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

Dongkyu Choi
Department of Aerospace Engineering
University of Kansas, Lawrence, KS

Pat Langley, Mike Barley, Ben Meadows
Department of Computer Science
University of Auckland, Auckland, NZ

Edward P. Katz
College of Computer and Information Sciences
Northeastern University, San Jose, CA

This research was supported by Grant N00014-15-1-2517. We thank J. Benton, Chris Pearce, Stephanie Sage, Mike Stilman, and Son To for helpful discussions.

A Motivating Example

Consider a planetary mission in which an autonomous robotic agent must:

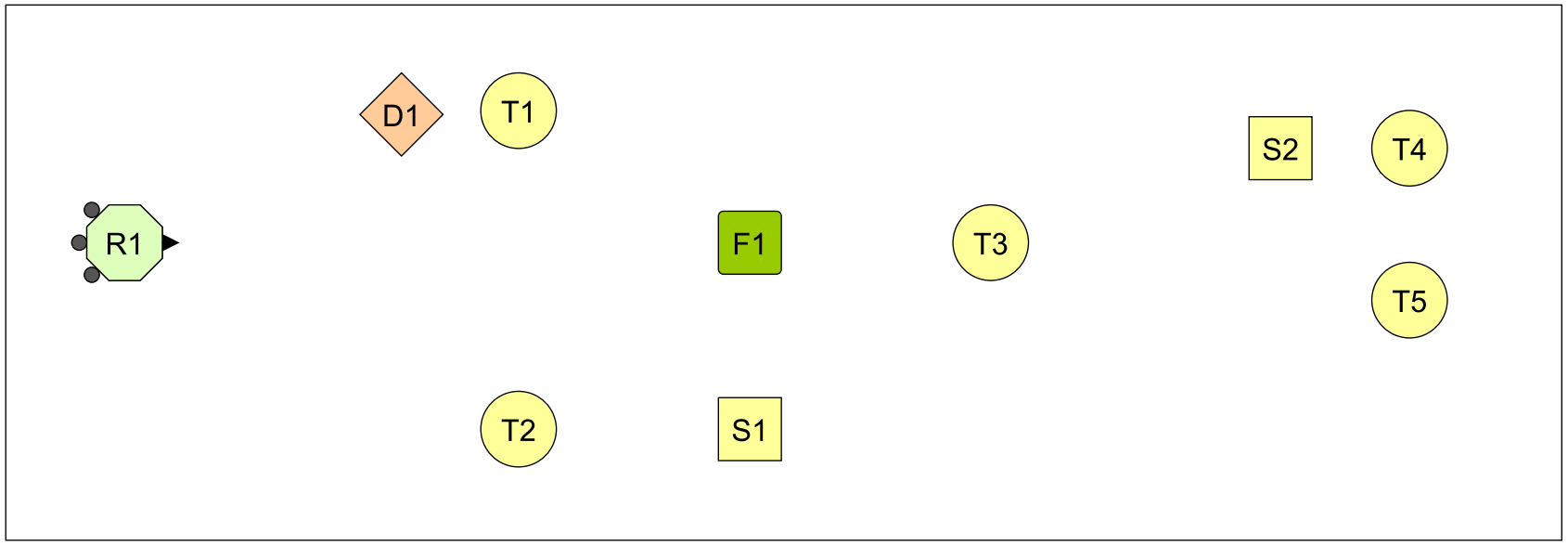
- Deposit sensors at prioritized target sites;
- Collect interesting samples that it encounters;
- Avoid the proximity of known danger areas; and
- Retain enough onboard fuel to carry out these tasks.

A mission will involve many competing goals, some mutually exclusive, that may have different value at different times.

The agent not only generate plans, but also monitor execution, detect problems, and replan in response.

A Motivating Example

Consider a scenario that includes three sensors, five target sites, one fuel depot, one sample, and one danger area.



Here an effective plan might deliver a sensor to T2, collect S1, refuel at F1, deliver a sensor to T3, collect S2, and deliver a sensor to T5. The agent would bypass T1, as it is near D1.

Target Abilities

We desire intelligent agents that reason about and plan their activities in continuous domains by:

- Operating over space and time in contexts with *competing* and even *inconsistent* objectives;
- Assigning different *importance* or *value* to objectives based on the agent's *situation*;
- Reasoning about activities that involve change in *qualitative* structure and *quantitative* attributes; and
- Producing reasonable behavior that *balances tradeoffs* among different objectives in a situation-aware manner.

These abilities are crucial in domains that require autonomy, from planetary exploration to disaster relief.

Review of the PUG Architecture

Last year, we reported on a theory of planning in continuous domains with conflicting goals that adopted four 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.

We also described PUG, a new architecture that incorporates these theoretical ideas.

Knowledge in the PUG Architecture

PUG incorporates four distinct types of generic knowledge:

- *Compositional rules* that define relational concepts and their associated numeric attributes;
- *Specialization rules* that discriminate among subclasses of more general concepts;
- *Goal-generating rules* that specify when goal instances should be active and what utility to assign them; and
- *Operators* that encode models of actions' immediate effects and final results under given conditions.

Together, these provide the content PUG uses for conceptual inference, goal creation, and plan generation.

Examples of PUG Conceptual Rules

Compositional rule:

```
((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)))
```

Specialization rules:

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

((at-ahead ^id (?r ?o))
   :specializes  (at ^id (?r ?o) ^angle ?a)
   :test         ((> ?a −0.01) (< ?a 0.01)))
```

Examples of PUG Goal-Utility Rules

Achievement rule:

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

Maintenance rule:

```
((not (vector ^id (?r ?o) ^from ?r ^to ?o))
   :type          maintenance
   :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)))
```

Examples of PUG Operators

```
((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))))))
```

Examples of PUG Beliefs

Primitive objects:

(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)

Composite relations:

(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)

Specialized relations:

(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)

Planning in the PUG Architecture

PUG carries out forward heuristic search through a space of partial plans by:

- Finding each operator O that is applicable in the current state;
- Using quantitative simulation to predict O's trajectory over time;
- Drawing inferences about qualitative relations on each time step;
- Determining active goals and associated values on each time step;
- Calculating utility of partial plans based on their matched goals.

The architecture uses average utility to guide heuristic depth-first search; the resulting plans may not satisfy all goals.

Parameters include maximum length, nodes to be considered, and number of solutions desired.

An Extended Theory

We have extended the prior theory to include four additional assumptions:

- Execution begins when planning finds an acceptable operator sequence using resources available for heuristic search.
- Plan monitoring compares quantitative mental simulations of expected states with sensing of actual states to track progress.
- Operator anomalies – when conditions, utilities, or results of operators differ from agent expectations – lead to replanning.
- Goal anomalies – when goals generated from observed states differ from expected ones – also lead to replanning.

We have incorporated these new postulates into PUG/X, an architecture that extends the previous one.

Execution and Monitoring in PUG/X

When executing a durative operator, like (move-to R1 T1), the architecture:

- Uses quantitative mental simulation, conceptual inference, and goal reasoning to generate expectations for each time step.
- Compares the expected and actual conditions, results, utilities, and goals on each step;
- Upon detecting anomalies – disagreements between the expected and observed situations – replans from the current state.

Alternation between plan generation and execution continues until a plan succeeds or no acceptable plan emerges.

Mental simulation is *deterministic* and so adds only a constant factor to computation costs.

Simplifying Assumptions

The current implementation of PUG/X makes six convenient assumptions:

- The agent can execute only one operator at a time;
- Operators have deterministic effects on the environment;
- Operator descriptions are usually (but not always) accurate;
- Environmental changes are typically due to agent actions;
- Once it perceives an object O, it has complete information on O;
- Execution halts while the agent is generating a new plan.

The assumptions are not central to the theoretical framework.

Empirical Demonstrations

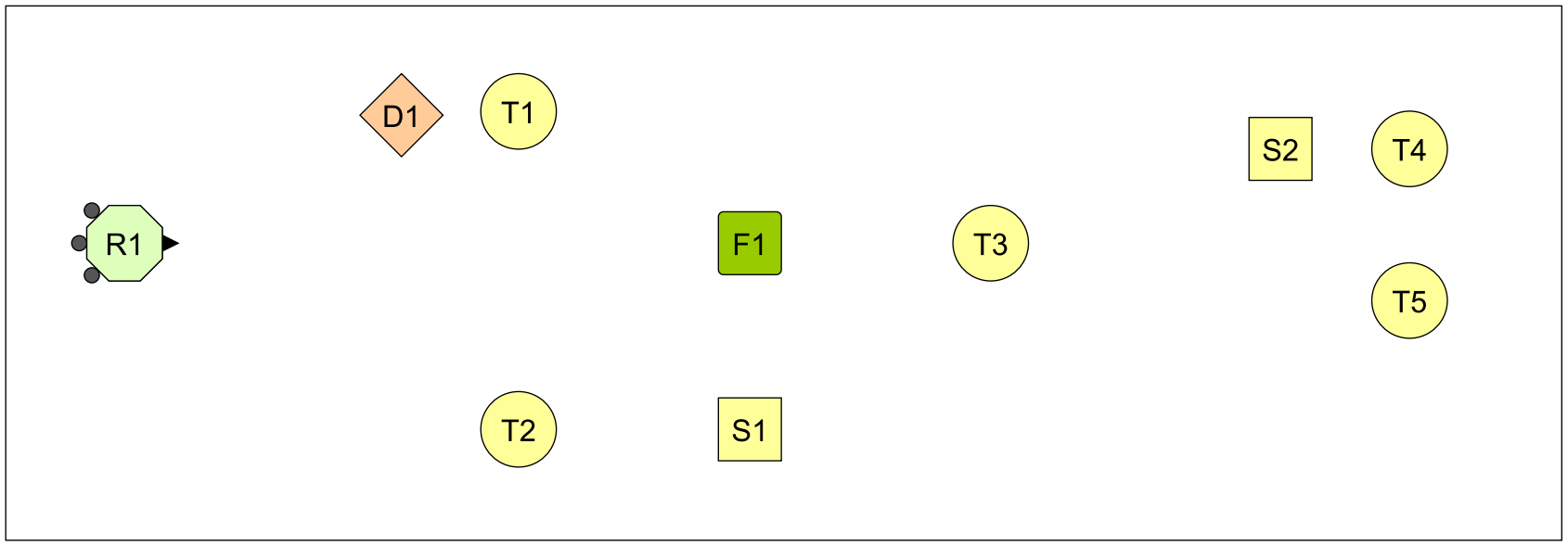
We demonstrated PUG/X's operation on scenarios that involve:

- Nominal plan execution in which no surprises arise
- Anomalous operator conditions
  - Robot veered slightly to right, samples moved on their own
- Anomalous operator termination
  - Robot advanced more slowly than expected
- Anomalous utilities
  - Increased radiation from danger area
- Unanticipated goals
  - Newly observed target sites, samples, danger areas

One scenario required extended operation: the agent continued to detect new (20) target sites and replan in response.

An Illustrative Scenario

Consider our motivating scenario with three sensors, five target sites, one fuel depot, one sample, and one danger area.



Here PUG/X builds an initial plan to deliver a sensor to T1, then shifts to T2 on detecting D1, extends it to include S1, decides to refuel on sensing T3, and finally adds S2 and T4.

Related Research / Contributions

Our research incorporates ideas a number of earlier efforts on plan generation, execution, and monitoring:

- Cognitive architectures (Soar, Prodigy/ROGUE, MIDCA)
- Three-tiered architectures and teleoreactive systems
- Integrated planning and execution (PRS, SIPE, CASPER)
- Goal reasoning (Talamadupula et al., 2010; Roberts et al. (2015)

The PUG planner is unique in its emphasis on quantitative simulation and goal-oriented utility.

The main contribution of the current work is extending it to support plan execution and monitoring.

Concluding Remarks

We have presented a theory of plan generation, execution, and monitoring in physical settings that posits:

- Symbolic goals are the distributed loci of numeric utility;
- Planning *and* monitoring rely on quantitative mental simulation;
- Replanning occurs when monitoring detects anomalies;
- Anomalies may involve operator conditions, results, or utilities;
- They may also involve unexpected introduction of goals.

We have incorporated these ideas into the PUG/X architecture.

We demonstrated its operation on scenarios with unexpected events, including one that required extended operation.

End of Presentation