be assigned. For instance, both *facing* and *at* are specializations of the *vector* predicate. Another form of knowledge states when to generate goals and the values to associate with them. One such rule states that, if the agent knows about a target site *T*, it should adopt a goal to have a sensor at *T*. Another specifies that the agent should not be near danger sites by assigning a negative utility to this occurrence. Finally, operators state the conditions under which actions have given effects, including quantitative changes produced on each time step and qualitative results at termination. For instance, when the robot is facing an object *O*, moving forward reduces its distance to *O*, consumes fuel, and, ultimately, places the robot at *O*.

The architecture's dynamic structures included the agent's beliefs, goals, and plans, which it generates by composing its knowledge elements. On each cycle, PUG iteratively matches its conceptual rules against descriptions of known objects to infer higher-order relations (e.g., the robot is facing a sample or a sensor is at a target site). The resulting structures combine abstract symbolic relations with detailed numeric attributes. Goal-related rules match in turn against these beliefs, producing goals and associated utilities. The architecture uses forward chaining to explore a space of partial plans, drawing on goal-based utility to guide depth-first heuristic search. Each node in the search tree is an elaboration of its parent that adds one operator and the state it generates. Each state includes not only descriptions of perceived objects, but also inferences about relations among them and goals activated by these relations. Although its plans are symbolic, the architecture invokes numeric simulation to evaluate durative operators, which can take many time steps to complete. Simulation reveals whether an operator will run to completion and predicts the state utility, in terms of matched goals, on each time step. PUG uses average utility to select among applicable operators and thus to guide its search for plans. The process of operator simulation is deterministic, introducing only a constant factor to the architecture's planning time.

For example, suppose that, from its initial position in Figure 1, the robot *R1* observes only the danger site *D1* and the target sites *T1* and *T2*. Inference produces three virtual objects that encode spatial relations between *R1* and *D1*, *T1*, and *T2*, respectively. PUG also concludes that *T1* and *D1* are on the robot's left, while *T2* is on its right. The first generator in Table 1 adds the positive goals *(sensor-at ?sensor T1)* and *(sensor-at ?sensor T2)*, whereas the second rule produces the negated goal *(not (vector ^id (R1 D1)))*. The planner notes that the only applicable operator instances are *(turn-left R1 T1)* and *(turn-right R1 T2)*, both of which it simulates quantitatively, revealing that turning left is slightly faster. Lookahead search determines that the plan *(turn-left R1 T1) (move-to R1 T1) (drop-sensor S1 T1)* achieves a positive goal of delivering a sensor to *T1*, but it has negative utility because the trajectory passes near the danger area. In contrast, the plan *(turn-left R1 T2) (move-to R1 T2) (drop-sensor S1 T2)* takes more time but has positive utility. Moreover, extending it to deliver another sensor to *T1* decreases utility, so the system returns the second alternative.

The earlier paper also reported demonstrations of PUG's behavior on ten different scenarios of the type we have mentioned. These provided evidence that the system's hybrid representations, and its mixture of symbolic with numeric reasoning, supported planning in continuous domains with conflicting objectives. Examples included cases that required selecting among targets when

Table 1. Sample PUG knowledge structures, from Langley et al. (2016) for the robot mission domain, including (a) a compositional rule that infers new relations and their attributes, (b) specialization rules that discriminate among these relations, (c) rules that generate goals and calculate their utilities, and (d) an operator that specifes conditions for applying an action, expected changes on each time step, and ultimate results.

```
(a) ((vector ^id (?r ?o) ^from ?r ^to ?o ^distance ?d ^angle ?a)
     :elements    ((robot ^id ?r ^xloc ?x1 ^yloc ?y1 ^orient ?f)
                   (object ^id ?o ^xloc ?x2 ^yloc ?y2))
     :binds       (?d (*distance ?x1 ?y1 ?x2 ?y2)
                   ?a (*angle ?x1 ?y1 ?x2 ?y2 ?f))
     :tests       ((< ?d 100)))

(b) ((at ^id (?r ?o))
     :specializes (vector ^id (?r ?o) ^distance ?d)
     :tests       ((< ?d 0.2)))

    ((facing ^id (?r ?o))
     :specializes (vector ^id (?r ?o) ^angle ?a)
     :tests       ((> ?a -0.01) (< ?a 0.01)))

    ((sensor-at ^id (?sensor ?target))
     :specializes (vector ^id (?s ?o) ^distance ?d)
     :conditions  ((sensor ^id ?s))
     :tests       ((<= ?d 0.2)))

(c) ((sensor-at ^id (?sensor ?target))
     :conditions  ((object ^id ?target ^type target ^priority ?p))
     :function    (* 100.0 ?p)

    ((not (vector ^id (?r ?o) ^from ?r ^to ?o))
     :conditions  ((object ^id ?o ^type danger) (robot ^id ?r)
                   (vector ^id (?r ?o) ^from ?r ^to ?o ^distance ?d))
     :tests       ((< ?d 20))
     :function    (/ 0.1 (+ (sqrt ?d) 0.01)))

(d) ((move-to ?r ?o)
     :elements   ((robot ^id ?r ^xloc ?x1 ^yloc ?y1 ^orient ?a ^fuel ?f)
                  (object ^id ?o ^type ?t ^xloc ?x2 ^yloc ?y2))
     :conditions ((facing ^id (?r ?o) ^distance ?d)(not (at ^id (?r ?o))))
     :tests      ((> ?d 0.2))
     :changes    ((robot ^id ?r ^fuel (- ?f 0.01) ^xloc (+ ?x1 (dx ?a))
                         ^yloc (+ ?y1 (dy ?a))))
     :results    ((at ^id (?r ?o))))
```

the robot lacked enough sensors for them all, taking detours around danger areas to deliver sensors, making side trips to refuel or collect samples before making deliveries, and deciding not to take additional actions because they would result in lower utility. An important limitation was that, although the architecture generated plans for continuous environments, it could not execute or monitor them. In this paper we attempt to remedy this drawback by extending PUG to incorporate these abilities. The planner certainly has other limitations that we should address in future work, but we will not focus on them here.

3. Theoretical Assumptions

Because we aim to build on the earlier framework, we will adopt its key theoretical assumptions. As noted earlier, these include the ideas that numeric utilities are attached to symbolic goals, that both goals and utilities are conditional on the agent's beliefs, and that these goal-oriented utilities guide decision making during planning. In addition, because the framework focuses on agents that operate in continuous physical domains, it assumes that mental structures include both symbols and numbers and that mental simulation supports the calculation of state and operator utilities. We will take these assumptions as given in this paper.

Thus, we must elaborate the theory to cover plan execution, monitoring, and replanning in a way that is consistent with these constraints, but that also moves beyond them. Our extended account comprises four additional statements:

- Plan execution becomes possible when plan generation has found an acceptable operator sequence using the resources available for heuristic search.
- Plan monitoring compares quantitative mental simulations of expected states with sensing of actual states – and inferences over them – to track the plan's progress.[1]
- Operator anomalies – when the actual conditions, utilities, or results of operators differ from the agent's expectations – lead to replanning.
- Goal anomalies – when goals generated from observed states differ from expected ones – also lead the agent to produce an updated plan.

Replanning involves the construction of new plans, or adaptation of existing ones, that start from the current inferred state but retain the original goal rules with their associated utility functions. None of these assumptions should be especially controversial, although the role of anomalous utilities and goals are somewhat novel, as is the use of quantitative simulation for monitoring.

In the remaining pages, we examine these ideas in the context of an implemented architecture. We cannot prove our assumptions are correct any more than Euclid's axioms; we can only see whether they support the autonomous behavior we desire. However, this necessarily requires making auxiliary assumptions that are not central to the theory and that one might instantiate in different ways. We can clarify and study our core tenets within such an extended framework, but it is important to keep them distinct from implementation details. Nor will we claim that our theory is the only approach to integrated planning, execution, and monitoring, but we believe it offers a simple and elegant account of this interaction on complex tasks that arise in continuous domains.

4. Extending the PUG Architecture

We have incorporated these theoretical ideas into a new cognitive architecture that we call PUG/X, as it extends its predecessor by adding abilities for plan execution and monitoring. In this section, we describe the three main stages of the system's processing, including the conditions for transitions among them. We will not focus on representation or plan generation because PUG/X adopts the same notation and search methods as its precursor, which we already reviewed in Section 2.

1. This differs considerably from most work on plan monitoring, which instead relies only on qualitative simulation.

4.1 Shifting from Planning to Execution

The extended architecture uses the PUG planner, described earlier, to generate a set of plans ranked by their utilities. The module inputs an initial state and background knowledge that includes operators, conceptual rules that define their predicates, and goal-generating rules. As we have noted, the system uses forward chaining through a space of partial plans, which it guides using a form of heuristic depth-first search. The latter assumption is not central to the framework, although it comes closer to human processing than most alternative search methods.

The search process is constrained by three user-specified parameters: the maximum length of plans, the maximum number of nodes to be considered, and the number of alternative solutions desired. The resulting plans need not satisfy all achievement goals, as some may be incompatible, say because there are not enough sensors to deposit one at each target. Achieving one goal may also mean violating others that are stated as negations, such as avoiding the proximity of danger areas. In this sense, the system carries out *partial satisfaction planning* (Benton et al., 2009), finding the highest quality plans it can with the resources available. PUG is not guaranteed to find the highest utility plans, but it considers a sequence of actions to be acceptable only if none of its subplans has higher utility. Thus, goal interactions can lead it to reject a plan that achieves more goals than its ancestors in the search tree. In an extreme case, PUG may favor the null plan of taking no action.

The planner halts when it has found enough acceptable solutions or when it has considered the maximum number of search nodes. This termination criterion serves as the condition for initiating plan execution. At this point, the augmented architecture selects the plan with the highest utility, breaking ties at random, and starts to carry it out in the environment. The execution process continues until monitoring, which we discuss shortly, detects a discrepancy, which in turn leads to replanning using the strategy described above. The current implementation begins anew from the current state, but with no memory of the original plan, although one can imagine variants that attempt plan repair. This cycle continues until there remain no unsatisfied achievement goals or until planning reveals no sequence of operators that produces positive utility, in which case the root plan, which includes no actions, is preferable.

4.2 Executing and Monitoring Plans

Execution and monitoring in PUG/X occur at two distinct levels – qualitative and quantitative – that parallel those in plan generation. The first follows the sequence of operator instances stored in the selected plan. Each plan step specifies an operator and its arguments (e.g., *(turn-right T1))*, the state before its application, the state expected after its completion, the number of iterations required, and the average utility during this interval. These steps describe the plan's qualitative structure, through which the execution module proceeds in order, one operator at a time.

PUG/X uses this qualitative description to organize the monitoring process, but the main work occurs at the quantitative level, which focuses on behavior within each durative operator. Although the planner uses mental simulation to guide search, it does not store the detailed trajectories that result. Instead, the execution monitor replays an operator's numeric simulation, state by state, as it repeatedly applies that operator. For instance, if *(turn-right R1 T1)* took 30 iterations to face toward a target object during plan generation, it repeats those 30 simulated steps during monitoring in tandem with 30 applications in the environment. These include object positions on each step, as well as their pairwise distances and angles, as the robot turns. As during plan generation, this simulation process is deterministic, which means that it introduces only a constant factor to monitoring time. On each pass, the architecture uses its inference rules to elaborate its beliefs about both the planned state and the actual one obtained from sensors, such as whether an object is to the robot's right or left.[2] From the inferred description, it determines whether the operator's application conditions match in both the simulation and the environment, whether the operator's results (termination conditions) are satisfied, and what goals should be active. These let it compute the utilities for the two trajectories. Monitoring compares these pairwise elements, which are important because they were factors considered during plan generation; if disagreements arise, they are cause for concern.

4.3 Shifting from Execution to Planning

These comparisons let PUG/X detect four broad classes of anomalies that lead to replanning. Two concern the details of durative operators, such as turning to face an object or moving toward it. The architecture applies such operators, whether mentally or in the environment, only when their conditions match the state. If the monitor finds that an operator's condition (e.g., that the robot is facing an object) matches in simulation but not in the environment, this difference deserves the agent's attention. Similarly, it should continue applying an operator only until it achieves the intended results. If the monitor notes that an operator has finished (e.g., the robot has reached an object) in simulation but not in the world, this is another important discrepancy. In both cases, the architecture halts execution and returns to planning, taking the perceived state as its starting point. The reverse can also occur, with a simulated operator failing to apply or terminate when the actual operator does; these are less likely events but also constitute grounds for replanning.

An operator's execution can also be anomalous with regard to its expected utility. The observed score can differ from that predicted by simulation when the attributes of objects that appear in a goal's utility function take on unexpected values. These objects need not be mentioned in the current operator to influence utility calculations. For instance, if the amount of radiation measured for a danger area or if the priority of a target has changed since the agent completed planning, then the utility of executing an operator (and thus the entire plan) can diverge from the anticipated amount. When this drops below a certain fraction (0.9 in the runs reported later) of the simulated utility, PUG/X also triggers replanning from its current belief state. When the operator's utility is higher than expected, then it continues executing the plan, as it does when utilities of other plans are higher than expected. Once the architecture has initiated execution, it focuses attention on the selected plan and ignores alternatives unless it enters replanning mode.

A final type of anomaly deals with an unexpected change in goals. Recall that a PUG/X problem specifies not a set of concrete goals, but rather a set of rules that generate goals when their conditions are satisfied. These typically match against relations among perceived objects, such as a sensor being at a target site, but they can also test numeric attributes, such as distance to an object. As noted earlier, the agent may not detect every object at the outset, say because they are too distant or otherwise obscured. As the agent moves through the environment, it may perceive new objects that

2. The architecture does not assume all objects are visible during planning, which is one source of surprises during execution. Current agents do not reason about occlusion, which would appear to require abductive reasoning.

generate new goals, which again leads to replanning. For example, it may detect a new sample that introduces an acquisition goal or an unexpected danger area that leads to an avoidance objective. Goals disappear if their conditions no longer match, which also invokes planning, although this has not arisen in the scenarios we have examined.

4.4 Extended Operation and Scalability

In the introduction, we motivated our research by a desire to support intelligent agents that can operate autonomously over extended periods. The augmented architecture supports such activities by its combined ability to generate plans that achieve goals, execute these plans in the environment, monitor their progress in relation to expectations, and revise them in response to anomalies. For extended operation, the most important type of discrepancies are linked to the agent's goals. PUG/X's ability to introduce new goals, and to update its plans in response, lets it continue activities indefinitely as long as it encounters new situations. This capacity is relevant not only to scenarios that involve mobile robots, but also to settings in which new objects arrive at a given location, as on a factory floor, and even ones in which a fixed set of objects enter new configurations, as on a chess board. PUG/X should support extended operation in any domains of these types.

We should also discuss the architecture's scalability to complicating factors. Increasing the number of perceived objects and depth of search leads to exponential growth in size of the space of plans considered. This should not cause planning time to increase because PUG/X places a limit on the number of plans examined, but it can reduce the probability of finding high-quality solutions. Increasing the number of active goals has at most a linear effect on computation time, as they are used only to compute utilities for simulated and observed states. However, we assume that the agent's limited perception makes it aware of a reasonably small number of objects at a time, and that active goals are based on these perceptions. Thus, like humans, PUG/X's processing time should remain unaffected by changes to environmental complexity, but at the cost of reasoning with limited information and possible loss of plan quality. Moreover, execution and monitoring effort should be independent of plan length, although they should grow with the number of observed objects, which influences inference time, and the number of active goals, which affects utility calculations.

4.5 Implementation and Simplifying Assumptions

We have implemented the PUG/X architecture in Steel Bank Common Lisp, using an unmodified version of the PUG planner as a module. The system makes a number of assumptions that are not central to the theory but that simplify processing in various ways. For example, it assumes that the agent can execute only one operator at a time, operators are deterministic and their descriptions are typically (but not always) accurate, and changes to the environment are typically (but not always) due to agent actions. The current implementation uses heuristic depth-first search during plan generation, constrained by user-specified limits on plan length and nodes considered, but this is not essential to our theory and we are not focused on plan generation here in any case.

We further assume that, although a PUG/X agent may not detect all objects in its environment (e.g., because some are too distant) at the start of a run, once it has perceived an object, the agent has complete information about it in the future. Moreover, the current implementation pauses the execution process during replanning. In settings like our planetary mission, where actions can waste precious fuel or lead to danger, pausing to think briefly seems a reasonable design choice. However, this approach is not essential to our framework and an alternative strategy would be to run the two processes in parallel. In this scheme, the agent would continue its physical activities while generating new courses of action, which may be more appropriate for some domains.

5. Empirical Demonstrations

The theoretical postulates from Section 3 and their incarnation in the extended architecture are plausible, but it is important to show empirically that they exhibit the intended behaviors. As we discuss later, our framework is not directly analogous to others that combine plan generation, execution, and monitoring, which means we cannot carry out direct experimental comparisons. Because PUG/X adopts a representation similar to other architectures, we will not attempt to show its generality here with runs on multiple domains. Neither will we focus on scalability issues, as the impact of exponential search is the same as for other planning systems and its use of numeric simulation within operators introduces only a constant factor to processing time. However, we will demonstrate that the architecture generates, executes, and monitors plans in continuous domains with conflicting goals, responds to each type of anomaly in a reasonable manner, and supports extended operation.

For this purpose, we used a simulated version of the environment that Langley et al. (2016) reported, in which a robot delivers sensors to target sites and collects samples while avoiding danger areas. For each run, we provided PUG/X with the same conceptual rules, goal generators, and operators. The latter included durative operators for turning to face an object, moving toward an object, and refueling at a depot, along with one-step operators for dropping a sensor and collecting a sample. Typical operator models accurately described how actions altered the environment, but we could also make them inaccurate. For most runs, we told the system to search no more than ten operators deep (although each might require hundreds of simulation steps), consider no more than 500 nodes, and return at most eight acceptable mission plans, from which it selected one with the highest utility for execution. These settings were arbitrary, but varying them should not change the overall pattern of results. Table 2 summarizes the types of objects in the environment and their attributes. The table also paraphrases the goal-generating rules, two of which correspond to rules in Table 1 (c), and their associated utility functions. Figure 1 depicts one reasonably complex scenario that has five target sites, three sensors, two samples, one fuel depot, and one danger area.

5.1 Nominal Plan Execution

Our first set of demonstrations show that the architecture operates as intended when monitoring during the execution process reveals no anomalies. Here we ran the system on the same ten scenarios reported by Langley et al., which had from two to four target sites, one to two sensors, zero or one samples, and at most one danger area. Plans ranged in length from three to 15 operators, on the assumption that the agent could perceive all objects in the environment and actual operator behavior closely followed their mental simulations, so no anomalies arose. Thus, the system executed each of the ten solutions without need for revising its plans. Figure 2 depicts the most complex of these scenarios, which includes two sensors, three targets, one sample, one fuel depot, and one danger

Table 2. (a) Object types and associated attributes for the robot mission domain and (b) English paraphrases of rules for generating concrete goals and their associated utilities.

(a) *Robot: id, xloc, yloc, orientation, fuel-level*
Target: id, xloc, yloc, priority, visibility
Sensor: id, xloc, yloc, held-by, visibility
Sample: id, xloc, yloc, held-by, visibility
Depot: id, xloc, yloc, visibility
Danger: id, xloc, yloc, radiation, visibility

(b) *If there is a target T with priority P,*
then post a goal to have some sensor S at T
with utility $10 * P$.

If there is a robot R and a sample S,
then post a goal to have S held by R
with utility 5.

If there is a robot R and a danger O
and the distance from R to O is D
and the radiation level of O is L,
then post a goal for R not to be near O
with utility $L/(D^2 + 0.01)$.

area. Here PUG/X found a reasonable plan that collected sample *S1*, delivered a sensor to target *T3*, refueled at depot *F1*, and delivered a sensor to target *T1*. The system avoided the other target site, *T2*, because it was close to the danger area. In this and other nomimal runs, the architecture could perceive all objects at the outset, so its goals did not change over time.

However, it was also important to show that the augmented PUG does not concern itself with ensuring that execution replicates the precise details of its plans. For this reason, we reran the architecture on two of the scenarios from the first set, but modified the *move-forward* action so it consumed fuel at 1.1 times the rate in its operator description. This was not enough to keep the agent from carrying out the plan, and fuel level appeared in no goals that impact utility, so the system did not view the difference as anomalous and continued to execute the original plan without revision. At first glance, this suggests that PUG/X distinguishes between significant variations and ones it can ignore, but one can design cases in which more rapid consumption would cause the robot to run out of fuel and become stranded. A more robust monitoring system would revise the operator's parameters and use the revised model to anticipate problems with its current plan.

5.2 Handling Operator and Utility Anomalies

Our second set of demonstrations aimed to show that the architecture deals appropriately with operator anomalies. In one run, we altered the environmental behavior of *move-forward* so that, rather than moving in a straight line toward an object, it veered slightly to the right. This meant that, after enough steps, the agent was no longer facing the object, so this operator's application condition matched in the simulation but not in the environment. PUG detected this discrepancy, halted execu-

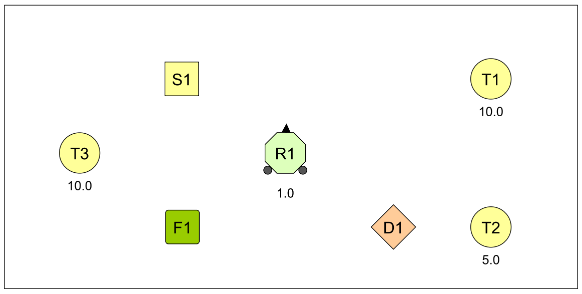

Figure 2. A reasonably complex scenario in which no anomalies arose during execution monitoring, so that no replanning was required. This includes one robot (*R1*), three target sites (*T1*, *T2*, and *T3*), two sensors (attached to the robot), one sample (*S1*), a single fuel depot (*F1*), and one danger area (*D1).*

tion, and generated a new plan whose first action was to turn back to the object. The faulty action model led the agent to repeat this loop multiple times before it finally reached the target. In another case, we arranged for a sample to move slightly during the approach; here the agent stopped, replanned, and executed the new sequence without further surprises or incidents.

In a third run, we changed the rate at which the robot moved toward an intended object. This led PUG/X to expect completing the operator's application well before it would actually terminate, with the monitoring process detecting the discrepancy on the simulated operator's completion. This produced behavior similar to the previous example in which the robot veered to the right: the architecture halted execution, replanned, and continued a number of times, until it eventually reached the target object in a single step, before monitoring had a chance to interrupt execution. A more effective response would instead alter, at least for the current plan, the expected rate at which the operator changes the environment. We intend to incorporate this idea in future versions of the architecture.

Finally, we arranged for an increase in the observed radiation level from a danger site during plan execution. Because the agent uses a negated goal to indicate its desire to avoid such objects, with closer proximity and higher radiation producing lower utilities, this caused it to invoke replanning when the ratio with the expected amount fell below a threshold (set to 0.9 in our runs). However, the system did detect this anomaly until executing the *move-forward* operator had taken the robot on a trajectory very near the danger site. In future work, we should extend PUG/X to revise the environmental description it uses during simulation, in this case the radiation level, and anticipate utility decreases in advance, rather than detecting them only when they become serious.

5.3 Responding to Anomalous Goals

Another set of demonstrations examined the unexpected creation of goals caused by perception of objects that were not visible to the agent when it created an initial plan. To simulate occlusion and similar factors, we arranged for different objects to become visible at different distances as determined by a 'visibility' attribute. The agent could perceive an object with visibility 10 from ten or fewer units but could detect one with visibility 3 from no more than three units. Thus, a given plan could only take into account objects that were currently visible or observed previously.

In one run, the system repeatedly became aware of new target sites during its approach to an already selected site. In each case, it halted execution and generated a new plan before proceeding. Because the older target was closer, the new plan retained the operators of the original but included additional ones for delivering a sensor to the new target. In another run, the agent perceived a new danger area just beyond a target site to which it was in the process of delivering a sensor. In this case, replanning led it to select a different target that was further away but more distant from danger, as this had higher overall utility.

We also ran PUG/X on scenarios in which, during the robot's movement toward a target site, it detected a sample, previously unseen, just off its path. This produced a new goal to acquire the sample, which halted execution and invoked replanning. In this case, the new plan included a side trip to collect the object before continuing to the target site for sensor delivery. In different scenarios, the agent became aware of a previously hidden danger area near its path to a target site it was already approaching. In one case, this led to an alternative plan that took the robot to another target where it could deposit a sensor. In another, lacking other targets, the agent elected to forgo any other actions, as none of them offered higher utility than doing nothing.

A more complex scenario, already seen in Figure 1, combined many of these elements. Here the agent could initially detect only targets T1 and T2, as the nearer danger area, D1, was not visible. In response, PUG/X formed an initial six-operator plan that would first deliver sensor to T1 and then another to T2. However, as the robot actually moved toward T1, danger D1 became visible and caused the system to reconsider. This led to a new plan that delivered a sensor to T2 before T1, moving around D1. As the agent approached the former, it detected a third target, T3, and a fuel depot, F1. In response, the architecture generated a new plan that continued the T2 delivery but replaced the T1 visit with a refueling stop and delivery to T3. During the final leg, the system became aware of two other targets, T4 and T5, and extended its plan to include delivery to the latter, which was slightly closer. However, movement in this direction revealed another sample, S2, which led to a side trip and sensor deposit at T4, which now offered higher utility than delivery to T5.

5.4 Demonstrating Extended Operation

Recall that one of our main research aims was to support autonomous operation over extended periods. To demonstrate that PUG/X exhibits this ability, we designed another scenario that involved delivering sensors to 20 distinct target sites that were arranged in a roughly linear chain. The sites were distant enough from each other that the agent could detect only a few at a time, which means the goals it adopted initially were a small fraction of the total possible. We avoided the need for the robot to carry many objects by altering the *drop-sensor* operator to create and deposit a new sensor when it is invoked near a target site. We also distributed nine fuel depots along the chain of targets so the robot can refuel as needed. We did not include any samples for the robot to collect or danger areas to avoid, as their presence would effect extended operation in only minor ways.

We ran PUG/X in this environment, setting the robot at one end of the target chain. Initially, the agent could sense only one of the targets, so it formulated a simple plan that would deposit a sensor there. As it approached the first target site, the system detected two new objects: a fuel depot and another target. This produced a new goal to deposit a sensor at the second site, which led the architecture to replan. The result included the original operators plus new ones for a sensor at the second site. PUG/X then began executing this plan, which proceeded as expected until the robot approached the second target, when it perceived a third site. This generated another anomolous goal, interrupted execution, and produced an extended plan, this one including actions for refueling before delivering a third sensor, which the system started to carry out. This cycle of execution, target perception, goal creation, and replanning repeated until the robot has deposited sensors at all 20 target sites, providing evidence that PUG/X can interleave planning and execution indefinitely as long as the environment suggests new goals and resources are available to achieve them.

6. Related Research

The AI literature has seen substantial research, since at least the 1970s, on interleaving planning, execution, and monitoring for physical agents, and we cannot cover it all here. This area has seen both theoretical progress and successful applications, as evidenced by Estlin, Rabideau, Mutz, and Chien's (2000) work on the CASPER system. Instead, we focus our comments on four sub-paradigms that we believe are most relevant to our approach, in each case discussing similarities and differences. We will not claim that our work is equal to its many predecessors in terms of efficiency, scalability, or potential for near-term applications. However, we will argue that, although most ideas in our theory have appeared previously, none of their earlier efforts have combined them into an architectural framework that integrates planning, execution, and monitoring – including goal generation – in continuous domains with conflicting objectives.

One important body of work comes from the cognitive architecture community. Frameworks like Soar (Laird, 2012) and ACT (Anderson, 1993), motivated originally by accounts of human problem solving, have the ability to reason over both symbolic and continuous representations, but particularly relevant to the current discussion is Laird and Rosenbloom's (1990) extension to Soar, which integrated planning with execution in external environments.[3] Similarly, Haigh and Veloso (1998) reported ROGUE, an extension to the Prodigy architecture (Carbonell et al., 1990) that interfaced with a mobile robot. These systems dealt with both unexpected events and insertion of new goals through replanning, making our framework similar in spirit. PUG/X's reliance on numeric simulation and utility calculation is atypical, although Soar's recent incorporation of expected values has similarities to the latter. Even more relevant is Cox et al.'s (2016) MIDCA architecture, which includes a metacognitive layer that generates goals, monitors plan execution, detects anomalies, and repairs plans as necessary. This is an example of recent work on *goal reasoning* (Aha, Cox, & Muñoz-Avila, 2013), which addresses similar research issues and ideas.

Another paradigm, known as 'three-tiered architectures', emerged from a focus on robotics, with early work by Fikes, Hart, and Nilsson (1972) inspiring later efforts. A prime example was Bonasso et al.'s (1995) 3T architecture, which used a symbolic planner for high-level decision making, reactive behaviors for reactive control, and a module for intermediate processing. This middle layer invoked 'reactive action packages' that, once initiated, monitored the situation in ways similar to PUG/X and detected cases that required replanning. A key difference was that 3T's modules operated at different levels of aggregation, while our architecture offers a more unified theory of planning and execution that comes closer to work on teleoreactive programs (Nilsson, 1994; Clark & Robinson, 2015), which also incorporate durative operators.

3. Trafton et al. (2013) report an ACT-R extension that interfaces controls a robot, but its planning abilities are unclear.

The AI planning community has also developed systems that integrate plan generation, execution, and monitoring. For instance, Georgeff and Lansky's (1987) PRS used domain knowledge to general hierarchical plans, execute them in the environment, and monitor their progress. Other knowledge was responsible for changing goals in response to unexpected events and reasoning about tradeoffs among them. Wilkins' (1985) SIPE had a similar flavor, although it relied on a more principled taxonomy for the types of problems that can arise during execution and appropriate responses. Our framework adopts similar assumptions, although it keeps knowledge about operators, concepts, and goals distinct, and its incorporation of utility supports the detection of a new kind of discrepancy between the agent's expectations and its observations.

Perhaps the closest research in the planning community comes from Talamadupula et al. (2010), who developed an integrated system to guide a mobile robot in complex office settings. They combined a partial satisfaction planner (Benton et al., 2009), a rule-based mechanism for creating weighted goals under specified conditions, perceptual and control modules that handled low-level processing, and a dialogue subsystem for communicating with humans. The first two elements overlap considerably with PUG/X processes for planning and goal generation, especially as the latter interrupted ongoing execution and led to replanning. However, their operators did not support fine-grained monitoring through numeric simulation, and their goal utilities did not vary over time.

Finally, we should mention Markov decision processes (Puterman, 1994), which also incorporate a repeated cycle of perception, lookahead search, calculation of expected utilities, and execution of the best action, but which differ in important ways. PUG/X constructs a plan and executes it, resorting to replanning only when it encounters anomalies, whereas Markov decision processes effectively replan after every action. In addition, research on the latter emphasizes optimal behavior, whereas our framework adopts instead the more human-like strategy of satisficing (Simon, 1956). Finally, they assume a fixed utility function, whereas our framework lets the agent's goals and their associated utilities change over time. We maintain that these differences make our approach more human-like and more likely to support agents that exhibit extended autonomy.

7. Concluding Remarks

We have presented a new theory of integrated plan generation, execution, and monitoring for continuous physical domains. The framework builds on an earlier one that associates numeric utilities with symbolic goals, assumes goals and utilities are conditional, combines relational structures with numeric attributes to describe states, and uses quantitative simulation to evaluate operators. Our expanded theory states that mental simulation of operators also supports plan monitoring during execution, and it posits that anomalies related to operator application conditions, termination criteria, and utilities lead to replanning, as do unexpected changes in the agent's goals. We described an extension to the PUG architecture that embodies these theoretical tenets, explaining when the system shifts from planning to execution, how monitoring compares expectations to observations, and which discrepancies lead to replanning. In addition, we reported the architecture's operation on scenarios that demonstrated its ability to handle each type of anomaly, as well as extended operation.

Despite the progress we have reported, our research program would benefit from further effort. We should show the framework's generality by testing it in other continuous domains that require integrated planning, execution, and monitoring. We should also extend the current architecture to use hierarchical task networks to represent, generate, and execute plans, which will reduce search and improve scalability by grouping actions into common sequences. As it stands, PUG/X invokes only a single operator at a time, and future versions should support concurrent execution, such as turning and moving forward to produce curved motion around obstacles. Moreover, the system should revise its models for how operators alter the environment when they prove inaccurate by revising parameters or even the functional forms of difference equations. This should let the agent anticipate the need for replanning before execution produces undesirable situations it might otherwise avoid.

We have shown how the system operates in domains with incomplete information, generating new goals when it perceives new objects and replanning in response. We should also extend the architecture to handle uncertainty, as might arise with measurements whose precision varies with distance. One response would be to replace numeric values with probability distributions and use the latter to calculate goal utilities during planning and execution. Another would be to create and store conditional plans that incorporate alternative outcomes during execution. Future versions should also support not only closed-loop monitoring that compares expectations to observations on every time step, but also less frequent sensing strategies in predictable domains. Finally, we should explore variants that attempt to repair the current plan rather than constructing them from scratch, which could reduce search substantially. These extensions should provide a more complete account of plan generation, execution, and monitoring in continuous settings.

Acknowledgements

This research was supported by Grant N00014-15-1-2517 from the Office of Naval Research, which is not responsible for its contents. We thank David Aha, Michael Cox, Mark Roberts, and Stephanie Sage for useful discussions that helped refine the ideas in this paper.

References

Aha, D. A., Cox, M. T., & Muñoz-Avila, H. (2013). (Eds.). *Goal reasoning: Papers from the ACS workshop*. Baltimore, MD: Cognitive Systems Foundation.

Anderson, J. R. 1993. *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Benton, J., Do, M., & Kambhampati, S. (2009). Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence*, *173*, 562—592.

Bonasso, R. P., Kortenkamp, D., Miller, D. P., & Slack, M. (1995). Experiences with an architecture for intelligent, reactive agents. *Proceedings of the International Workshop on Agent Theories, Architectures, and Languages* (pp. 187–202). Springer.

Carbonell, J. G., Knoblock, C. A., & Minton, S. (1990). Prodigy: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence*. Lawrence Erlbaum.

Choi, D. (2011). Reactive goal management in a cognitive architecture. *Cognitive Systems Research*, *12*, 293–308.

Clark, K. L., & Robinson, P. J. (2015). Robotic agent programming in TeleoR. *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 5040–5047). Seattle: IEEE.

Cox, M. T., Alavi, Z., Dannenhauer, D., Eyorokon, V., Munoz-Avila, H., & Perlis, D. (2016). MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (pp. 3712–3718). Phoenix, AZ: AAAI Press.

Estlin, T., Rabideau, G., Mutz, D., & Chien, S. (2000). Using continuous planning techniques to coordinate multiple rovers. *Electronic Transactions on Artificial Intelligence*, *4*, 45—-57.

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial intelligence*, *3*, 251–288.

Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. *Proceedings of the Sixth National Conference of Artificial Intelligence* (pp. 677–682). Seattle: AAAI Press.

Haigh, K. Z., & Veloso, M. M. (1998). Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots*, *5*, 79–95.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., & Rosenbloom, P. S. (1990). Integrating execution, planning, learning in Soar for external environments. *Proceedings of the Eighth National Conference of Artificial Intelligence* (pp. 1022–1029). Boston: AAAI Press.

Langley, P., Barley, M., Meadows, B., Choi, D., & Katz, E. P. (2016). Goals, utilities, and mental simulation in continuous planning. *Proceedings of the Fourth Annual Conference on Cognitive Systems*. Evanston, IL: Cognitive Systems Foundation.

Nilsson, N. (1994). Teleoreactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Puterman, M. L. (1994). *Markov decision processes*. New York: John Wiley.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, *63*, 129–138.

Talamadupula, K., Benton, J., Schermerhorn, P., Kambhampati, S., & Scheutz, M. (2010). Integrating a closed world planner with an open world robot: A case study. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 1561–1566). Atlanta: AAAI Press.

Trafton, J. G., Hiatt, L. M., Harrison, A. M., Tamborello, F., Khemlani, S. S., & Schultz, A. C. (2013). ACT-R/E: An embodied cognitive architecture for human robot interaction. *Journal of Human-Robot Interaction*, *2*, 30–55.

Wilkins, D. E. (1985). Recovering from execution errors in SIPE. *Computational Intelligence*, *1*, 33–45.