Self-Confidence in Autonomous Systems
Papers from the AAAI 2015 Fall Symposium

Computational Mechanisms to Support Reporting of Self Confidence of Automated/Autonomous Systems

Ugur Kuter and **Chris Miller**
Smart Information-Flow Technologies, LLC
319 1st Ave N., Suite 400,
Minneapolis, MN 55401-1689
{ukuter,cmiller}@sift.net

Abstract

This paper describes a new candidate method of computing autonomous "self confidence." We describe how to analyze a plan for possible but unexpected break down cases and how to adapt the plan to circumvent those conditions. We view the result plan as more stable than the original one. The ability of achieving such plan stability is the core of how we propose to compute a system's self confidence in its decisions and plans. This paper summarizes this approach and presents a preliminary evaluation that shows our approach is promising.

Introduction

Autonomous agents, whether embedded in a robot, managing an industrial process or advising us on stock purchase decisions, are becoming increasingly prevalent, and increasingly competent. As such, we are under increasing pressure to rely on their capabilities — that is, to grant them an increased space of autonomy, or sphere of action in which their behaviors are under their own control or their recommendations are accepted with little or no inspection. This, when it works, leads to increased efficiency. Of course, the problem is knowing when to trust the agent, and when not, without controlling the robot or analyzing the decision oneself.

The situation is analogous, in some ways, to the position a human supervisor finds himself or herself in when deciding whether and how to trust a subordinate (Hutchins et al. 2015). The human pair, however, has a cue that is missing from most human-machine interactions: expressions and assessments of self confidence.

We view self confidence as growing from a number of different sources[1] but in the interaction between a supervisor and subordinate, the self confidence of the subordinate has the role of providing a short hand abbreviation of how much the subordinate believes s/he should be trusted. Detailed explanations about the sources and threats to confidence are, undoubtedly, a better and richer source of the same information, but they are also more time consuming to gather, understand and process. In lieu of that detailed understanding, a shorthand indicator can provide substantial added value — a point Lee and See make about etiquette cues in communication in their seminal paper defining and illustrating different trust mechanisms in human-machine interaction (Lee and See 2004). It serves as a simple heuristic (Gigerenzer and Todd 1999) for a receiver to rapidly assess whether or not (or to what degree) to accept and trust a recommendation or behavior. And while an actor's self confidence may well be erroneous, that tendency itself can be rapidly learned (arguably more rapidly than the actor?s actual trustworthiness across situations).

So, there is reason to believe that if an automated agent could compute and report its "self confidence" in a fashion similar to a human actor, this would be a valuable addition to human-automation interaction. But, before we can test that hypothesis, we need a method for computing machine self confidence across contexts and problems. That was the focus of our work, as reported here.

Plan Stability as a Method to Compute Self Confidence

Brittle plans may fail catastrophically, causing mission losses but even minor failing in "coverage" will result in user trust loss. We view a system's self confidence as the system's ability to be self-aware of the following: even if the system's plan is probabilistically the best, there are conditions that could arise that its plan will not be good for and the plans may may break down in such foreseeable, but less likely, conditions. We will refer to these conditions as *contingencies* in this paper. Our notion of the stability of a plan depends on (1) how we can identify where plans are threatened by contingencies and (2) to what extent we can adapt the plan to circumvent them.

For a system to self-evaluate its self confidence, we take the following approach: we produce counterexamples to a plan — that is, contingency instances — to provide decision support to both human and automated planners to make plans robust to such contingencies. We assume that the system will receive observations about environment changes, adversarial events, changing objectives and task assignments. We describe our *counter-planning* technique (Goldman, Kuter, and Schneider 2012), called Murphy, that recognizes plan faults if these world states violate one or more safety constraints known by the system. After recognizing

[1] See companion article by Ahmed & Sweet for model and definitions

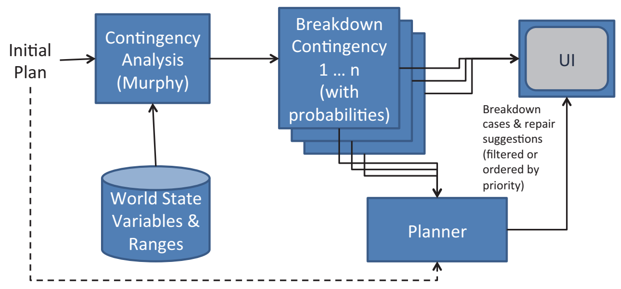
Initial Plan
Contingency Analysis (Murphy)
World State Variables & Ranges
Breakdown Contingency 1 … n (with probabilities)
UI
Breakdown cases & repair suggestions (filtered or ordered by priority)
Planner

Figure 1: A high-level function view of our computational approach to measure the self confidence of a system and its user interactions.

a fault, our approach uses hierarchical re-pair techniques, based on our previous work on HOTRiDE (Ayan et al. 2007) and SHOPLifter (Kuter 2012) to localize and fix the error. Our hierarchical plan repair techniques walk up the task structure of a plan, trying small fixes first, then expanding. Once this generates contingency libraries, it transforms both counter-plans and world-state knowledge into operational behavior models for the users' review and for autonomy planners and managers as well. Figure 1 shows a high-level functional description of our computational approach. Below, we describe the major technical components in this architecture.

Murphy: Analyzing Contingencies in Plans

Murphy takes a representation of the initial plan and a planning domain definition consisting of a set of exogenous events (e.g., the actions of other agents, the actions of "nature," etc.). We then treat these components as a disjunctive planning problem and can use any AI planner to find counterexamples to illustrate ways in which a plan may go awry in execution.

Murphy translates a plan into what we call a "counter planning" problem, combining a representation of the initial plan with the definition of a set of exogenous events. In particular, we developed a translation that takes as input (1) a formal description of planning domain written in the widely-known PDDL (Planning Domain Description Language) in AI planning (McDermott 1998; Fox and Long 2001); (2) a PDDL planning problem; (3) a plan that solves the domain + problem; (4) a counter planning domain extension with new objects and planning operators for exogenous events. The translation yields a new counter planning problem and domain that can be solved to find a counterexample or, if searched exhaustively without finding a trace, indicates that no counterexample exists. There are three components to the translation process: (1) generating a "skeleton" for the original plan that ensures that the original plan actions occur in the proper order; (2) formulating a goal expression that will cause a planner to search for a counterexample and (3) encoding the resulting disjunctive goal expression in a way that is amenable to the use of a standard PDDL planner.

The details of our translation and a comprehensive experimental evaluation of Murphy is described in (Goldman, Kuter, and Schneider 2012). The soundness of our translation procedure follows immediately from its construction. Given access to a sound and complete classical planner, we have a sound and complete algorithm for deciding a counter planning problem as defined above.

Hierarchical Plan Repair

Once the possible contingencies on a plan are identified by Murphy, we proceed to generate ways to adapt the plan to alleviate those contingencies, should they happen. We believe that the ability to adapt the plan to all or most of the known or identified contingencies should boost the system's self confidence for two reasons: (1) adapting the plan to the identified contingencies makes the plan more robust, and therefore, increases the likelihood that the plan will succeed in execution; and (2) the system increases its belief that it may be able to adapt to any contingency that is learned or identified later.

We use a hierarchical planning and plan repair (HPR), based on our previous works on HOTRiDE and SHOPLifter systems. HPR is based on the well-known Hierarchical Task Network (HTN) planning paradigm (Erol, Hendler, and Nau 1994; Nau et al. 2003). The purpose of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of planning operators and methods, each of which is a prescription for how to decompose a task into its subtasks (smaller tasks). The description of a planning problem contains an initial state as in classical planning. Instead of a goal formula, however, there is a partially-ordered set of tasks to accomplish. Planning proceeds by decomposing tasks recursively into smaller and smaller sub-tasks, until primitive tasks, which can be performed directly using the planning operators, are reached. For nonprimitive each task, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates other methods to decompose the subtasks even further. If the constraints on the sub- tasks or the interactions among them prevent the plan from being feasible, the planning system backtracks and tries other methods.

In addition to a plan (i.e., a sequence of actions) for an input HTN planning problem, HPR also returns a task-dependency graph. The task-dependency graph generated by HPR consists of the hierarchy of tasks decomposed and generated by an HTN planner to find a plan, and it holds information about the dependencies among those tasks in the hierarchy. We represent such dependencies between tasks via the causal links among them. Using these causal links, HPR can determine which parts of the plan are affected by the result of a certain operator application. If an action fails during execution, these causal links help HPR to find which decompositions in the HTN trace are not valid anymore and need to be replanned.

When a contingency is identified for an action in the solution plan, HPR checks every parent of the failed action using the dependency graph generated before. The algorithm first identifies the minimal failed parent, a nonprimitive task t in the dependency graph, and it attempts to generate a new decomposition for t. Then it identifies the set of causal links in the current dependency graph that are supported by the plan originally generated for t. Then, HPR generates a new plan for t and its corresponding dependency graph. Next, HPR establishes all of the causal links between the any task in the new dependency graph and the tasks in the previous dependency graph that are not accomplished yet. After this operation, there may some tasks in the original dependency graph that are not supported by the current plan.

If the algorithm generates a new plan in which all the causal links are satisfied, it resumes the execution of this new plan, starting from first unexecuted action. Otherwise, HPR calls it self recursively on the parent task of t in the plan hierarchy. HPR repeats the plan repair process above until one of the following holds: either HPR generates a plan that is executed successfully in the world, or the plan-repair process marks a goal task as failed. In the latter case, HPR reports failure.

Measuring Self Confidence

Given all counter examples for contingencies C, and their probability of occurrence, and possible adaptations against those contingencies, we used a simple measure of self confidence in a plan π:

$$P(C') = \sum_{c \in C} (1.0 - prob(c)),$$

for each counter example c found against π, where $prob(c) = p(failure \mid events)$; e.g., events might include cloud cover, high winds, etc.

We assume that a priori probabilities of the possible exogenous events are given as input. For example, weather forecast might report that there will be a cloud cover at particular location with some probability. We also assume that every planning operator in he planning domain that Murphy and HPR works with is associated with a baseline probability of failure – i.e., each action represents a conditional probability that describes the event that the effects of the action will not be achieved given the preconditions are satisfied in a particular world state.[2] Given this probabilistic model of the planning domain, we use approximate Bayesian inference (Pearl 1988) to estimate the conditional probability $p(failure \mid events)$ above.

Note that in cases where HPR can repair the plan against the exogenous events, the robustness, and therefore the self confidence, of the plan increases. If HPR fails to produce a repair and reports a failure, this does not change the self confidence measure over a plan produced the above computation. Therefore, the self confidence measure about is monotonically non-decreasing by the amount of adaptation made by HPR in the system. In those cases where the self confidence of the system does not change, the system can report this to the users who can decide if the plan must be modified against an exogenous event and how.

[2]The reason that we assume a probability of failure instead of a probability of success for each action is that we believe the former is easier to produce as input to the system. The successful execution of an action can be disturbed for many reasons, some of which we may not know a priori. However, we may predict a lower bound probability of failure for an action, without enumerating all of the break down cases for it.

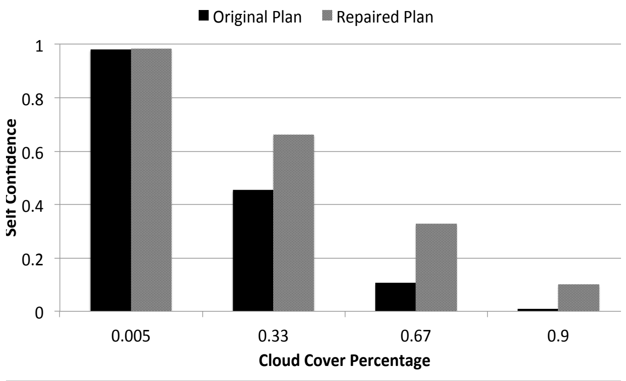

Figure 2: Preliminary results on our contingency analysis and plan repair approach to derive self confidence measures.

Preliminary Evaluations

We have implemented a preliminary system based on Murphy and hierarchical plan repair techniques described above. Our implementation is on Common Lisp. For experiments, we have developed a multi-UAV Intelligence, Surveillance, and Reconnaissance (ISR) scenario, in which a group of UAVs are tasked with flying to a target location and taking an image of that location. The UAVs may be equipped with different types of cameras for this task, and we used EO or IR cameras for our scenarios. For exogenous events that may break an ISR plan, we modeled weather conditions such as cloud cover and dust storm, which may occur depending on the terrain a scenario is taking place.

Figure 2 shows our preliminary results with our approach in a suite of above ISR scenarios. In this case, the weather event of a cloud cover forming at the target area so that the UAVs' camera's may not take a picture of their targets. The results show that as we increase the likelihood of a contingency happening, the adapted plans are more robust to failure compared to the original plan and therefore, the system's self confidence in its plans are higher. Note that in both the original plans and the adapted plans, the self confidence value drops due to the contingencies, but the rate of the drop is much higher in the original plan than in the adapted plans.

Conclusions

We have described our approach to compute an autonomous system's self confidence in its plans. Our approach uses two different but complementary automated planning techniques: (1) counter planning to identify likely contingencies in a plan before those contingencies create a failure during execution; and (2) plan repair to adapt the plan to prevent those failures to happen. Together, these techniques make it a plan more robust and stable to those identified contingencies and provide a self-proof to the system that it can adapt to failures. We believe such self-proofs constitute the basis of the system's self confidence in its operations and output. We describe a simple probabilistic measure that generates a self confidence value associated with a plan. Preliminary experiments show that this self confidence measure yields much higher self confidence indeed when the system can adapt its plans, compared to its self confidence in the original plans.

Acknowledgments. This work was supported under Air Force Research Laboratory contract #FA8650-09D-3900 Task Order 0015 via a subcontract to Infoscitex Corporation (IST). SIFT would like to thank Dr. Mark Draper (AFRL) and Dr. Tom Hughes (IST) for their support and oversight.

References

Ayan, F.; Kuter, U.; Yaman, F.; and Goldman, R. P. 2007. HOTRiDE: Hierarchical Ordered Task Replanning in Dynamic Environments. In *Proceedings of the ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution*.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Semantics for hierarchical task-network planning. Technical Report CS TR-3239, UMIACS TR-94-31, ISR-TR-95-9, University of Maryland.

Fox, M., and Long, D. 2001. PDDL+ level5 : An extension to PDDL2.1 for modeling domains with continuous time-dependent effects. technical note, University of Durham.

Gigerenzer, G., and Todd, P. M. 1999. *Simple heuristics that make us smart*. Oxford University Press, USA.

Goldman, R. P.; Kuter, U.; and Schneider, A. 2012. Using classical planners for plan verification and counterexample generation. In *Proceedings of CP4PS-12*.

Hutchins, A. R.; Cummings, M.; Draper, M.; and Hughes, T. 2015. Representing autonomous systems self-confidence through competency boundaries. Technical report, DTIC Document.

Kuter, U. 2012. Dynamics of behavior and acting in dynamic environments: Forethought, reaction, and plan repair. Technical Report 2012-1, SIFT.

Lee, J. D., and See, K. A. 2004. Trust in automation: Designing for appropriate reliance. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 46(1):50–80.

McDermott, D. 1998. PDDL, the planning domain definition language. Technical report, Yale Center for Computational Vision and Control.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Fransisco, CA: Morgan Kaufmann.