Advances in Cognitive Systems 4 (2016) Submitted 4/2016; published 6/2016

Goals, Utilities, and Mental Simulation in Continuous Planning

Pat Langley PATRICK.W.LANGLEY@GMAIL.COM
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

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

Edward P. Katz E.P.KATZ@IEEE.ORG
Silicon Valley Campus, Carnegie Mellon University, Moffett Field, CA 94035 USA

Abstract

Like humans, autonomous agents will need to operate in physical settings that involve competing objectives which may be incompatible and vary in importance over time. They will also need to reason about both qualitative relations and quantitative attributes to produce behavior that is appropriate to the situation. In this paper, we report PUG, a problem-solving architecture that combines conceptual inference with plan generation, encodes both relational and quantitative content, and integrates symbolic goals with numeric utilities. Mental simulation plays a key role in evaluating operators, guiding search, and determining when a plan is acceptable. We describe the system's representational assumptions and the mechanisms that operate over them, after which we demonstrate its behavior in settings that involve conflicting goals with utilities that vary over time. In closing, we discuss related work and outline our plans for future research.

1. Background and Motivation

There is general agreement that autonomous agents have the potential to aid society in many ways. We will say that a system is autonomous if it operates over time, responds adaptively to its situation, and, although it may interact with others, decides for itself which actions to take, which goals to pursue, and how to allocate its physical and cognitive resources. Humans clearly exhibit substantial autonomy, and attempting to model this ability in computational terms, at least in its high-level features, offers a promising path toward replicating it in machines.

Consider a scenario in which a robotic agent with a number of goals pursues an extended planetary mission. Some goals involve achieving desired situations, such as depositing sensors at specific target sites, while others revolve around maintaining certain conditions, such as having enough fuel, or avoiding others, such as dangerous areas. A complex mission will involve many such goals, some even mutually exclusive, so that the agent must decide which ones to pursue. Moreover, these goals may only be active under some conditions and they may have different values at different times. We desire a computational theory that supports all of these abilities.

The AI community has certainly addressed mechanisms for pursuing objectives over time, but research has been divided into two responses to this issue. The planning community has viewed problems as sets of explicit goals encoded as logical literals, typically having equal importance. In contrast, the heuristic search and reinforcement learning communities state agent objectives as numeric evaluation functions, usually with simple features as components. We hold that both frameworks are overly simplified. A robust autonomous agent must be able to express sets of relational goals that describe complex target states, but it must also be able to encode the utility of those goals. The agent should also have the ability to generate new goals and alter their utilities, rather than assuming both come from a human, as in most prior work. In this paper, we report a novel theory that unifies these two paradigms and uses it to guide generation of plans in continuous domains.

Our approach incorporates many ideas from earlier research efforts. We borrow distinctions about representation and processing from the paradigm of cognitive architectures (Langley, Choi, & Rogers, 2009), as well as mechanisms for drawing logical inferences and for generating multi-step plans. Naturally, we build on prior work that uses goals and utilities to guide this process, but we also incorporate Nilsson's (1994) notion of durative operators and more recent techniques for goal reasoning (Aha, Cox, & Munoz-Avila, 2013). The novelty lies in their combination to support behavior in continuous domains that involve conflicting and time-varying objectives. As in other research on cognitive systems, the primary contribution involves integration of ideas previously seen as disconnected to achieve entirely new capabilities.

In the next section, we present the primary theoretical postulates that underlie our work, after which we describe an implemented problem-solving architecture that incorporates them. Next we demonstrate the latter's behavior on a number of planning tasks that involve reasoning about conflicting goals with time-varying importance in a continuous domain. Our purpose has been to devise a theory that addresses important but understudied concerns, so we will not compare our system experimentally to others that were designed with different aims. We conclude by discussing other work that is relevant to our theoretical tenets and proposing avenues for future research.

2. Target Abilities and Theoretical Assumptions

As we have just explained, we desire a computational framework that lets an intelligent agent reason about and plan its activities in continuous domains. Such settings raise a number of challenges, in that they require abilities to:

- Operate over space and time in contexts that involve competing and even inconsistent objectives;
- Treat some objectives as more important than others, but also vary their influence on behavior with the particular situation;
- Reason about activities in domains that involve change over time in both qualitative structure and quantitative attributes; and
- Produce reasonable behavior that balances tradeoffs among the agent's different objectives in a situation-aware manner.

The scenario described earlier offers one example of such a setting, but these issues arise in many other contexts. For instance, a mission to evacuate injured humans from a damaged building would include objectives for finding and extracting people, not injuring them further, not losing person-

nel or equipment in the process, maintaining communication with the outside, avoiding additional damage to the building, and so forth. Moreover, these goals may interact with each other, making it impractical or impossible to achieve or maintain them all, and their importance may change with time and circumstance. Traditional work on problem solving and planning, in both psychology and AI, has sidestepped such issues despite their central role in complex cognition.

The account of problem solving we have developed to explain these abilities incorporates five central theoretical assumptions:

- *Goals are the locus of utility*. Each symbolic goal has an associated numeric score that indicates its desirability, with the total utility for a state distributed among these goals.
- *Both goals and their utilities are conditional*. Goals may be active only under certain symbolic conditions and their associated utilities may be numeric functions of the agent's situation.
- *Mental structures include both symbolic and numeric content*. Concepts, operators, beliefs, and goals incorporate not only relational structures but also numeric attributes that influence utilities.
- *Goal-oriented utilities play a central role in problem solving*. Utilities that accrue over the course of a plan guide choices during heuristic search and determine judgements about plan success.
- *Mental simulation underlies the estimation of operator utility*. Simulation involves applying an operator repeatedly until termination, updating numeric attributes on each time step, matching active goals, and combining their values over the trajectory.

In the remainder of the paper, we explore these ideas in the context of an implemented planning architecture. However, this means introducing additional assumptions that are not our central interest, and one can imagine other architectures that incorporate our core claims in different ways. Because we can only study the assumptions within such a larger framework, we cannot offer compelling evidence that they are strictly necessary for the abilities we desire, but we will demonstrate how they support them in direct ways.

3. An Architecture for Planning with Utilities and Goals

We can now turn to PUG, an architecture for planning in continous domains that incorporates the theoretical tenets just proposed.[1] We begin by describing the system's knowledge about concepts, goals, and actions. After this, we present its notation for encoding specific beliefs, goals, and operator instances, along with how it embeds them into plans and search trees. Finally, we clarify its mechanisms for conceptual inference, goal generation, heuristic search, and mental simulation.

3.1 Conceptual, Goal, and Operator Knowledge

Following ICARUS (Langley et al., 2009), the PUG architecture distinguishes between knowledge about concepts, which it uses to describe situations in the environment, and skills or operators, which it uses to specify activities that alter the environment. We will see that these long-term structures play very different roles in the system's operation. This structural distinction is far more important than the rule syntax, which adopts a notation similar to that in Soar (Laird, 2012) and ACT-R (Anderson & Lebiere, 1998).

1. The acronym PUG expands to *Planning with Utilities and Goals*, which is not very euphonic but at least is short.

Table 1. Partial conceptual knowledge for the robot mission domain, including (a) a compositional rule that infers new relations and their attributes and (b) specialization rules that discriminate among these relations.

```
(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)))

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

    ((at-ahead ^id (?r ?o))
     :specializes (at ^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)))
```

The framework further distinguishes between *compositional* rules, which define concepts that combine primitive objects into new virtual objects, and *specialization* rules, which describe categories into which objects, primitive or otherwise, should be assigned. Table 1 (a) shows a sample compositional rule that defines a vector relation between the robot and another object, such as a target site. This refers to numeric attributes of the components, calculates the distance and angle between them, includes a test on the former, and associates the original and computed values with the *vector* relation. Table 1 (b) presents sample rules that specialize this concept. For instance, the *at* relation holds when the robot and target are less than 0.2 units distance apart, whereas *ahead* applies when the target's angle relative to the robot's orientation falls between 0.01 and -0.01 degrees. A third rule, *at-ahead*, describes situations in which both of these relations hold, whereas *sensor-at* is a variant of *at* in which the first object is a sensor. These rules effectively place a partial ordering on the predicates in their heads, much as in a discrimination network (Feigenbaum, 1963).

Another form of knowledge in PUG states when it should generate specific goals and what utilities they should receive. Table 2 shows two examples of such structures. The first rule indicates that, if the agent knows about a target site, it should create a goal for having an unspecified sensor at that site, and that its utility should be 100 times the priority associated with the site. The second rule gives the conditions on establishing a negated goal that a robot should not be within 20 units of a known danger point. Here the goal's utility varies inversely with the distance, ranging from 10.0 at zero units to only 0.02 at 20 units.

Table 3 shows examples of operators, the final type of knowledge. These have a format similar to STRIPS operators or production rules, in that they describe conditions for carrying out an action

Table 2. Two rules for the robot mission domain that specify the conditions for generating specific goals and the functions used to calculate their utilities.[3]

```
((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)))
```

and the results of its application. Positive structures in the *:results* field indicate elements that become true at termination, while negative ones describe elements that become false. The primary difference from traditional notations is the *:changes* field, which specifies how numeric attributes of objects change on each application step. This is important for durative operators, like *move-to* and *turn-left*, which may require many iterations to achieve their intended results.

3.2 Beliefs, Goals, Operator Instances, and Plans

Naturally, PUG also incorporates more dynamic structures that its conceptual and operator knowledge can inspect and modify, similar to the contents of working memory in a production system. These include primitive beliefs like those in Table 4 (a), composite beliefs like those in 4 (b), and specializations of such composites, like those in 4 (c). They take the same form as the heads of conceptual rules except that constants replace variables. Both type of belief can vary across states in plans and simulations. Only numeric attributes change for primitive elements, like robots and sensors, and the same typically holds for composite beliefs. In contrast, specialized beliefs are added and removed regularly as their conditions become true or false, respectively.

Goals are similar to specialized beliefs, in that they may be added or retracted depending on whether their conditions match against the agent's beliefs. However, they differ in three other ways. First, they may contain unbound variables that can match against any constant term. Second, they are stored separately from beliefs to ensure they remain distinct. Finally, each goal has an associated utility that itself can vary over time as the agent's belief state changes. We should also mention operator instances, which are instantiated versions of operators like those from Table 3 where constants have replaced variables except for some of those in negated conditions.

The PUG architecture utilizes beliefs, goals, and operator instances as building blocks to express larger-scale structures that describe its intended behavior over time. One type of structure is the *problem*, which specifies an initial state (a set of beliefs) and a set of goal rules. We refer to an elaborated problem that specifies an operator instance – and possibly a subproblem – as a *plan*. A plan in PUG specifies an hierarchical decomposition of a problem into simpler problems that produce goal-relevant states. Here we focus on plans in which the associated operator's conditions

3. The syntax for negative goals is counterintuitive. This rule includes the positive condition *(vector ^from ?r1 ^to ?o ^distance ?d)*, but this makes sense because, as we will see, a negated goal only has (negative) utility when violated.

Table 3. Three operators for the robot mission domain that specify the objects involved, the conditions for application, the expected changes on each time step, and the ultimate results. The functions *dx* and *dy* compute changes in *x* and *y* coordinates.

```
((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))))
((turn-left-to ?r ?o)
 :elements   ((robot ^id ?r ^orient ?f) (object ^id ?o ^type ?t))
 :conditions ((to-left ^id (?r ?o) ^angle ?a) (not (at ^id (?r ?o)))
              (not (at-with-no-sensor ^id (?r ?other))))
 :changes    ((robot ^id ?r ^orient (+ ?f 1)))
 :results    ((ahead ^id (?r ?o)) (not (to-left ^id (?r ?o)))))
((drop-sensor ?r ?s)
 :elements   ((robot ^id ?r ^xloc ?x1 ^yloc ?y1)
              (sensor ^id ?s) (object ^id ?t ^type target))
 :conditions ((carrying ^id (?r ?s)) (at ^id (?r ?t))
              (not (sensor-at ^id (?s ?t))))
 :changes    ((sensor ^id ?s ^held none ^xloc ?x1 ^yloc ?y1))
 :results    ((at ^id (?s ?t)) (not (carrying ^id (?r ?s)))))
```

match in the initial state, such as those produced by forward chaining. Thus, each element in a plan specifies an initial state, an operator instance, and a successor state that serves as the initial state for the subplan. For instance, a robot might have a plan to turn toward a target site, with a subplan to move toward it, and a subsubplan to drop a sensor there. The resulting decomposition maps onto a sequence of three operator instances, but storing it as a hierarchical structure has advantages for bookkeeping. We do not distinguish between partial and complete plans because, in some cases, it will be impossible to satisfy all of the active goals, making all plans effectively partial.

The system organizes candidate plans into a search tree, with the original problem – an empty plan with no operator or subproblems – serving as the root node. Each child in the search tree elaborates on its parent by committing to an operator instance, adding a successor state, and introducing a subproblem. Each node specifies as its focus a terminal structure in the problem hierarchy that has not yet been decomposed, along with the average utility accrued over the course of its partial plan. Nodes in the search tree also include information about their parents, for use in backtracking, and which operator instances have been tried in their various children.

3.3 Inferring Beliefs, Goals, and Utilities

The PUG architecture's most basic mechanism is conceptual inference, which uses knowledge to generate beliefs and goals from primitive objects and their descriptions. The inference process operates in two stages, with the results from the first feeding into the second. The initial phase uses

Table 4. Some beliefs from the robot mission domain about (a) primitive objects provided to the system, (b) virtual objects that are compositions of primitive ones, and (c) specializations of the composed relations.

```
(a) (robot ^id r1 ^xloc 0.0 ^yloc 0.0 ^orient 180.0 ^fuel 1.0)
    (object ^id t1 ^type target ^priority 1.0 ^xloc 2.0 ^yloc 0.0)
    (object ^id t2 ^type target ^priority 2.0 ^xloc -2.0 ^yloc 0.0))
    (object ^id d1 ^type danger ^xloc -1.0 ^yloc 1.0)
(b) (vector ^id (r1 t1) ^from r1 ^to t1 ^distance 2.0 ^angle -180.0)
    (vector ^id (r1 t2) ^from r1 ^to t2 ^distance 2.0 ^angle 0.0)
    (vector ^id (r1 d1) ^from r1 ^to d1 ^distance 1.41 ^angle -45.0)
(c) (to-right ^id (r1 t1) ^from r1 ^to t1 ^distance 2.0 ^angle -180.0)
    (ahead ^id (r1 t2) ^from r1 ^to t2 ^distance 2.0 ^angle 0.0)
    (to-right ^id (r1 d1) ^from r1 ^to d1 ^distance 1.41 ^angle -45.0)
```

compositional rules to generate virtual objects that describe relations among primitive ones. The module matches all such rules against percepts, using those with satisfied conditions to generate new beliefs that relate two or more objects. For example, the *vector* rule in Table 1 would match against the robot and target percepts *r1* and *t1*, shown in Table 4 (a), to produce the first *vector* belief in Table 4 (b). This involves calculation of new attributes for the relation, such as the *distance* and *angle* between the two objects. The compositional inference mechanism typically generates a number of such virtual objects, distinguishing them by their *^id* field, which is simply a list of the component objects. The current system makes a single pass over the percepts, although multiple passes would be needed to infer multi-level and recursively defined objects, such as towers.

The second stage of conceptual inference uses specialization rules to generate more restricted descriptions of objects and relations. This mechanism involves a number of passes, each one finding all matches of specialization rules, adding their instantiated heads as beliefs, and repeating the process until quiescence. For example, the *ahead* rule in Table 1 would match against the second instance of the *vector* relation in Table 4 (b), because its conditions are satisfied, but the *at-ahead* rule would not because robot *r1* is not at target *t1*. The effect is similar to sorting a stimulus description through a discrimination network.

Once the system has finished inferring beliefs on a given cycle, it uses knowledge to generate goal instances using a similar mechanism, but with an important difference. Recall that, in addition to activation conditions, each goal rule has an associated function. The module substitutes the values of variables bound in the conditions into this function and evaluates it to compute the inferred goal's utility. Because the function can match against numeric values of objects' attributes, a goal's utility can change with the agent's beliefs about the environment. To calculate the overall utility of a belief state, PUG finds all positive goals that are satisfied and sums their contributions. The system then finds all negative goals that match current beliefs and subtracts their values. If any goal is satisfied in multiple ways, each match contributes its amount to the overall utility. The architecture distinguishes between *achievement* goals, which contribute utility only when they are first satisfied, and *maintenance* goals, which contribute on every cycle.

The current implementation makes two simplifying assumptions. The first is that no compositional rules have conditions that match against the results of specialization rules, which lets the module terminate after the second phase is complete. Future versions should revisit the compositional rules to see if specializations cause them to match and, in turn, lead to more specializations. Second, the system infers beliefs and goals anew on each cognitive cycle. Primitive beliefs persist across cycles unless modified by operators, but composite and specialized beliefs, as well as goals, are rederived each time. A more efficient scheme would reduce computation by storing dependencies and revising only beliefs or goals whose support changes, as in truth maintenance systems.

3.4 Problem Solving and Search

Now we can turn to PUG's mechanisms for generating plans that address the agent's goals. The details of this module are not essential to the theoretical claims that we made earlier, but we describe its operation for the sake of completeness. Briefly, the problem solver carries out heuristic search through an OR space, retaining the nodes it generates in the search tree described earlier. Recall that each node in this search tree specifies a partial plan, with children elaborating on the plan in their parent. For this reason, it is best described as adopting a 'plan-space' approach (Kambhampati, 1997) even though its reliance on forward chaining is usually associated with 'state-space' planners (Hoffmann & Nebel, 2001).

The problem-solving module operates in discrete cognitive cycles, each of which takes one or more meta-level actions that advance the search process. PUG considers these conditional actions in a particular order, much as in early production systems:

- If there are enough acceptable plans or no further choices are available, then halt and return the candidates ranked according to their utilities.
- If the plan associated with the current node is acceptable, then store this plan with its utility and select another node in the search tree for elaboration.
- If the current plan is unacceptable (e.g., involves a repeated qualitative state with lower utility or has no untried operators), then mark it as failed and select another node in the search tree.
- If the current plan has no associated operator instances, then find which operators have conditions that match against the current state and generate evaluation scores for each matched instance.
- If the current plan has untried operator instances but none selected, then select the untried alternative with the best evaluation score and apply it to generate a new state and subproblem.

We have described some activities, such as selecting nodes, in abstract terms because different settings in PUG can produce different search strategies. For instance, selecting a node's parent on failure produces heuristic depth-first search, whereas selecting the root leads to a systematic variant of iterative sampling (Langley, 1992). These offer the problem solver considerable flexibility, as in Soar (Laird, 2012) and FPS (Langley et al., 2014), but they are not our focus here.

However, some of the problem solver's parameters relate directly to the theoretical postulates that we presented earlier. In particular:

- When selecting among operator instances, the system uses their average utility as determined by active goals that match over the course of their simulated application.

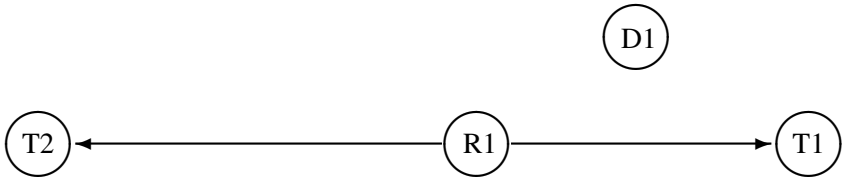

Figure 1. A simple scenario in which the robotic agent *R1* must select between two target sites to deliver a single sensor (not shown). Site *T1* is closer, letting the agent achieve its delivery goal sooner, but it would also require the robot to pass near danger area *D1*, so that slower delivery to *T2* gives higher utility.

- To obtain this average, PUG simulates each operator's influence on the environment over time, comparing the resulting beliefs to goals on each time step and summing the utilities of matched elements. If a simulation halts before producing its expected results, then the operator fails.
- When determining whether a plan is acceptable, the architecture compares the average utility of its sequence of operators to the utilities for its ancestors in the search tree. Because the empty plan at the root node has zero utility, this means that it rejects any plan with a negative score.
- To rank the alternative plans it generates, PUG sorts acceptable candidates by their average utilities over the entire sequence of their simulated trajectories.

One can certainly envision other planning systems that differ in the operational details yet still incorporate these key assumptions. We maintain that these latter features are crucial to dealing successfully with the types of scenarios we outlined earlier, but, to clarify the ideas, we must describe the role of mental simulation in calculating expected utilities.

3.5 Mental Simulation and Utility Projection

The PUG architecture calculates the utility for a candidate plan by summing the average utilities of its component operator instances weighted by their durations. To determine the latter two quantities, the system simulates each operator instance from an initial state by applying it repeatedly, much as when simulating difference equations. This involves, on each time step, matching the operator's *:elements*, *:conditions*, and *:tests* fields against the current state, substituting any bound variables into contents from its *:changes* field, and updating the attribute values of these structures. This process continues until the simulation reaches a state that matches the relations included in the operator's *:results* field or until its conditions no longer match, in which case it fails.

On each time step, the simulation module invokes the inference process to derive beliefs about the current situation, generate relevant goals, and calculate the utilities for these goals. For each positive goal *P* that is satisfied by current beliefs, the system adds *P*'s utility to the total. For each negated goal *N* that is satisfied, it subtracts *N*'s utility from the total. The beliefs, the active goals, and the associated utilities can vary over the course of a simulation, but typically the first two remain qualitatively the same, with only quantitative atttributes and utilities changing. The system also tracks the number of time steps so that, if the simulation reaches the expected results, it divides the total by this count to return the average utility along with the final state. Unlike the higher-level search process, simulation is deterministic and runs in time linear with the number of steps, the number of objects updated, and the number of goals matched.

Consider the simple scenario in Figure 1, which involves a robot, *R1*, two target sites, *T1* and *T2*, one danger area, *D1*, and a single sensor, *S1*, that the robot is carrying but is not shown. Suppose the system has generated two sensor-related goals – *(sensor-at ?s T1)* and *(sensor-at ?s T2)* – each with the constant utility 100 and one danger-related goal – *(not (vector ^id (R1 D1)))* – whose utility varies inversely with the distance squared. Because there is only one sensor, the first two goals are mutually exclusive and the agent must choose between them. If the robot initially faces north, then the time required to turn toward each target is the same, but *T1* is closer, so traversal time will be less, letting the system achieve *(sensor-at ?s T1)* more rapidly than *(sensor-at ?s T2)*, giving higher average utility. However, this advantage is offset because moving to *T1* brings *R1* much closer to the danger area *D1*, so that moving to *T2* and depositing *S1* there gives a higher overall score.

This example clarifies how multiple goals can influence the utility calculated for individual operator instances, and the theme recurs on the larger scale of multi-step plans. Whenever PUG extends a plan by adding an operator, it generates a new node in the search tree and calculates its average utility as a weighted average of its parent's utility and the operator's score. Average utility seems like a reasonable metric for plan quality, although one might also introduce a discounting scheme, as in work on reinforcement learning. The system currently supports only depth-first search and iterative sampling, but a best-first variant could use plan utility in node selection. More important, plan utility provides PUG with a termination criterion – when a plan has no children (extensions) with higher average utility and no ancestors with better scores. We will see that this has important implications for the system's decision making.

4. Empirical Demonstrations

To provide evidence that PUG supports the target abilities described earlier in the paper, we designed a number of mission scenarios that involve a robot delivering sensors to target sites in the presence of danger areas. In each case, the robot had access to the same knowledge about concepts, goals, and activities. The latter including two operators for turning left or right to face an object, another operator for moving toward an object, and actions for refueling at a depot, depositing a sensor at a target, and collecting a sample. The first four operators are durative in character and thus require extended simulation, whereas the final two take only one time step. We instructed PUG to return multiple acceptable plans for each mission, which it ranked according to their average utilities.

We will summarize the system's behavior on ten classes of problems that share goal rules but that differ in the number and layout of objects. Figure 2 depicts the basic version of each scenario in terms of the initial state. For most runs, we gave PUG a depth limit of eight and asked it to return up to eight plans. The exception was the final, more complex, problem, which required search to 20 levels deep. None of these scenarios are very complicated, but together they demonstrate the range of abilities that the system supports.

- *Scenario 1* included one sensor and two target sites, T1 and T2, with T1 nearer to the robot than T2. In this case, PUG found two plans to deliver the sensor that involve three steps each (turning to face a site, moving to the site, and depositing a sensor). The solution that involved delivery to T1 had higher average utility. This runs demonstrates the system's basic ability to find multiple plans and to rank them by utilities.

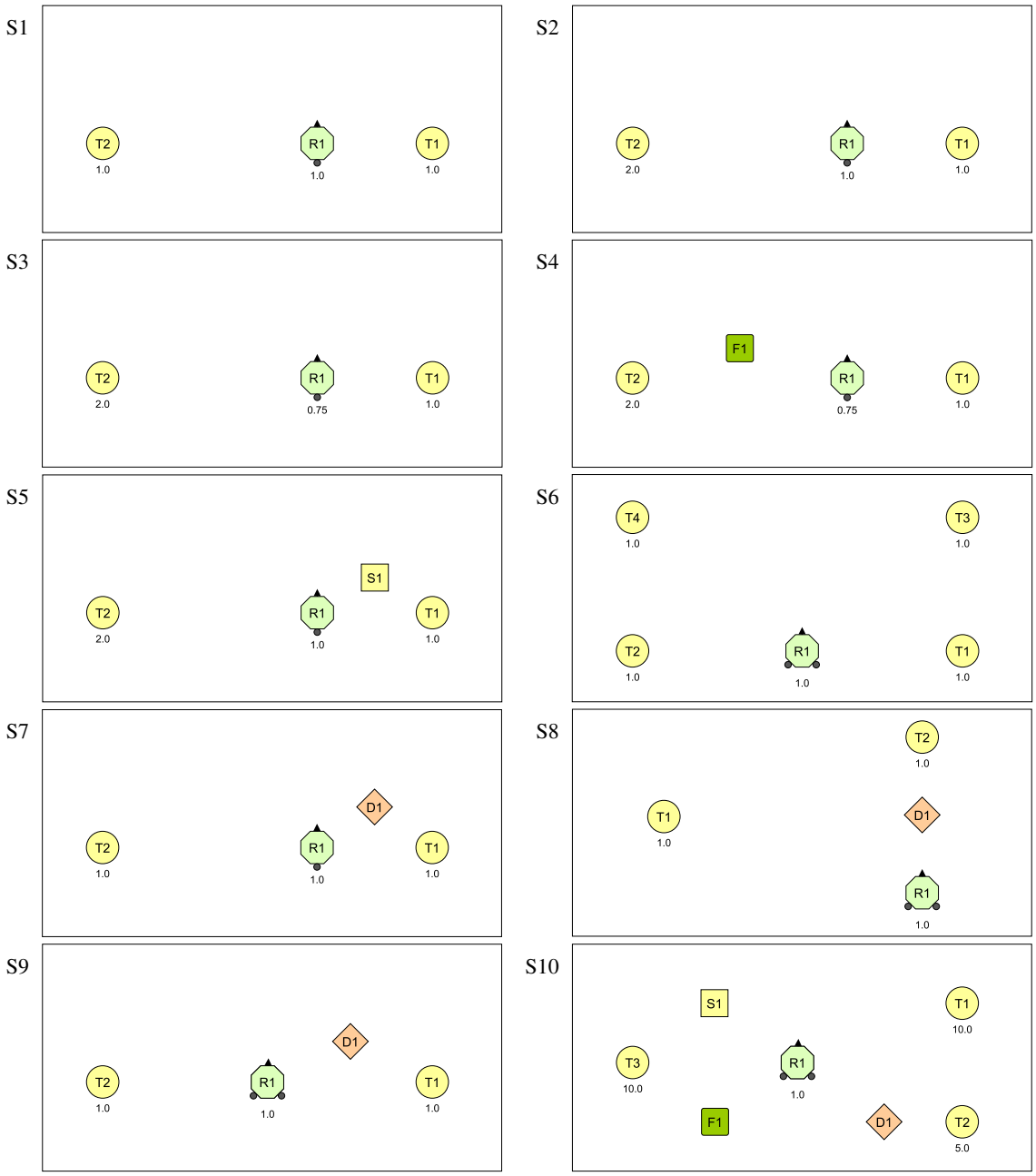

Figure 2. Ten scenarios that demonstrate different facets of the PUG architecture's ability to reason about goals and utilities during planning in continuous domains. Objects that start with T denote a target site, D a danger area, and S a sample. The number below the robot (R1) is the fuel level at the outset, the arrow indicates its initial orientation, and the attached dots are sensors it carries. Each target site has an associated priority. Scenario numbers correspond to those in the text, with some scenarios differing in their scales.

- *Scenario 2* involved one sensor and two target sites, T1 and T2, with T1 nearer to the robot but with T2 having a higher priority. As before, PUG found two plans to deliver the sensor, but this time the solution with delivery to T2 had higher utility. Altering distance or priority sufficiently changes this ordering, showing the system can trade off multiple factors in its utility calculations.
- *Scenario 3* included one sensor and two target sites, T1 and T2, with T1 nearer to the robot but with T2 having a higher priority. PUG attempted to generate a delivery plan for T2, but this ran out of fuel along the way, so it found only one plan, which delivered the sensor to T1. This demonstrates that the system's use of mental simulation lets it take limited resources into account.
- *Scenario 4* involved one sensor and two target sites, T1 and T2, with T1 nearer to the robot but with T2 having a higher priority but also too distant to reach with available fuel. Here the system found a high-utility plan that delivered the sensor to T2, but only after diverting to a fuel depot along the way. It also found a delivery plan to T1 with lower average utility. This shows that PUG not only reasons about resource limitations, but find ways to overcome them.
- *Scenario 5* included one sensor and two target sites, T1 and T2, with T1 nearer to the robot but with T2 having a higher priority but with a desirable sample near the path to T1. In this case, PUG found six distinct plans, the two best involving collection of the sample followed by delivery of the sensor to T1. A plan for delivering the sensor directly to T2 came in third, with others scoring much worse. This demonstrates that the system supports simple forms of opportunistic planning.
- *Scenario 6* involved two sensors and four target sites, two on the robot's left and two on its right. In this case, the highest ranking plans delivered sensors to the two leftmost objects or the two rigthmost objects. PUG found other plans that required longer traversal and had lower utilities, although depth-first search revealed some of these solutions before finding the better ones.
- *Scenario 7* included one sensor and two target sites, one nearer to the robot than the other but also with a danger area near its path. We have already seen this task in Figure 1. Here PUG found two plans to deliver the sensor, with traversal to the safer but more distant target having higher utility. Moving the danger area further away caused this ordering to reverse. This scenario demonstrates the ability to trade off positive and negative factors when making decisions.
- *Scenario 8* involved two sensors, two target sites, and one danger area, with the direct path to the nearest target passing very close to the danger area. PUG instead found two plans that visit the more distant target first, deposit one sensor or the other, and then visit the other target, effectively making a detour around the danger area. The system did not return any plan that visits the nearest target first, as they all have negative utility.
- *Scenario 9* included two sensors, two target sites, and one danger area that lay very near the path to one of the targets. In this case, PUG decided that it was better off delivering only one of the sensors, as taking the second to another target would reduce overall utility below zero, less than that of taking no further action. More distant danger areas led to plans that delivered both sensors. This demonstrates the ability to avoid action entirely when it would be counterproductive.
- *Scenario 10* combines the earlier factors into one problem, incorporating two sensors, three targets, one sample, one fuel depot, and one danger area. PUG finds an intuitive plan that collects a sample, refuels, delivers a sensor to the near target, refuels again, and delivers a sensor to the far target, avoiding the site near the danger area. However, the system ranked this plan third after simpler candidates that collect a sample but deliver only one sensor without needing to refuel.

These runs are generally consistent with our aims for PUG's behavior. The architecture can generate multi-step plans that take into account inconsistent goals, such as wanting to place a sensor at two sites when only one is available. The system can also calculate the utilities of operators and combine them into scores for entire plans. The different rankings it produces for distinct configurations and positions of objects show that its utilities are sensitive to the situation and that it can combine these scores to reason about tradeoffs. The system requires different amounts of search on these tasks, with the easiest generating only 12 nodes and the hardest examining some 110 candidate plans, but we are concerned here more with plan evaluation than guidance through the search space.

Mental simulation of operators underlies PUG's calculation of utilities, even though our description of runs has not emphasized that fact. We will not argue that simulation of durative operators is strictly essential to calculating average utilities over time, as one might imagine forms of geometric reasoning that produce the same results analytically, say by integrating over the triangular area bounded by the robot, a target site, and a danger area. However, this would not extend easily to situations with multiple danger areas, and quantitative simulation of operators is a general technique that can handle any number of goals. The arguments in favor of goal-based utilities are even stronger. One might provide a standard optimization technique with an objective function equivalent to our distributed utility metric, then use it to find high-quality plans. But the objective criterion would encode a particular number of objects for each type, rather than generating new goals with associated utilities as the need arises. Our framework's combination of goals with utilities, and their summation over the course of plans, offers more flexible, adaptive support for autonomous agents.

5. Related Research

Our approach to problem solving and planning incorporates ideas from many predecessors. These include cognitive architectures, especially Langley et al.'s (2009) ICARUS, which also assumes separate structures and processes for concepts and activities. PUG also draws on Nilsson's (1994) notion of durative operators, as well as standard methods for making logical inferences and carrying out heuristic search for multi-step plans. Building on such earlier traditions is common in cognitive systems research, as it often involves the development of integrated computational artifacts. The deeper issue revolves around the originality of our theoretical postulates, so let us consider them in turn and examine prior work related to each one.

Two central claims are that *goals are the locus of utility*, with the latter being distributed among goals as numeric annotations, and that such goal-oriented utilities *guide the planning process*. Some cognitive architectures, such as Soar (Laird, 2012) and Prodigy (Carbonell, Knoblock, & Minton, 1990) encode knowledge about goal importance, but these take the form of symbolic preference rules rather than as quantitative scores, and Wilensky's (1983) seminal work on reasoning about goal conflicts was primarily qualitative. ACT-R (Anderson & Lebiere, 1998) associates numeric strengths with production rules and activation levels with working-memory elements, but it makes no architectural distinction between goals and beliefs. The older behaviorist literature emphasized strengths on stimulus-response pairs to characterize agent motivations, but rejected internal mental structures like goals. Thus, our approach combines ideas from a number of psychological traditions in an effort to cover a broader range of qualitative behaviors. Our approach comes closest to Benton et al.'s (2009) technique for partial satisfaction planning, which associate weights with symbolic

goals to handle competing objectives. They attach costs to actions rather than to negated goals, and weights do not vary across situations, but the two frameworks adopt very similar halting criteria.

Another theoretical tenet is that both *goals and utilities are conditional*, with the former becoming active only when symbolic conditions are satisfied and the latter varying as numeric functions of the situation. The *goal reasoning* paradigm has explored techniques for creating new goals appropriate to the agent's situation, with examples including Choi (2011), Hanheide et al. (2010), and Talamadupula et al. (2010). Choi's architecture not only generates goals but alters their priorities based on how closely the state approximates continuous prototypes, which is less flexible than our utility functions but has similar effects. More recently, Roberts et al. (2015) report an integrated system for multi-agent plan execution, monitoring, and repair that activates, deactivates, and refines goals in a sophisticated life cycle. Our architecture differs in many details, including its focus on plan generation rather than execution, but the two frameworks share high-level features.

The final postulates are that – for agents in continuous physical domains – *mental structures include both symbols and numbers* and *calculation of utility relies on mental simulation*. There has been work on planning with limited resources, but they have typically modeled resource consumption as numeric effects on discrete operators. In some sense, all planning systems rely on simulation, but this typically involves qualitative reasoning about the effects of symbolic operators. Our framework uses simulation to predict how numeric attributes, which influence utility on each time step, change over the course of durative operators' application. Laborie (2014) presents combined approaches to planning and scheduling that reason over time about symbolic actions and continuous resources, but his constraint-based methods do not draw on intra-operator simulation to reveal complex side effects. In summary, each of our theory's assumptions has appeared in the literature, but earlier techniques differ in many details and, more important, they have not been combined into a unified account of behavior in physical domains with dynamic goals and utilities.

6. Directions for Future Work

Despite the progress we have reported, our research program would benefit from additional work. We should demonstrate the viability of the framework more generally, both in different planning domains that combine symbolic relations with numeric attributes and on other problem-solving strategies that the flexible architecture supports.

We should also extend the theory, and its implementation in PUG, to address the challenge of scaling to complex tasks. One natural approach is to incorporate domain knowledge about ways to decompose problems into subproblems. Hierarchical task networks use this idea to constrain the search for viable plans, but they lack the ability to fall back on planning with primitive operators. Li et al. (2012) and To et al. (2015) have reported systems that combine the two paradigms, but they did not emphasize their application to continuous domains. Of course, humans also respond to complexity by *satisficing* (Simon, 1956), so we should also explore variants that consider a plan to be acceptable when its utility score exceeds a threshold. This maps well onto the notion of aspiration level, which is central to Simon's account of human decision making. We should also introduce heuristics that focus the architecture's attention on a subset of objects. PUG currently utilizes conditions in inference rules for this purpose, but we can replace this all-or-none scheme with one that calculates object 'salience' in terms of relevance to goal utilities.

On another front, we should explore alternative ways to estimate utility and run experiments that examine their effect on behavior. We should demonstrate the problems that arise when one calculates state utilities only before and after operator application. We have already mentioned using geometric reasoning over areas bounded by the robot, a target site, and a danger area, which offers an option to mental simulation for computing utility at the operator level. We should also explore variants on utility that are discounted over time, as in work on reinforcement learning. This would give different results than average utility at individual operators, but it would be more important at the level of entire plans. In addition, we should extend the theory and architecture to support operators with stochastic outcomes. This could require more expensive repeated simulations, but it would bring our notion of utility into closer alignment with that in decision theory.

Moreover, autonomous agents require more than the ability to generate plans, and we should augment PUG to execute them in simulated or physical environments, and even to interleave plan generation with execution to handle divergences between simulated and actual events. The execution module would operate over the same mental structures as used during plan creation, comparing expected and osbserved utility to determine when behavior has diverged enough from the current plan to revise it. Some domains also benefit from parallel execution of operators, such as moving forward and turning to produce curvilinear motion and even movement around obstacles. Potential fields offer a natural way to combine numeric utilities with multiple effectors to produce smooth behavior in continuous settings. Together, these extensions should produce a more complete theory of physical behavior that is guided by goals, utilities, and mental simulation.

7. Concluding Remarks

In this paper, we presented a new theory of planning in physical settings that combines ideas from distinct traditions. The framework posits that symbolic goals are the locus of numeric utility, with a state's total score based on all matched goals, and that these component utilities vary with the agent's beliefs about its situation. The theory also claims that mental structures, whether concepts, operators, beliefs, or goals, include both relational structure and numeric attributes. These assumptions suggest a new approach to planning that relies on quantitative mental simulation to generate state trajectories, which in turn produce utilities that guide search for solutions.

We also reported an implemented problem-solving architecture, PUG, that embodies these theoretical tenets. We provided examples of the system's formalism for stable structures, like concepts and operators, and more dynamic ones, like beliefs, goals, and plans. We explained its mechanisms for generating the latter from the former using conceptual inference and heuristic search. In addition, we illustrated PUG's operation on a number of scenarios that involved competing goals and tradeoffs among different sources of utility. Finally, we discussed our intellectual debts to earlier work on planning, goal reasoning, and utilities, along with likely directions for future research that would elaborate on the promising cognitive synthesis that our theory embodies.

Acknowledgements

This research was supported by Grants N00014-12-1-0143 and N00014-15-1-2517 from the Office of Naval Research, which is not responsible for its contents. We thank J. Benton, Chris Pearce, Stephanie Sage, Charlotte Worsfold, and for useful discussions about the ideas in this paper.

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

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

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.

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought*. New York: McGraw-Hill.

Hanheide, M., Hawes, N., Wyatt, J., Gobelbecker, M., Brenner, M., Sjoo, K., Aydemir, A., Jensfelt, P., Zender, H., & Kruijff, G. M. (2010). A framework for goal generation and management. *Proceedings of the AAAI Workshop on Goal-Directed Autonomy*. Atlanta: AAAI Press.

Hoffmann, J. (2001). FF: The Fast-Forward planning system. *AI Magazine*, *22*, 57–62.

Kambhampati, S. (1997). Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, *18*, 67–97.

Laborie, P. (2014). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results (pp. 143–149). *Proceedings of the Sixth European Conference on Planning*.

Langley, P. (1992). Systematic and nonsystematic search strategies. *Proceedings of the First International Conference on AI Planning Systems* (pp. 145–152). San Francisco: Morgan Kaufmann.

Langley, P., Choi, D., & Rogers, S. (2009). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, *10*, 316–332.

Li, N., Stracuzzi, D. J., & Langley, P. (2012). Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems*, *1*, 109–126.

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

Roberts, M., Vattam, S., Alford, R., Auslander, B., Apker, T., Johnson, B., & Aha, D. W. (2015b). Goal reasoning to coordinate robotic teams for disaster relief. *Planning and Robotics: Papers from the ICAPS Workshop*. Jerusalem: AAAI Press.

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.

To, S. T., Langley, P., & Choi, D. (2015). A unified framework for knowledge-lean and knowledge-rich planning. *Proceedings of the Third Annual Conference on Cognitive Systems*. Atlanta, GA.

Wilensky, R. (1983). *Planning and understanding: A computational approach to human reasoning*. Reading, MA: Addison-Wesley Publishing.