Learning Constraints via Demonstration for Safe Planning*

Ugur Kuter[†] and **Geoffrey Levine**[‡] and **Derek Green**[§] and **Anton Rebguns**[§]

Diana Spears[§] and **Gerald DeJong**[‡]

[†]University of Maryland, Institute for Advanced Computer Studies, College Park, MD 20742, USA

[‡]University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, IL 61801, USA

[§]University of Wyoming, Computer Science Department, Laramie, WY 82071, USA

Abstract

A key challenge of automated planning, including "safe planning," is the requirement of a domain expert to provide the background knowledge, including some set of safety constraints. To alleviate the infeasibility of acquiring complete and correct knowledge from human experts in many complex, real-world domains, this paper investigates a technique for automated extraction of safety constraints by observing a user demonstration trace. In particular, we describe a new framework based on maximum likelihood learning for generating constraints on the concepts and properties in a domain ontology for a planning domain. Then, we describe a generalization of this framework that involves Bayesian learning of such constraints. To illustrate the advantages of our framework, we provide and discuss examples on a real test application for Airspace Control Order (ACO) planning, a benchmark application in the DARPA Integrated Learning Program.

Introduction

Great strides have been made recently in automated planning, most notably in techniques that use background knowledge to efficiently generate plans. The effectiveness of these advances has been demonstrated in international planning competitions (Fox & Long 2002; Bacchus 2000), as well as partially in some real-world applications such as railroad management systems (Cimatti *et al.* 1997), autonomous navigation and space applications (Aiello *et al.* 2001; Bernard *et al.* 1998; Ai-Chang *et al.* 2003), and rescue & evacuation operations (Ferguson & Allen 1998).

Along with these successes, there has been a growing recognition that sophisticated, *multi*-planner systems are needed for full automation in real-world applications involving complex, uncertain, and time-directed situations. In those situations, extensive planning knowledge is required but is difficult to obtain. This is partly because of the complexities in the environments, e.g., rescue and evacuation operations, and it is partly because there is often no expert to provide it, e.g., space operations. In such complex domains, a planning system that can learn such knowledge to develop ways on how to safely and correctly operate in the world holds great promise for success.

*Author contact emails: ukuter@cs.umd.edu, levine@uiuc.edu, {derekg,anton,dspears}@cs.uwyo.edu, mrebl@uiuc.edu

Copyright © 2007, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

This paper describes a new framework for automated learning and use of planning knowledge in the form of "safety constraints," in the context of a multi-planner system. In particular, we present the following:

- A framework called ConstraintLearner (CL), which learns safety constraints from a "demonstration trace" that contains a sequence of actions and/or decisions taken by an expert in solving a planning problem. The demonstration trace includes expert behavior, but not high-level planning knowledge/strategies.
- A module, called SafetyChecker (SC), which is responsible for verifying the correctness of plans. SC uses the set of learned safety constraints to evaluate correctness.
- An extension of our CL framework for Bayesian learning of safety constraints. This extension enhances CL's robustness and flexibility, which is needed when information is lacking in the demonstration trace.
- A preliminary explanation-based learning approach to further enhance the learning from sparse data by interjecting causality.
- Examples that illustrate the operation and the advantages of our framework, which has been deployed in a realistic test application on Airspace Control Order (ACO). This application is one of the benchmarks used in the DARPA's Integrated Learning Program.

Preliminary Definitions

For effective learning of safety constraints for planning, we have developed a *domain knowledge ontology*, O, that specifies conceptual knowledge about the underlying learning domain and the relationships, i.e., *domain properties*, between domain concepts. More formally, a *domain concept* defines a set of individuals in the world that belong together because they share some properties and a *domain property* is a partial function of the form $C \rightarrow C'$ where C and C' are two domain concepts in O. The domain theory, D, consists of both O and a set of facts.

The input to CL includes an *expert demonstration learning trace*, which consists of a sequence of actions taken by a domain expert. The output is a set of safety constraints.

Let C be a set of concepts from a domain ontology, let P be a set of properties from that ontology such that every property $p \in P$ has a concept from C as its domain, and

Procedure ConstraintLearner(D, T, C, P)
1. initialize the lower and upper bounds given C and P, if such knowledge exists
2. loop over each demonstration d in the expert trace T
3. $X \leftarrow \{x \mid x$ is an instance of a concept $c \in C$ that appear in $d\}$
4. for each $x \in X$ do
5. let p be a property in P on x, and let $p(x)$ be the value specified for x in d
6. if a lower bound $lb(p)$ has been learned before for x then
7. if $p(x) < lb(p)$ then $lb(p) \leftarrow p(x)$
8. else
9. $lb(p) \leftarrow p(x)$
10. if an upper bound $ub(p)$ has been learned before for $p(x)$ then
11. if $ub(p) < p(x)$ then $ub(p) \leftarrow p(x)$
12. else
13. $ub(p) \leftarrow p(x)$
14. return all of the lower and upper bounds learned

Figure 1: The ConstraintLearner (CL) procedure.

let T be a learning trace. We define a (safety) *constraint* as a tuple of the form (c, p, lb, ub), where c is an instance of a concept in C, p is a property in P, and lb and ub are the lower and the upper bounds on the value of the property p associated with the concept instance c. An example concept might be the types of missions that a plane can be used for, and an example property might specify the altitude of that plane. A *constraint database* χ is a finite set of constraints.

A constraint database is *admissible* for the learning trace, T, if for each instance $c \in C$ and the property $p \in P$ associated with c that appears in T, there is a constraint (c, p, lb, ub) in χ such that lb and ub are lower and upper bounds on all of the possible values of $p(c)$ in T.

A *learning problem description* (or a *learning problem* for short) is a tuple (D, T, C, P), where D is the background domain theory, T is a learning trace, C is a finite set of concepts from O, and P is a finite set of properties on the concepts from C. A *solution* for a learning problem is a constraint database χ that specifies an admissible constraint on the value of each property in P associated with each instance of a concept in C that appears in the learning trace T.

Learning Constraints over a Domain Ontology

In this section, we describe our framework, called ConstraintLearner (CL), for learning a constraint database that is admissible for a given learning problem. CL takes as input a learning problem (D, T, C, P) and returns a solution constraint database for that problem. The learning trace T contains specific choices that an expert user has made on the possible instances of a set of domain concepts and the properties on those concepts. CL notes every concept instance from C and property value in P for that concept instance in the learning trace. Whenever new concept instances appear in the trace, either new lower and upper bounds are learned on the value of the property associated with that concept, or previously-learned bounds are updated. A key assumption of our algorithms is that lower and upper bounds exist for all domain concepts and their specified properties, whether they are previously known or not.

We now explain the learning algorithm we have implemented for CL, which is a maximum likelihood algorithm. Figure 1 shows the pseudocode. With the input described above, CL starts by initializing the lower and upper bounds for each concept c in C and each property p in P for c by using prior bounds from the domain theory, if they exist in D. After the initialization, learning proceeds by traversing the entire input demonstration trace. During this traversal, the expert trace provides new and modified lower and upper bounds from which the CL can learn new constraints and/or update existing ones using generalization and specialization.

In Figure 1, Lines 3–13 perform the learning and/or update steps in the algorithm. Let d be an acceptable demonstration in the learning trace T. CL first generates every instance of a concept c from C that appears in the demonstration d. In the pseudocode in Figure 1, the set of all such concept instances is shown as the set X. The algorithm then checks every concept instance x in X, if it has learned a lower and upper bound for that concept instance before. Let $p \in P$ be a domain property for x. If there is a previously-learned lower bound $lb(p)$ and if the value $p(x)$ specified in d is less than the previous lower bound, then the CL updates its lower bound on x to be $p(x)$. If $p(x)$ is larger than $lb(p)$, then it continues without taking action. The update for the upper bound $ub(p)$ is similar.

The CL terminates after completing its examination of the entire trace. It returns the learned constraints.

We now demonstrate the learning behavior of CL given the portion of the user demonstration trace shown in Figure 2 for an Airspace Control Order (ACO) domain. An ACO problem requires 4D deconfliction of a given set of 3D airspaces with associated time periods for each aircraft. As these airspaces are created by distinct entities, they can conflict. An ACO problem consists of making necessary adjustments to deconflict the airspaces.

In a learning problem, ACO domain concepts and properties include descriptions of airspaces and specifications of the altitude and time period associated with each airspace. For example, Figure 2 highlights two domain properties: the minimum and maximum allowed altitudes of an airspace identified as **F4**. Traversing this trace, CL learns that the lower and upper bounds on the minimum altitude of **F4** are

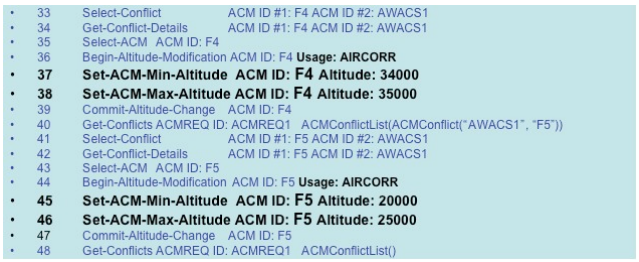

Figure 2: An example user demonstration trace in the Airspace Control Order Domain.

both the same and they are 34,000 feet. Similarly, it learns that the lower and upper bounds on the maximum altitude of **F4** are 35,000 feet. Now suppose the trace contains another demonstration on the value of the minimum altitude of **F4** to be 25,000 feet. In that case, CL updates the lower bound for the minimum altitude of **F4** to be 25,000 and does not change the upper bound.

Using the Learned Constraints for Planning and Plan Verification

After the constraints have been learned, they are then used by one or more plan learners to generate constrained candidate partial plans. Because these candidate partial plans have been generated by multiple independent plan learners, they may be inconsistent. Therefore, the SafetyChecker (SC) is called to verify that the composed plan fragments obey the learned safety constraints.

Consider the ACO domain, for example. Each planner has the job of proposing a set of deconfliction actions (which constitute a candidate partial plan). These actions are intended to be applied to an ACO, which consists of spatio-temporal locations of all airspaces. To verify that the candidate partial plan satisfies all safety constraints, the SC is invoked. The SC applies the proposed set of actions, thereby developing a hypothetical scenario – a modified ACO. The SC then uses its 4D Spatio-Temporal Reasoner to verify whether each constraint is still satisfied or not. Any violations are reported for evaluation. Violation reports include the violated constraint, specific information about the violation, optional advice for plan repair, and the degree (severity) of violation [0.0, 1.0]. Specific information about the violation describes which aircraft in the ACO caused the violation, e.g., fighter **F4**. The degree of violation is used to rank violations, to allow planners to first concentrate problem resolution on more critical violations. It also makes it possible to ignore less severe violations in the event that no completely safe plan is discovered within the alloted time.

Planners use this feedback from the SC to repair and replan, until a plan is found that is violation-free or has an acceptable violation level. Figure 3 shows a screen shot of our 4D Spatio-Temporal Reasoner performing safety checking.

Bayesian Constraint Learning

In many complex domains, such as Airspace Deconfliction, the learning framework we described so far may only have access to a limited length execution trace, but have a large number of constraints to learn. In these cases, there will be very few examples from which to learn constraints. For this reason we expect that the simple maximum likelihood approach may be overly restrictive. By extending the CL to learn within a Bayesian framework, we may be able to more accurately learn constraint values from few examples.

In a Bayesian framework, learning proceeds as follows. As described previously, CL is given a set of domain concepts $C = \{c_1, c_2, ...\}$, a set of domain properties $P = \{p_1, p_2, ...\}$ on the concepts in C, and an expert trace T made up of demonstrations of acceptable values for the individual domain properties, $\{p_r(c_i), p_s(c_j), ...\}$. The goal is to learn a constraint database, $\chi = \{(c_i, p_r, lb(p_r), ub(p_r)), (c_j, p_s, lb(p_s), ub(p_s)), ...\}$.

In the Bayesian setting, we estimate $P(\chi|T)$, the posterior probability of χ, given the observed demonstration T. Bayes' rule implies that $P(\chi|T) = P(T|\chi) \times P(\chi)/P(T)$, where $P(T|\chi)$ represents the probability of observing trace T given constraint database χ, $P(\chi)$ is the prior belief over constraint databases from our domain theory, and $P(T)$ is the normalizing probability of observing trace T. Assuming that constraints corresponding to distinct concepts and properties are independent, we have $P(\chi|T) = \prod_{c \in C, p \in P} P((c, p, lb(p), ub(p))|T)$.

The above assumption enables us to decompose the general problem into manageable subproblems. Consider one such subproblem, the case of learning the constraint for the range of altitudes associated with some airspace, $\omega = (c_i, p_r, lb(p_r), ub(p_r))$. We learn by witnessing evidence in the form of $p_r(c_i)$, from the expert, as he/she positions and reposition the corresponding airspace. Bayesian learning proceeds as follows. Our prior belief over the values of $lb(p_r)$ and $ub(p_r)$ is represented by a distribution $P(\omega)$. When we observe evidence from the expert, $p_r(c_i)$, this distribution is updated to the posterior $P(\omega|p_r(c_i))$. Again, applying Bayes' rule we have that:

$$P(\omega|p_r(c_i)) = \frac{P(p_r(c_i)|\omega) \times P(\omega)}{P(p_r(c_i))} \quad (1)$$

Here, $P(\omega)$ represents the prior generic distribution of airspace min/max altitudes and $P(p_r(c_i)|\omega)$ is the probability that an airspace will be placed at a certain location given the altitude constraint.

These values can usually be defined using reasonable assumptions. For instance, in the context of our airspace deconfliction example, $P(\omega)$ can be obtained from a Gaussian approximation of the real distribution by asking the expert for the average, variance, and covariance of the minimum and maximum altitudes.

$P(p_r(c_i)|\omega)$ is the probability that an airspace will be placed at a certain location given the altitude constraints. There are two cases to consider. First, we must consider when an airspace is first created. Second, we must consider when an airspace already exists, but is moved to resolve a conflict with another airspace. In the first case it is reasonable to assume that the expert never violates safety constraints, and that the airspace is placed uniformly between the minimum and maximum altitude constraint values. For

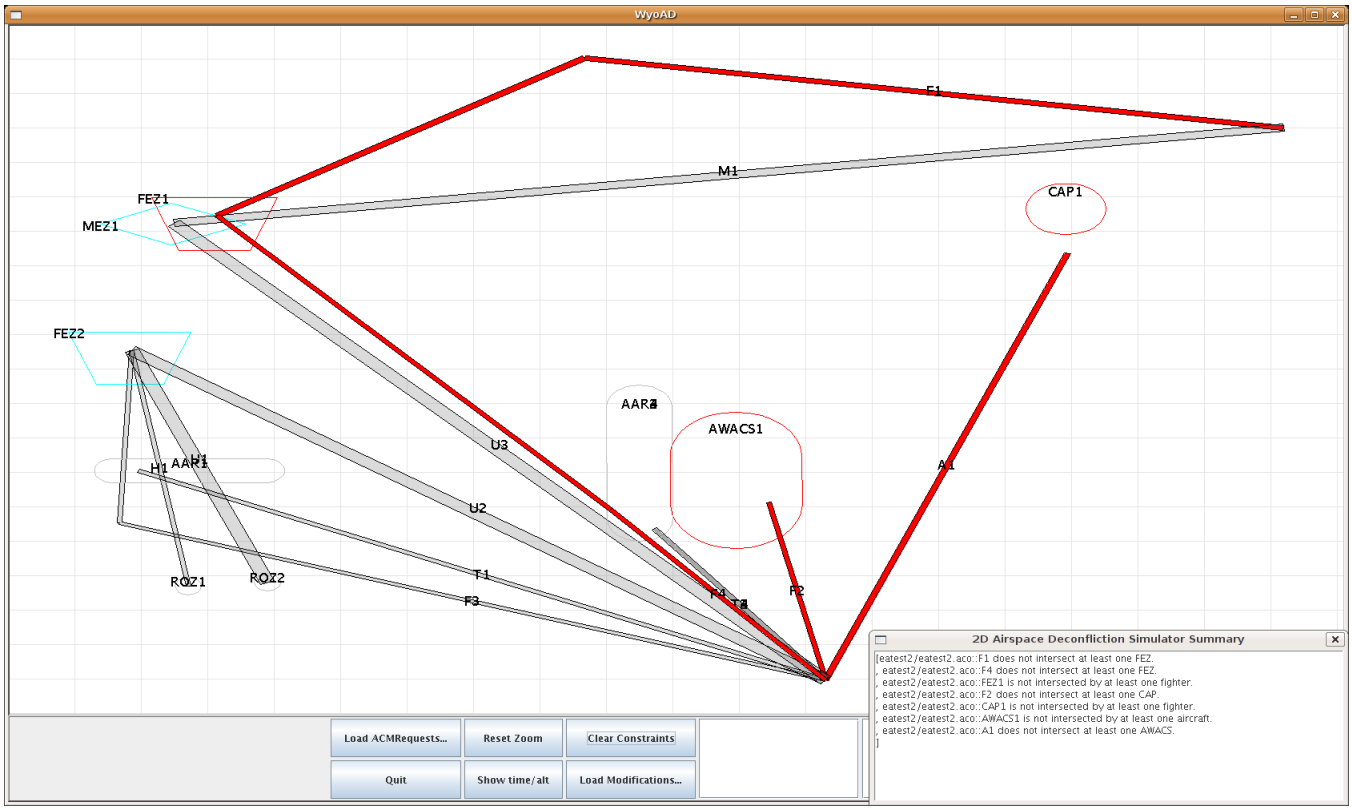

Figure 3: Visualization tool showing SafetyChecker verifying constraints.

the second case, we must consider not only the final position of the airspace, but also how the expert moves the airspace during deconfliction. Experts, for example, demonstrate a preference for moving airspaces minimally far. A reasonable assumption in the context of deconfliction is that the expert always chooses the minimum cost deconfliction operation that does not violate safety constraints. To solve such constrained minimization problems, we plan to explore the relevant literature in the field of operations research.

A significant benefit of the Bayesian framework is the ease of converting to ϵ-safe constraints. Consider the case of the altitude upper bound, $ub(p_r)$. Given a posterior belief $P(ub(p_r))$ and safety parameter ϵ, the upper bound output for use by the planners and verifier is z s.t. $P(ub(p_r) < z) = \epsilon$. Thus, a safety constraint $ub(p_r)$ is output that is at least as safe as the true unknown safety constraint with probability 1 - ϵ. Adjusting ϵ affects how conservative or risky the output constraints will be. Setting $\epsilon = 0$ defaults to the simple maximum likelihood learning introduced earlier.

We now demonstrate the behavior of the Bayesian learning approach for a particular airspace type with true upper and lower altitude bounds 25000 and 40000 feet, respectively. Suppose that our prior knowledge suggests that lower and upper bounds of airspaces have means 20000 and 50000 feet, each with standard deviation 15000 feet and joint covariance (10000 feet)^2. We observe the expert successively allocate corresponding airspaces, detailed as follows:

Demonstration	Altitude (p_r) Value
1	31,847
2	36,431
3	25,278
4	37,321
5	31,671
...	...

Figure 4 plots how the posterior belief over $P((c_i, p_r, lb(p_r), ub(p_r))|T)$ changes as the number of demonstrations observed increases. Starting with a broad Gaussian distribution, the posterior belief is updated to eliminate all regions of the $(lb(p_r), ub(p_r))$ space that are inconsistent with the observed data. As more evidence is observed without any altitudes outside of [25000, 40000], the posterior becomes sharper about this point. We also display the ϵ-safe constraints based on the posterior for several values of ϵ.

Explanation-Based Constraint Learning

We are currently working on further generalizing our learning framework toward the acquisition of subtler functional patterns over deconfliction decisions. Due to the paucity of data assumed in our domains, statistical learning is inappropriate. Instead, we combine inductive and Bayesian learning (see above) with Explanation-Based Learning (EBL). Although this portion of our work is not yet implemented,

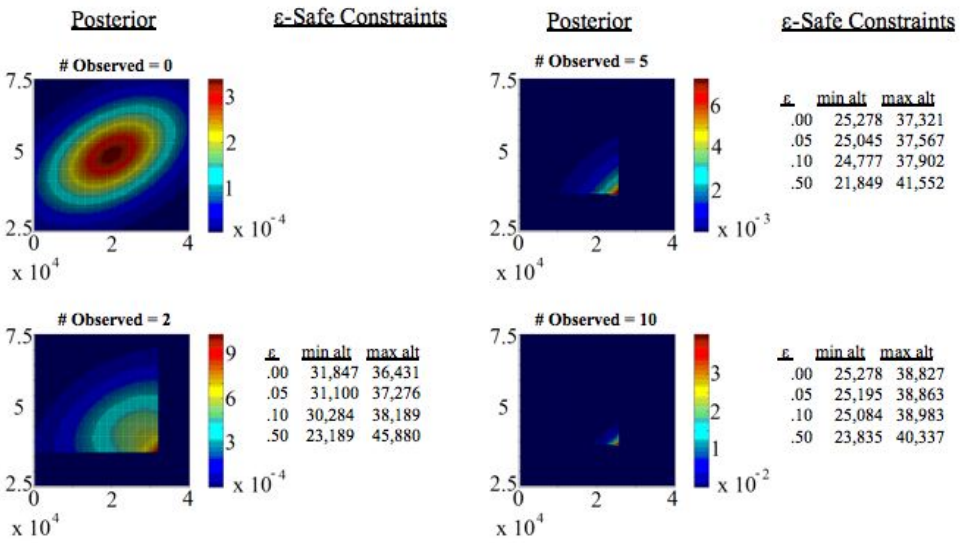

Figure 4: Constraint posterior belief and epsilon-safe constraints after observing 0, 2, 5, and 10 demonstrations from the expert. Axis correspond to the minimum altitude bound (x-axis) and maximum altitude bound (y-axis).

this section outlines our approach. The EBL framework will make use of declarative prior knowledge from an expert. In the ACO domain we expect to exploit prior knowledge about airspaces and their causal relationships. For example, our prior knowledge will include the fact that each airspace serves a purpose. The system will be provided with prior knowledge relating airspaces to possible purposes, interactions and relations among these purposes, and how functional purposes are likely to influence airspaces.

Suppose, for example, that an expert moves two airspaces, **F4** and **T1**, but only **F4** is actually in a conflict condition (say with airspace **A3**). One possibility is that the expert is adding specious complexity to his solution. But another possibility is that the second airspace, **T1**, is in a kind of virtual conflict induced by moving **F4**. That is, **T1** is instrumental to the purpose of airspace **F4** and must be moved to preserve the integrity of the underlying functional relation. If **F4** is (or can be inferred to be) the loiter orbit for a protective combat air patrol or other persistent air asset, then some other nearby airspace is likely to be the loiter orbit of the supporting refueling tanker. When the primary airspace is moved, its functionally derivative airspaces must be examined to insure that the functional support is intact and if not, they must be suitably adjusted.

An EBL system will annotate airspaces with inferred properties that are mentioned in the domain theory and prove to be useful in explaining or justifying the experts' decision traces. The learning problem is then reduced to estimating parameters that govern the tradeoffs among preferences rather than inventing the existence of those tradeoffs. Relatively few expert examples may be required to learn a good balance between tanker safety and refueling effectiveness for forward assents. Without the inferential ability to incorporate this domain knowledge, the expert behavior may appear artificially complex. With sufficient knowledge and inferential capacity, the system could understand why an expert might depart from such learned tradeoffs in order, for example, to exploit an existing defensive air patrol to aid in protecting a newly moved tanker loiter region.

Related Work

Our general approach of learning from observing human expert behavior can be traced at least back to the learning apprentice paradigm. For example, Mitchell *et al*'s LEAP is a system that learns to VLSI design by unobtrusively watching a human expert solving VLSI layout problems (Mitchell, Mahadevan, & Steinberg 1985). Similarly, in (Shavlik 1985), Shavlik shows how the general physics principle of Momentum Conservation can be acquired through the explanation of a "cancelation graph" built to verify the well-formedness of the solution to a particular physics problem worked by an expert. More recently, the apprenticeship paradigm has been applied to learning hierarchical task networks (Nejati, Langley, & Könik 2006), and to learning autonomous control of helicopter flight (Abbeel & Ng 2005).

Our learning framework, when seen in the context of multiple planners, may at first seem to fit into the paradigm of integrated architectures (Langley 2006). These include ACT*, SOAR, THEO, ICARUS, PRODIGY, and many others. But our motivation is quite different. These architectures are directed toward integration in a psychologically plausible way. The cognitive design criteria, the homogeneity of a unifying computational mechanism, their claims of general intelligence, and their sharing of common representations set them apart from our multi-planner framework. In addition, they explicitly eschew the concept of a single proof-of-concept

application domain that we follow.

Our research is also strongly related to prior research in two areas: learning control rules for search/planning, and "safe planning." The former area has a long history, e.g., see (Minton & Carbonell 1987), and has more recently evolved into the learning of constraints (Huang, Selman, & Kautz 2000) for constraint-satisfaction planning (Kautz & Selman 1999). The purpose of safe planning is to ensure that plans made by agents obey safety constraints that prevent them from resulting in dangerous consequences (Weld & Etzioni 1994; Gordon 2000).

Our research is novel in two respects. First, unlike the prior research on learning control rules and safe planning, our work focuses on learning from observing an expert's behavior. For this reason, we maximize the information from the input demonstration trace. To do this, we are employing a hybrid learning approach that combines induction, Bayesian learning, and EBL. We are unaware of any other approaches to learning constraints that combine all three. The second novel aspect of our work is that we are not only learning constraints for planning, but we are in fact learning constraints for *multiple heterogeneous planners*, *as well as for a verification module*. This general-purpose use of the learned constraints further motivates our hybrid approach, and it also motivates a future study of lazy-eager tradeoffs. In particular, recall the ϵ-safe aspect of constraints. The ϵ safety margin could be fixed at constraint learning time, or it could vary based on run-time considerations of the planners and the Safety Checker.

Conclusions

This paper has described a new framework for learning safety constraints in an application domain by observing a demonstration trace generated by a domain expert. The trace includes information about the expert's behavior, but it does not include complex high-level planning knowledge. A preliminary implementation of our approach in a multi-planner learning/reasoning system developed for the DARPA Integrated Learning Program demonstrated its effectiveness at facilitating the production of safe plans. Our approach will soon be a hybrid of three learning techniques: inductive, Bayesian, and explanation-based. It will then be capable of even greater exploitation of few examples.

Acknowledgments

This work is funded by the DARPA GILA Contract # FA8650-06-C-7605. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

Abbeel, P., and Ng, A. Y. 2005. Exploration and apprenticeship learning in reinforcement learning. In *ICML*, 1–8.

Ai-Chang, M.; Bresina, J.; Charest, L.; Hsu, J.; Jónsson, A. K.; Kanefsky, B.; Maldague, P.; Morris, P.; Rajan, K.; and Yglesias, J. 2003. MAPGEN Planner : Mixed-initiative activity planning for the Mars Exploration Rover mission. In *Printed Notes of ICAPS'03 System Demos*.

Aiello, L. C.; Cesta, A.; Giunchiglia, E.; Pistore, M.; and Traverso, P. 2001. Planning and verification techniques for the high level programming and monitoring of autonomous robotic devices. In *Proceedings of the European Space Agency Workshop on On Board Autonoy*. Noordwijk, Netherlands: ESA.

Bacchus, F. 2000. AIPS-00 planning competition. `http://www.cs.toronto.edu/aips2000`.

Bernard, D.; Gamble, E.; Rouquette, N.; Smith, B.; Tung, Y.; Muscettola, N.; Dorias, G.; Kanefsky, B.; Kurien, J.; Millar, W.; Nayak, P.; and Rajan, K. 1998. Remote agent experiment. ds1 technology validation report. Technical report, NASA Ames and JPL report.

Cimatti, A.; Giunchiglia, F.; Mongardi, G.; Pietra, B.; Romano, D.; Torielli, F.; and Traverso, P. 1997. Formal Validation & Verification of Software for Railway Control and Protection Systems: Experimental Applications in ANSALDO. In *Proc. World Congress on Railway Research (WCRR'97)*, volume C, 467–473.

Ferguson, G., and Allen, J. 1998. TRIPS: An integrated intelligent problem-solving assistant. In *AAAI/IAAI Proceedings*, 567–572.

Fox, M., and Long, D. 2002. International planning competition. `http://www.dur.ac.uk/d.p.long/competition.html`.

Gordon, D. 2000. Asimovian Adaptive Agents. *JAIR* 13:95–153.

Huang, Y.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *Proc. 17th International Conference on Machine Learning (ICML'00)*, 337–344.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 318–325.

Langley, P. 2006. Cognitive architectures and general intelligent systems. *AI Mag.* 27(2):33–44.

Minton, S., and Carbonell, J. 1987. Strategies for learning search control rules: An explanation-based approach. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 228–235.

Mitchell, T. M.; Mahadevan, S.; and Steinberg, L. I. 1985. Leap: A learning apprentice for vlsl design. In *IJCAI*, 573–580.

Nejati, N.; Langley, P.; and Könik, T. 2006. Learning hierarchical task networks by observation. In *ICML*, 665–672.

Shavlik, J. W. 1985. Learning about momentum conservation. In *IJCAI*, 667–669.

Weld, D. S., and Etzioni, O. 1994. The first law of robotics (a call to arms). In *AAAI-94*, 1042–1047.