*Computational Intelligence*, Volume 59, Number 000, 2009

# LEARNING AND VERIFICATION OF SAFETY PARAMETERS FOR AIRSPACE DECONFLICTION

Antons Rebguns, Derek Green, Diana Spears
*University of Wyoming, Department of Computer Science*
*Laramie, WY 82071, USA*

Geoffrey Levine
*University of Illinois at Urbana-Champaign, Department of Computer Science*
*Urbana, IL 61801, USA*

Ugur Kuter
*University of Maryland, Institute for Advanced Computer Studies*
*College Park, MD 20742, USA*

We present a Bayesian approach to learning flexible safety constraints and subsequently verifying whether plans satisfy these constraints. Our approach, called the *Safety Constraint Learner/Checker (SCLC)*, is embedded within the *Generalized Integrated Learning Architecture (GILA)*, which is an integrated, heterogeneous, multi-agent ensemble architecture designed for learning complex problem solving techniques from demonstration by human experts. The SCLC infers safety constraints from a single expert demonstration trace, and applies these constraints to the solutions proposed by the agents in the ensemble. Blame for constraint violations is then transmitted to the individual learning/planning/reasoning agents, thereby facilitating new problem-solving episodes. We discuss the advantages of the SCLC and demonstrate empirical results on an Airspace Planning and Deconfliction Task, which was a benchmark application in the DARPA Integrated Learning Program.

*Key words:* Keyword 1; Keyword 2.

## 1. INTRODUCTION

It is becoming increasingly apparent that sophisticated, multi-agent systems are needed for full automation in real-world applications involving complex, uncertain, and time-sensitive situations. Such real-world applications require a variety of capabilities including: planning under uncertainty and time, learning knowledge for planning, reasoning about world dynamics, and coordinating all of the above in an integrated artificial intelligence (AI) system.

One approach to addressing such a problem is to develop a single, centralized planning (and scheduling) system that provides all of these capabilities. However, such large monolithic centralized systems tend to be too brittle and complex, and are notoriously difficult to debug.

At the other extreme, we have completely decentralized and distributed multi-agent systems. These systems can provide a wide variety of capabilities and they are much simpler, and more robust and fault-tolerant, than monolithic centralized systems. However, fully decentralized systems with desirable joint behavior are much harder to design and construct initially than centralized systems. Therefore Xuan and Lesser, for example, propose that designers of planning systems start by engineering a monolithic centralized system, and then follow a procedure for converting to a decentralized system (Xuan and Lesser, 2002). This results in a fully-distributed, robust multi-agent planning system. The drawback, however, of decentralized multi-agent planning (and scheduling)

Address correspondence to Diana Spears, University of Wyoming, Department of Computer Science, Laramie, WY82071, email: dspears@cs.uwyo.edu.

© 2009 The Authors. Journal Compilation © 2009 Wiley Periodicals, Inc.

systems is that their behavior can be extremely difficult to understand and predict. In fact, successful formal verification of the behavior of such systems can be quite challenging, e.g., see (Gordon, 2000).

This paper describes a new approach to automated learning and application of problem-solving knowledge in the form of safety constraints. When reasoning in real-world domains, providing a safe solution can be just as important as fulfilling the goals. In particular, if the plan is used to determine the activities of humans in dangerous environments (as is the case for the military airspace deconfliction domain presented below), failing to adhere to safety constraints can lead to catastrophic results. We call our new approach the *Safety Constraint Learner/Checker (SCLC)*. The SCLC is part of a very large system called the *Generalized Integrated Learning Architecture (GILA)*, funded by DARPA. GILA adopts a multi-agent planning approach that consists of a loosely-coupled ensemble of heterogeneous learning and planning agents; it is a compromise between the fully-centralized and fully-decentralized extremes. (Hereafter, we sometimes abbreviate the term “learning and planning agent” with the term “agent.”) By adopting a compromise between the two extremes, GILA is more robust than the centralized extreme and more comprehensible and predictable than the decentralized extreme.

Although GILA’s ensemble approach yields many benefits, it introduces a new challenge. GILA combines multiple agents with a global system executive, thereby instantiating a hybrid of decentralization and centralization. The use of multiple agents gives the system multiple perspectives on a problem, and the central executive combines and arbitrates inputs from the loosely-coupled agents. The challenge in such a hybrid approach is that combining the outputs from the individual learning and planning agents into a single coherent plan can be a monumental task. Our solution to this challenge is to introduce SCLC constraints that can ameliorate this situation to a large extent, e.g., by helping to prune unacceptable subplan mergings, thereby resulting in a more tractable search space. Therefore, the SCLC plays a critical role in GILA.

This paper focuses on the learning and application of constraints in the context of the GILA architecture. Here, we describe the SCLC, which has been experimentally determined to be critical to the outstanding performance of GILA. The SCLC consists of two components – a learner and a checker:

(1) The SCLC’s *Constraint Learner (CL)* automatically infers safety constraints over a given domain from a trace demonstrating an example of a human expert’s solution to a problem within that domain. Such a *demonstration trace* consists of a sequence of actions and/or decisions taken by the expert while solving the planning problem. The CL uses a Bayesian learning technique to analyze such traces of expert behavior and generate safety constraints for the domain. The safety constraints are subsequently used to pinpoint safety violations within solutions generated by the ensemble of agents. Any proposed plan containing actions that violate safety constraints will result in an incorrect or unsafe solution. One of the most important tasks of a human expert is to resolve such conflicts. The constraint information allows the GILA system to more closely mimic such expert behavior.

(2) The SCLC’s *Safety Checker (SC)* is responsible for verifying the correctness of plans (proposed solutions) in terms of their satisfaction or violation of the safety constraints. The SC inputs a final proposed plan (composed by the executive from plan fragments from the individual planners), along with the set of safety constraints from the CL. It then outputs a degree of violation, which is used by GILA’s executive module to assign blame for each constraint violation found. The output of the SC can then be used as diagnostic feedback within the system for the purpose of proposing improved solutions.

Why is the SC needed if the learning/planning agents may use the safety constraints during their operations? The reason is that although the individual agents may use the safety constraints during their planning operations, they do not interact with one another during that phase. Because each agent may not have the domain knowledge, representational expressiveness, or learning and planning capabilities to solve the entire input problem, the agents output *partial* (incomplete) solutions, also called *solution fragments*. These incomplete solutions are intended to solve portions of the entire problem. The executive subsequently merges these results into a final full plan. This final plan needs to be checked because interactions between the partial solution plans will not emerge until they have been composed into the final plan.

We have implemented the SCLC as a component of the GILA multi-agent system. It could be considered yet another agent, but we do not call it an "agent" in this paper – because that might confuse the reader; we reserve the term "agent" for the agents that do planning (as well as learning and reasoning). The objective of the SCLC within GILA is to automate the constraint learning/satisfaction subtask in order to simulate the expert's verification process. Learning and checking constraints is especially important and challenging for systems that learn or adapt their strategies (Spears and Gordon, 2000). Sophisticated credit assignment and system recovery/repair algorithms are needed in order to compensate for the tendency of GILA's agents to propose schedules that, when combined, violate domain constraints. Although this is a considerable challenge, most recently (after two years of research) GILA has become competitive with and even slightly exceeded human novice trainee performance, which we believe is a considerable accomplishment. The SCLC appears to have played a notable role in this success, as described in the experimental results section below.

## 2. BACKGROUND: GENERALIZED INTEGRATED LEARNING ARCHITECTURE (GILA)

We start with a summary of the *Generalized Integrated Learning Architecture (GILA)*, funded by DARPA. (Zhang *et al.*, 2009) gives a much more detailed description of the GILA system. GILA consists of a set of three learning and planning agents, integrated and coordinated by a central meta-reasoning executive, and the SCLC.

### 2.1. Application Domain: Airspace Deconfliction and Management

The domain of application for the GILA project is military Airspace Management in an Air Operations Center (AOC). Airspace Management is the process of making changes to requested airspaces so that they do not overlap with other requested airspaces or previously approved airspaces. The task of de-conflicting these requested changes is normally executed by a human expert, the airspace manager. The airspace manager receives an *Airspace Control Order (ACO)* which specifies all the locations and trajectories of currently approved airspace objects. A set of *Airspace Control Means Requests (ACMReqs)* specifies the locations and trajectories of new airspaces that have been requested. Figure 1 is included to give an impression of the complexity of the problem. Notice the large number of overlapping airspace shapes. The airspace manager merges the ACMReqs with the initial ACO. He then *modifies* the airspaces to remove any spatio-temporal conflicts and/or safety constraint violations. The types of modifications include movements in latitude, longitude, time and altitude. Modifications are made with the intention of minimizing the impact on the mission.

The overall task objective is to derive a *solution plan* consisting of airspace modification actions that, when executed, merges the ACMReqs into the ACO, while de-conflicting all airspaces and satisfying all safety constraints. An airspace manager's task, namely, that of generating such airspace modification plans, is *extremely* complex and challenging. In fact, discussions with the DARPA BlueForce experts at this task reveal that it takes many years to become an expert. In some respects it is analogous to solving a huge jigsaw puzzle. However, in a puzzle each piece has one correct location, whereas in airspace deconfliction there are multiple good (or even optimal) solutions. This complicates the problem, thereby making it exceptionally difficult to automate – because re-planning might lead an airspace manager down a very different path.

The airspace manager usually uses some implicit expert knowledge in terms of safety constraints in order to resolve the ambiguity and conflicts between the proposed solutions – i.e., the airspace manager ensures that the overall solution adheres to critical safety, liveness, and stability constraints such that when the solution is executed there are no unsafe side effects.

### 2.2. GILA System Description

GILA has been developed as part of a large team effort, and was funded by DARPA in 2006-2008. There are three major learning and planning agents in GILA: namely, the *Symbolic Agent*, the

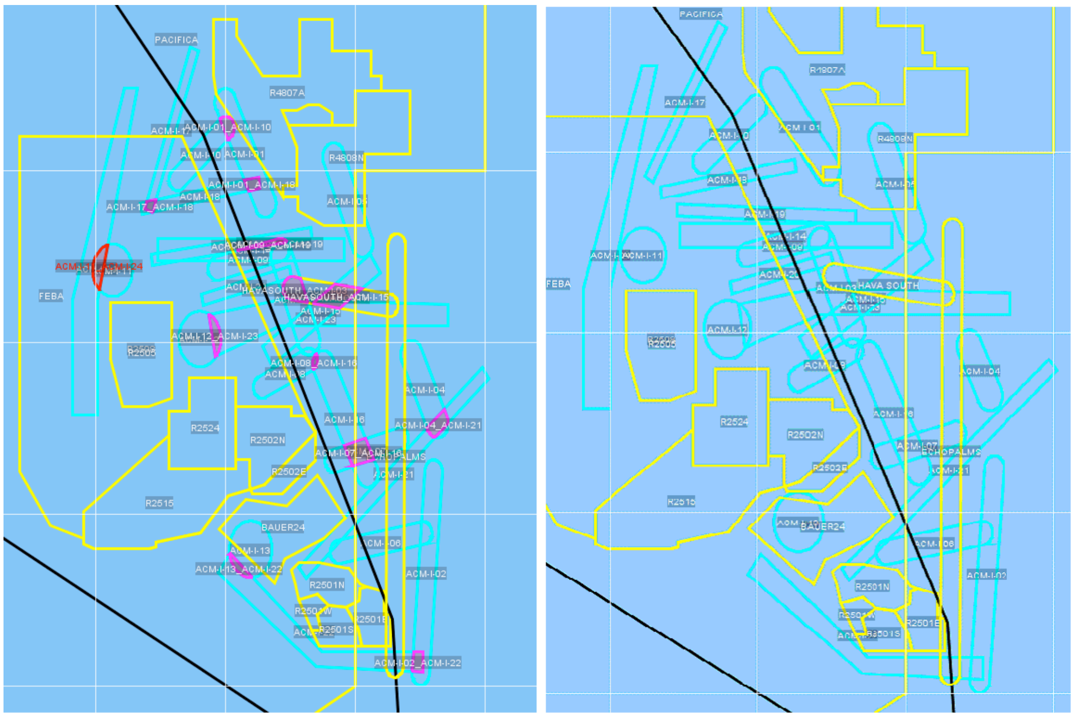


FIGURE 1. An example of an ACO (left) and its de-conflicted solution (right). Yellow: ACO airspaces. Light Blue: airspace requests. Red: conflicts.

*Decision-Theoretic Agent*, and the *Case-Based Agent*. We next describe these agents, which were designed and implemented by other GILA researchers.

Symbolic Agent. GILA's Symbolic Agent learns and applies hierarchically-organized planning knowledge as well as plans. It selects one of three top-level tasks, each of which corresponds to one way of modifying an airspace. These tasks are: modify the geometry (i.e., latitude and longitude) of an airspace, modify the altitude, and modify the airspace's schedule (i.e., starting and ending times). Each task is associated with a cost function.

The Symbolic Agent starts with one of the three tasks, along with an expert demonstration trace, and it learns a primitive sequence of modification operations analogous to those seen in the demonstration trace but tailored to the current problem. The learning process is similar to that in a macro-operator learner (see (Botea *et al.*, 2005) for a recent example of macro-operator learning in automated planning). However, unlike typical macro learners, the Symbolic Agent represents macros with a *Hybrid Hierarchical Representatation Machine* that combines both numeric and symbolic reasoning (Yoon and Kambhampati, 2007). The learning process is a form of simple logical deduction augmented by numeric reasoning (Yoon and Kambhampati, 2007).

Decision-Theoretic Agent. The primary objective of the Decision-Theoretic Agent is to learn the expert's cost function that he/she uses while solving airspace-deconfliction problems. This agent assumes that the expert *implicitly* applied an optimized cost function when generating the input demonstration trace. This implicit cost function must be inferred; therefore, the Decision-Theoretic Agent applies a decision-theoretic gradient boosting algorithm to infer numeric weights for domain features. Like the Symbolic Agent, the Decision-Theoretic Agent also learns minimal-cost plans.

Case-Based Agent. Note that the Symbolic Agent plans at an abstract symbolic level, whereas the Decision-Theoretic Agent provides lower-level numeric weights and costs. Therefore, these agents have *biases* (i.e., representational and procedural preferences) that are complementary. However, neither of these agents capitalizes on previous experience. The Case-Based Agent has been introduced to fill this gap. The Case-Based Agent learns, stores and uses a feature-based case database, by applying *case-based reasoning (CBR)* techniques (Aamodt and Plaza, 1994). It reuses the stored

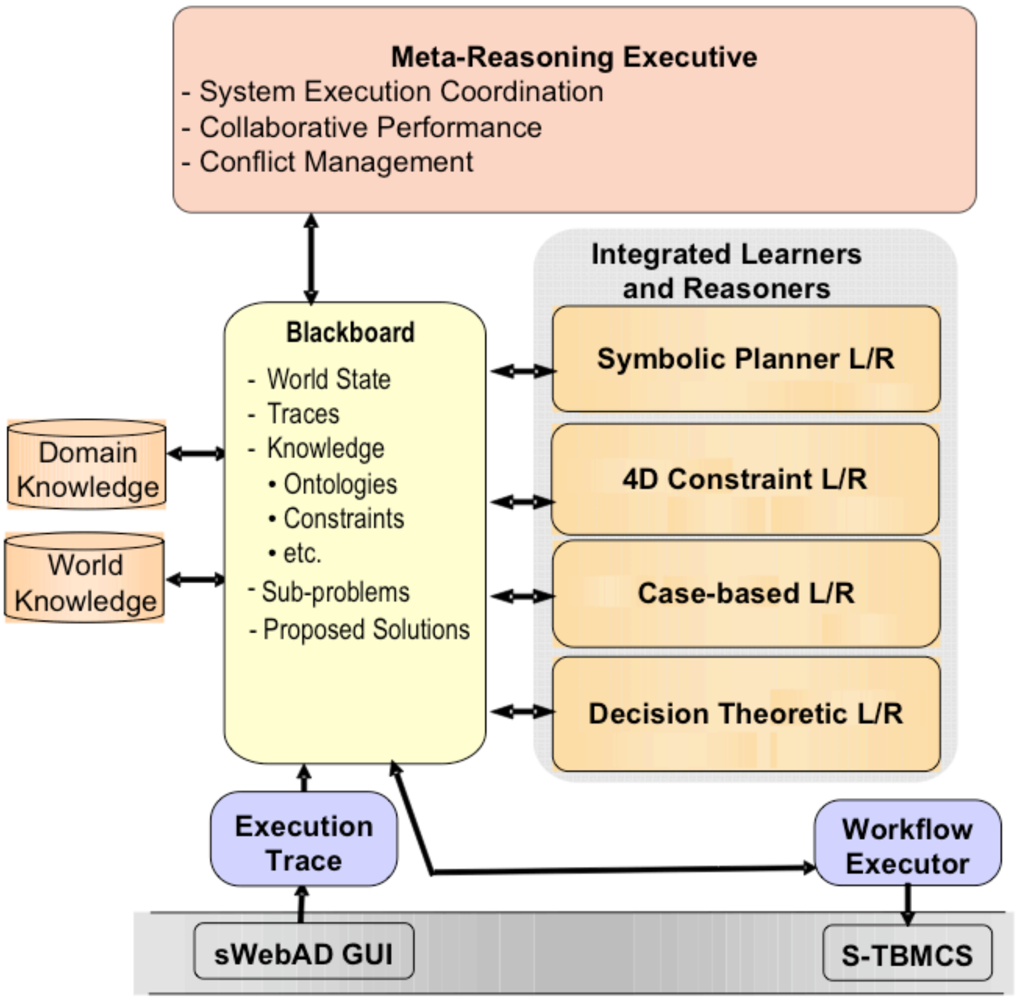


Figure 2. GILA System Architecture.

cases for solving previous deconfliction problems to derive plans for solving new (but similar) deconfliction problems.
The Case-Based Agent uses the expert demonstration to learn cases that describe conflicts and their solutions. The cases are formed by identifying and grouping modification steps corresponding to a conflict. Then, when a new airspace-deconfliction problem is presented to the system, the Case-Based Agent generalizes and matches the conflicts in that problem with the previously-learned cases, and it proposes generalizations of the sequence of steps (in the trace) as potential solutions for the new conflict. Unlike the other agents that generalize at learning time, the Case-Based agent does "lazy learning" in the sense that it generalizes cases at performance time.

The complementary biases of the three agents, when combined in the GILA loosely-coupled ensemble, demonstrate a favorable synergy for improved performance. GILA's ensemble of the above three agents, plus the SCLC and the executive module, are shown in the context of the GILA overall system architecture in Figure 2. The figure uses the terminology "L/R" to indicate that each agent, as well as the SCLC, is both a learner L and a reasoner R. The SCLC is called the "4D Constraint L/R" in the figure to clarify the SCLC's role.

A highly simplified version of the overall GILA algorithm contains the following main steps:

2.2.0.1. *Learning phase:*
(1) Input an expert-generated demonstration trace, to be used as an illustrative example of expert behavior, along with a practice problem and its (expert-provided) solution.
(2) The SCLC learns safety constraints about the domain from the demonstration trace.
(3) The three agents study the trace and the constraints and learn (using the practice problem and solution) to perform the deconfliction task as the expert does.

2.2.0.2. *Performance phase:*
(1) A new problem is presented to the system, divided by the executive into sub-problems.
(2) Each of the three learning, planning, and reasoning agents proposes solution fragments for sub-problems that they can solve, and sends these fragments to the executive module.
(3) The executive may optionally send solution fragments to the SC to check for constraint violations. The executive can then use this information to prune unacceptable paths from its search space.
(4) The executive module composes the solution fragments into one candidate solution.
(5) The SCLC verifies whether the final composed solution violates any of the constraints.

   (a) If violations are found, the violation details, including the degree of violation, are reported to the executive module, which then assigns blame to the agents. The process then returns to step 2 to continue the search for an acceptable solution to the problem.
   (b) If no violations are found, the solution is returned and the performance phase terminates.

In other words, the executive module decomposes each (practice or final) problem into an ordered set of sub-problems (where each sub-problem corresponds to one airspace conflict), sends these sub-problems to a blackboard for the three learning/reasoning agents to solve, and then composes the solutions from the agents. Each of the three agents inputs the same expert demonstration trace generated by an airspace manager, in addition to a jointly-shared airspace-deconfliction problem to learn on, and outputs a solution to whatever sub-problems it believes it can solve (where this belief is based on previous experience and previously stored self-knowledge), resulting in what we call a "plan fragment" or a "solution fragment" for solving a sub-problem. Note that multiple agents often tackle the same sub-problem(s).

Once the fragments of the deconfliction solution are generated by the three learning, planning, and reasoning agents and sent to the executive, the executive may optionally send them to the SCLC to check for constraint violations. This check is optional because the agents may have already used the constraints during their planning process. The executive then uses different heuristics in order to search to find the "best" solution (plan fragment) to each sub-problem, and it composes the sequence of best solutions into one overall, final solution (plan). In the context of the executive's search, "best" is defined as the executive's estimate (based on the demonstration trace and confidence values from the agents) of what would be most similar to a decision made by the human expert. While the executive module performs plan composition, it prunes unacceptable plan fragments from the search space, i.e., fragments that exceed the safety constraint violation threshold.

Ultimately, the executive sends the final composed plan to the SCLC to verify that the plan satisfies all of the safety constraints. Constraint satisfaction implies that the number and degree of constraint violations do not exceed a predetermined threshold. The SCLC outputs a *violation report* that includes the set of constraint violations for each proposed solution. The violation report is used by the executive module for assigning blame to the agents to use in re-planning and re-learning, and for guidance in its own plan composition process. As mentioned previously, the final plan must be checked in order to determine whether any violations were introduced by interactions between plan fragments.

Consider how GILA fits into a broader context. DARPA's purpose for initiating the GILA project was to explore learning and planning in the context of very few training examples. DARPA posed the extreme challenge of one training example in order to advance the state-of-the-art in machine learning, which typically assumes a large number of training examples. Mitchell *et al*'s LEAP system was one of the first to tackle the challenge of few examples – it used explanation-based learning to infer concepts from very few training examples and an extensive domain theory (Mitchell *et al.*, 1985). In a similar fashion, GILA maximizes the amount of information derived from a training example, but DARPA has further challenged GILA in two ways. First, GILA is only allowed one training example.

Second, its domain theory is minimized (to prevent over-engineering the solution). To meet these challenges, GILA exploits the single example and a minimal amount of domain knowledge by using its heterogeneous ensemble of learning/planning/reasoning agents and its SCLC. To the best of our understanding, GILA's challenges and its solution to these challenges are novel. GILA is also unique because although ensembles are very popular, and there have been very successful uses of bagging and boosting ensembles for classification learning (Polikar, 2006), GILA is the first ensemble system of which we are aware that has been applied in the context of complex problem solving. Because of the heterogeneous ensemble approach with complementary sub-systems, GILA has demonstrated that it performs as well or better than humans, after learning on the same training data (Zhang *et al.*, 2009). Furthermore, experimental results described in (Zhang *et al.*, 2009) demonstrate that the quality of the final plans produced by GILA's ensemble of all three agents is superior to that produced by the individual planners (when they are run independently without the other two agents, for experimental purposes). This confirms the value of our design methodology of choosing an ensemble of agents that are intended to be complementary and synergistic. For evaluating the entire GILA system, the performance of each plan, including its quality, is measured by a DARPA-assembled team of human airspace deconfliction experts, who give a score to each plan output by GILA. In this paper, we specifically evaluate the contribution of the SCLC – using our own evaluation methodology and performance metrics, as described in Section 6.

In the future, the DARPA BlueForce intends to use GILA to augment or replace human experts at the task of airspace management. Furthermore, we would like to apply it to different domains/problems. Its methodology, architecture and algorithms are sufficiently general for task transfer; only its ontology (including the specific constraints) would need to be replaced.

## 3. DEFINITIONS AND NOTATION

In the sections below, we will describe the SCLC architecture. However, before we are able to do that, some preliminary definitions and concepts need to be presented.

The SCLC learns and applies three types of constraints: *usage*, *shape*, and *airspace* constraints. A usage constraint specifies properties on a class of airspaces with the same mission (e.g., Combat Air Patrol (CAP)). A shape constraint (e.g., an air corridor (AIRCORR) or an Orbit) specifies properties on a class of airspace with the same airspace shape. Finally, an airspace constraint specifies properties on a specific airspace. Constraint templates are provided to the system, and the SCLC learns specific values to fill into the templates. For example, a constraint template might specify that an airspace has a maximum altitude, and the SCLC has to learn the value of this maximum.

Each constraint has a *bound* value, e.g., an upper bound value for an altitude constraint on an airspace, and a probability distribution associated with that bound, which represents the uncertainty associated with knowing the precise value. In our work, we use a dense, discrete set of *probability points* to approximate the continuous probability distribution. The probability at each point is inferred via Bayesian learning, described below. The Safety Checker then uses this set of probability points in its calculations of the *Expected Degree of Violation* (see below).

A constraint specifies a *lower bound*, or an *upper bound*, or both for the particular type of parameter for which it is defined. Typically, these bounds are used to provide an upper bound on the distribution for a maximum value, or to provide a lower bound on the distribution for a minimum value. Figure 3 illustrates this. Example 1 shows that given any ACM, when its ID is "F4," its maximum altitude should be no greater than the upper bound of 15000 feet. Example 2 shows that given any ACM, when its usage is "CAP," its maximum altitude should be no greater than the upper bound of 30000 feet and its minimum altitude should be no smaller than the lower bound of 10000 feet.

For the sake of clarity in our discussion on Bayesian learning, we use the following mathematical notation for our ontology concepts and the properties over those concepts. Let $\mathcal{C}$ be the set of all concepts defined for an airspace deconfliction problem. Let $\mathcal{F}$ be the set of all properties defined over those concepts. Formally, an object property is a function $f : \mathcal{C} \rightarrow \Re$. For example, if **F4** is a fighter, then its maximum altitude could be described as $f_{maxalt}(\mathbf{F4}) = 60000$.

Similarly, we also entertain binary domain relations $g : \mathcal{C} \times \mathcal{C} \rightarrow \Re$ that define properties on pairs of airspaces. For example, $g_{distance}(\mathbf{F4}, \mathbf{F15}) = 5$ represents that airspaces **F4** and **F15** are

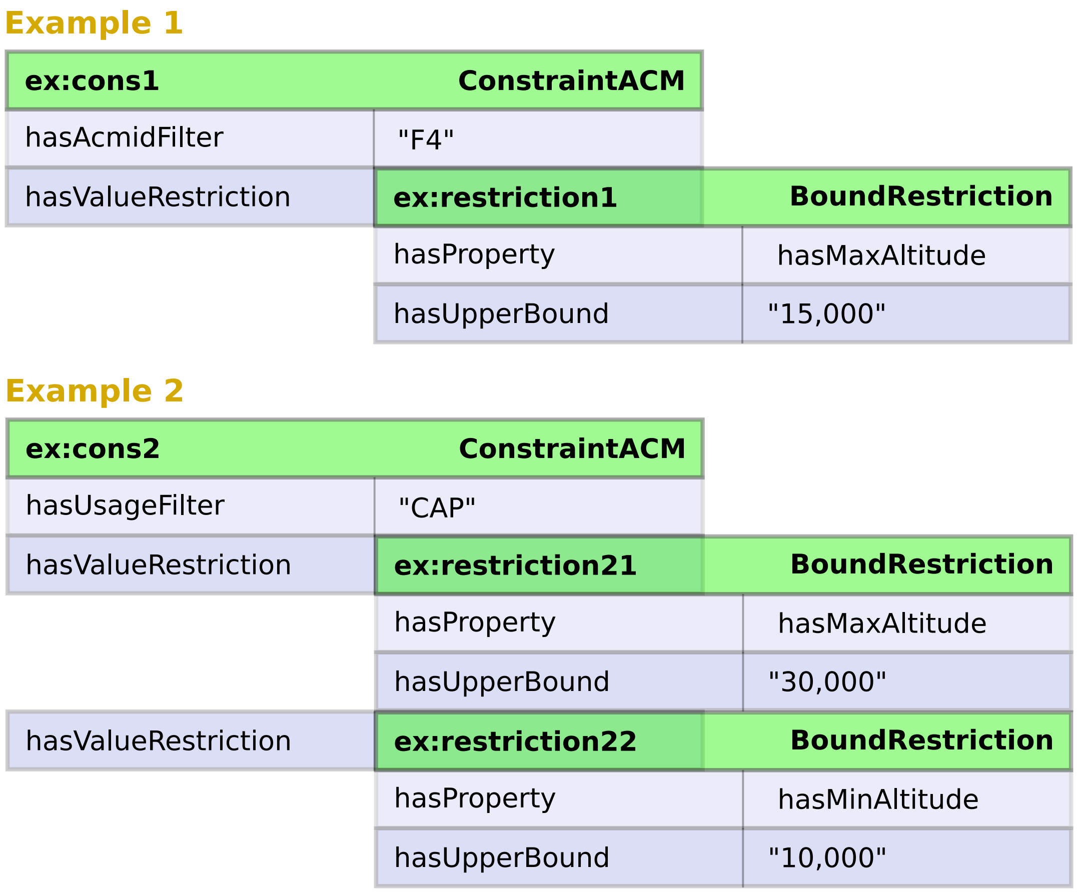


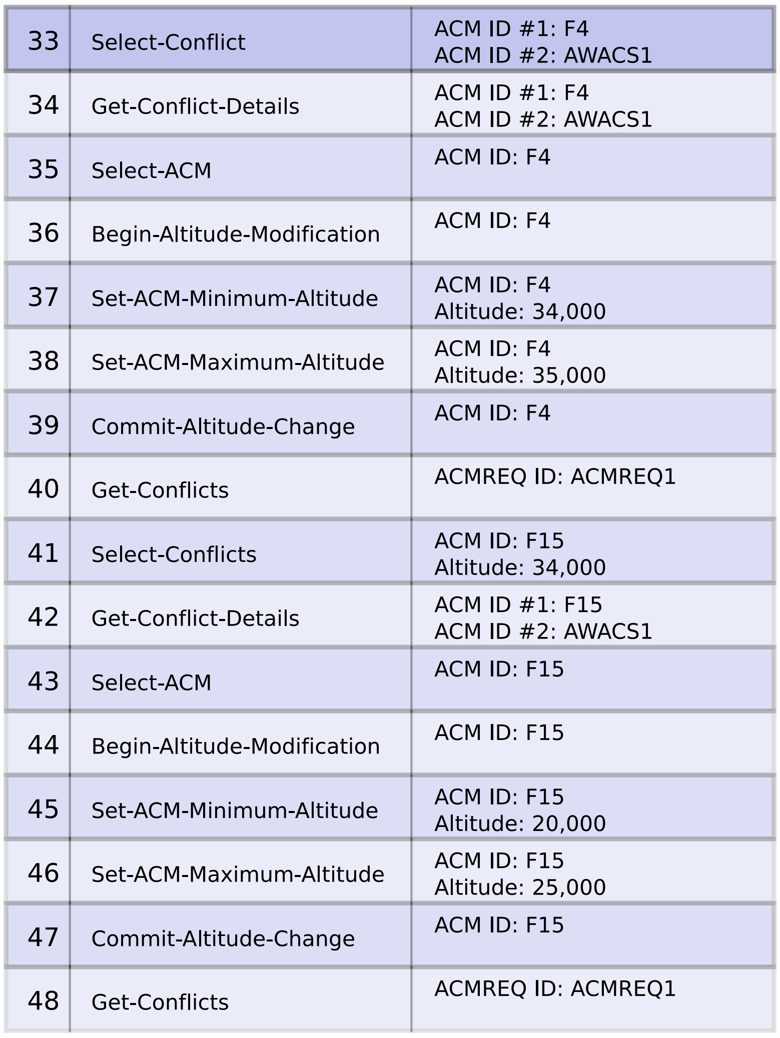
FIGURE 3. Example instances of safety constraints on lower and upper bounds of airspaces.

located 5 nautical miles from each other. Below, we show how these properties can be rephrased in the form of safety constraints. There is also background knowledge – in the form of facts. The background knowledge is minimized, according to DARPA's restrictions; however without any background knowledge at all, learning from a single expert demonstration trace would be impossible. The *domain theory* $D$ consists of the available concepts, the relations among them, and the set of all known facts.

A *learning element* is a tuple of either of the following forms: $(c, f)$ or $(c_1, c_2, g)$ where $f$ is an object property and $g$ is a binary domain relation. We call the former a *property learning element* and the latter a *relational learning element.*

Let $A$ be the set of all possible actions provided for the airspace deconfliction domain. A *demonstration trace* $T$ is defined as a sequence of actions provided by the expert. Figure 4 shows an example of a demonstration trace for deconflicting airspaces. Each row in the figure describes a particular action in this domain. For example, row 37 in the figure shows a "Set-ACM-Min-Altitude" action being applied to ACM F4. This action sets the minimum altitude of airspace F4 to 34000. Row 41 shows a selection action where the GILA system chooses the next conflict for deconfliction. The many different actions are used to change various properties of the airspaces. These actions form a basis for learning constraints.

Next, consider our constraint representation. Let $C$ be a domain concept, let $F$ be a set of properties over $C$, $G$ the set of binary relational properties (where we use $F$ and $G$ because they are functions), and let $T$ be a demonstration trace. A *safety constraint* is a triple of the form $(e, lb, ub)$, where $e$ is a learning element as previously defined. We define two types of safety constraints learned by the CL. If $e$ is a property learning element $(c, f)$ then $lb$ and $ub$ are respectively the lower and



| | | |
|---|---|---|
| 33 | Select-Conflict | ACM ID #1: F4<br>ACM ID #2: AWACS1 |
| 34 | Get-Conflict-Details | ACM ID #1: F4<br>ACM ID #2: AWACS1 |
| 35 | Select-ACM | ACM ID: F4 |
| 36 | Begin-Altitude-Modification | ACM ID: F4 |
| 37 | Set-ACM-Minimum-Altitude | ACM ID: F4<br>Altitude: 34,000 |
| 38 | Set-ACM-Maximum-Altitude | ACM ID: F4<br>Altitude: 35,000 |
| 39 | Commit-Altitude-Change | ACM ID: F4 |
| 40 | Get-Conflicts | ACMREQ ID: ACMREQ1 |
| 41 | Select-Conflicts | ACM ID: F15<br>Altitude: 34,000 |
| 42 | Get-Conflict-Details | ACM ID #1: F15<br>ACM ID #2: AWACS1 |
| 43 | Select-ACM | ACM ID: F15 |
| 44 | Begin-Altitude-Modification | ACM ID: F15 |
| 45 | Set-ACM-Minimum-Altitude | ACM ID: F15<br>Altitude: 20,000 |
| 46 | Set-ACM-Maximum-Altitude | ACM ID: F15<br>Altitude: 25,000 |
| 47 | Commit-Altitude-Change | ACM ID: F15 |
| 48 | Get-Conflicts | ACMREQ ID: ACMREQ1 |

FIGURE 4. An example demonstration trace for deconflicting airspaces.

upper bounds on the possible values of $f(c)$. In this case the triple $(e, lb, ub)$ is referred to as a *property safety constraint*. If, on the other hand, $e$ is a relational learning element $(c_1, c_2, g)$, then the triple $(e, lb, ub)$ is referred to as a *relational safety constraint* and $lb$ and $ub$ are lower and upper bounds on $g(c_1, c_2)$. When expanded, the property safety constraint appears as $(c, f, lb, ub)$ and the relational safety constraint becomes $(c_1, c_2, g, lb, ub)$.

As an example, suppose we wish to represent the lower and upper bounds on the minimum altitude of the fighter **F4** as 5000 and 10000 feet respectively. Formally this would be expressed as $(\mathbf{F4}, f_{minalt}, 5000, 10000)$. The lower and upper bounds delineate the lowest and highest values of property $f$ permissible according to the constraint. Finally, note that we need not use *both* the lower and upper bound slots in a constraint – only the one that is relevant, such as $lb$ but not $ub$ on $minalt$. Also, because we adopt a Bayesian approach to constraint learning, we actually produce posterior beliefs over the values of $lb$ and $ub$. For this reason, we represent $lb$ and $ub$ with probability distributions over property values.

Constraints can be learned simultaneously at two or more levels of abstraction, given in the domain ontology. For example, the aircraft types **F4** and **F15** can be abstracted to the class "fighter." Greater abstraction implies increased succinctness.

A *constraint database*, $\chi$, consists of a finite set of constraints. Given a demonstration trace $T$, $\chi$ is *admissible* if the following hold:

- For all property constraints $(c, f, lb, ub) \in \chi$, if $f(c)$ appears in $T$, $lb \leqslant f(c) \leqslant ub$, and
- For all relational constraints $(c_1, c_2, g, lb, ub) \in \chi$, if $g(c_1, c_2)$ appears in $T$, $lb \leqslant g(c_1, c_2) \leqslant ub$.

Note that admissibility implies consistency with respect to $T$, which is important because the main objective of GILA is to produce results that are similar to the results produced by human experts, as exemplified in $T$.

## 4. BAYESIAN LEARNING OF SAFETY CONSTRAINTS

Learning safety constraints of any arbitrary syntax and semantics is intractable. Therefore, we focus on a more tractable form of constraint learning. Constraint templates are provided by the system designer to the Constraint Learner (CL), and then it is the job of the CL to infer the values of parameters within these templates. For example, a constraint template might state that a fighter has a maximum allowable altitude, and then the learner would infer what the value of that maximum should be.

The CL learns an *admissible* constraint database from a given demonstration trace $T$. The CL notes every object in $C$ and its properties, $F$ and $G$, that appear in the trace. Entities appearing in the demonstration trace provide evidence of the existence of constraint bounds. This evidence is appreciated in a Bayesian way, resulting in constraint updates, e.g., increasing the upper bound on a maximum altitude or decreasing the lower bound on a minimum altitude. In the CL, we assume that bounds exist for all entities.

Before going into detail about the CL, we first clarify a point of potential confusion. The CL does *not* learn *absolute physical* constraints from an aircraft manual, e.g., the minimum or maximum possible flying altitude of an airplane. Such physical constraints are provided a priori as part of the background domain knowledge. It does, however, learn constraints that are "hard" in the sense that they can never be violated, but they are context-sensitive, where the "context" is the task mission as exemplified in the ACO. The expert trace in Figure 4, for example, shows the expert setting upper and lower altitude bounds, based on the context of the ACO, during the process of deconfliction. For instance, a recommended maximum altitude of an aircraft may be lowered if the scenario involves the threat of enemy surface-to-air missiles. In other words, the CL infers constraints that capture the notion that each entity (e.g., airspace) is embedded in a dynamic mission-oriented task; the constraints are therefore mission-specific.

### 4.1. Generating the Conditional Probabilities

Bayesian learning is a method in which observed evidence, $E$, is used to infer the probability of a hypothesis, $H$. For certain learning problems, it may be difficult to directly define the conditional probability of a hypothesis given the evidence, i.e., $P(H|E)$. However, after conditioning on a hypothesis, it is straightforward to define the probability of observing a certain piece of evidence, i.e., $P(H|E)$. We leverage this fact, along with Bayes' theorem, to describe $P(H|E)$:

$$P(H|E) \;=\; \frac{P(E|H) \cdot P(H)}{P(E)} \tag{1}$$

In the Bayesian framework, our learning algorithm proceeds as follows. As described previously, CL is given a set of domain objects $C = \{c_1, c_2, ...\}$, a set of object properties $F = \{f_1, f_2, ...\}$, $G = \{g_1, g_2, ...\}$ on the objects in $C$, and a demonstration trace $T$ made up of demonstrations of acceptable values for the individual domain properties, $\{f_r(c_i), f_s(c_j), g_u(c_k, c_l), ...\}$.

For example in our airspace-deconfliction trace from Figure 4, the set $C$ of objects includes an **F4**, an **F5**, and an **F15**. An example of an object property $f$ on **F4** is a bound restriction specifying a lower bound on the possible maximum altitude that **F4** should fly in the given context. The demonstration trace, shown in Figure 4, consists of examples of expert-generated modifications to the airspaces and their properties, such as Set-ACM-Minimum-Altitude of the **F15** to 20000 feet.

The objective of Bayesian Learning in SCLC is to learn the probability distribution over constraint databases, given the expert trace. For Bayesian learning, the CL represents the constraints as follows: $\chi = \{(e_1, P(e_1)), (e_2, P(e_2)), \ldots, (e_n, P(e_n))\}$, where $P(e_i), i = 1, 2, \ldots, n$ are discrete probability distributions over the values of properties and relations in $e_i$.

For simplicity, we assume that constraints corresponding to distinct objects and properties are independent. It would be possible to entertain more complex probability models where observations regarding one type of airspace will influence our opinion of other related airspaces. If constructed properly, these probability models could hasten learning. However, designing such models would be very domain knowledge intensive. For example, observing a reconnaissance airspace of an A-10 Thunderbolt aircraft could justifiably have a strong influence on our beliefs over the safety parameters for A-10 aircraft involved in escort, but less so for a refueling aircraft or unmanned aerial vehicles. Furthermore, in many practical cases, such domain knowledge may not be available due to the fact that experts are not available or are simply non-existent.[1] Therefore, instead of considering the relationship between the properties of all such aircraft, we chose to make the independence assumption stated above. With this assumption, we have:

$$P(\chi|T) = \prod_{c \in C, f \in F} P((c, f), P((c, f)))|T) \times \prod_{c_1 \in C, c_2 \in C, g \in G} P((c_1, c_2, g, P((c_1, c_2), g)))|T).$$

The above assumption also enables us to decompose the general problem into manageable learning sub-problems. Consider one such sub-problem, the case of learning the constraint $\omega_{i,maxalt} = (e_{i,maxalt}, P(e_{i,maxalt}))$, where $e_{i,maxalt} = (c_i, f_{maxalt})$, for the range of values of the maximum altitude associated with a particular airspace $c_i$. [2] Learning proceeds by witnessing evidence at each step $k$ of the demonstration trace. This evidence from the expert is in the form $f^k_{maxalt}(c_i)$. For example this might be a change in maximum altitude that occurs as the expert positions and repositions an airspace for the purpose of avoiding or removing conflicts. On line 38 of Figure 4, the expert sets the maximum altitude to 35000 feet. Then on line 46, he sets it to 25000 feet. Each of these pieces of evidence appears in the expert demonstration trace. The expert may change his mind as the context changes, and the trace reflects this. Bayesian learning from this trace proceeds as follows. Our prior belief over the value of $P(\omega_{i,maxalt})$, which is equivalent to the prior belief distribution $P(f_{maxalt})$, is represented by a distribution $P_{prior}(\omega_{i,maxalt})$. When we observe evidence from the expert, $f^k_{maxalt}(c_i)$, this prior distribution is updated to become the posterior $P(\omega_{i,maxalt}|f^k_{maxalt}(c_i))$.

## 4.2. Learning the Posterior Distributions

We estimate $P(\chi|T)$, the posterior probability of $\chi$, given the observed demonstration trace $T$. Bayes' rule implies that $P(\chi|T) = P(T|\chi) \times P(\chi)/P(T)$, where $P(T|\chi)$ represents the probability of observing demonstration trace $T$ given constraint database $\chi$, $P(\chi)$ is the prior belief over constraint databases from our domain theory, and $P(T)$ is the normalizing probability of observing demonstration trace $T$. For example, given the trace shown in Figure 4, the probability that the constraint database $\chi$ will contain a property such as "the minimum altitude for **F15** for this mission is 20000" is encoded in $P(\chi|T)$ above. The probability $P(T|\chi)$ specifies the probability of observing a trace $T$ in which "the minimum altitude for the **F15** is 20000."

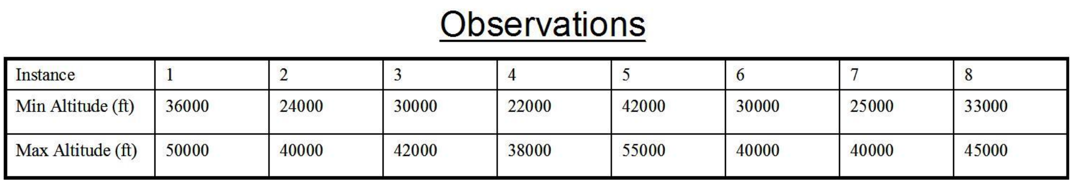
In general, applying Bayes' rule to find the posterior distribution, we have that:

$$P(\omega_{i,r}|f^k_r(c_i)) = \frac{P(f_r(c_i)|\omega_{i,r}) \times P_{prior}(\omega_{i,r})}{P(f_r(c_i))} \tag{2}$$

Here, $P_{prior}(\omega_{i,r})$ represents the prior distribution over property $f_r$ of airspace $c_i$, and $P(f_r(c_i)|\omega_{i,r})$ is the probability that the demonstration trace will contain a particular value of $f_r$ for object $c_i$, given

[1] Recall that DARPA restricted the amount and nature of domain knowledge allowed in GILA in order to see if GILA could learn effectively despite such restrictions.

[2] More precisely, if the property is the maximum altitude, then $P(e_{i,maxalt})$ is actually a probability distribution over the upper bound on *maxalt*. Likewise, if the property is the minimum altitude, then $P(e_{i,maxalt})$ is a distribution over the lower bound on *minalt*.



Observations

| Instance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Min Altitude (ft) | 36000 | 24000 | 30000 | 22000 | 42000 | 30000 | 25000 | 33000 |
| Max Altitude (ft) | 50000 | 40000 | 42000 | 38000 | 55000 | 40000 | 40000 | 45000 |

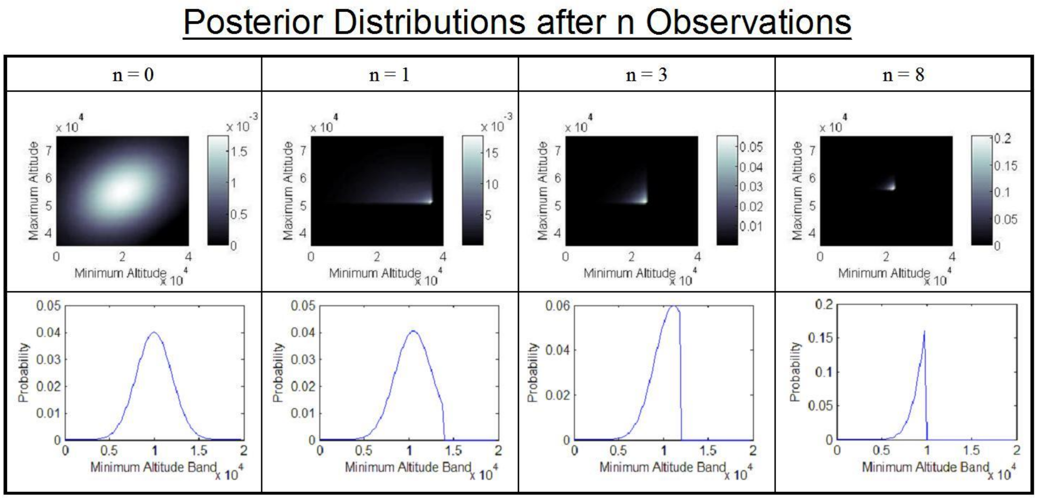


Figure 5. Bayesian Constraint Learning. The CL observes the airspace instances (top). Bayesian updates lead to tight posterior distributions over the safety constraint values (bottom).

the constraint $\omega_{i,r}$. These values can usually be defined using reasonable assumptions. $P_{prior}(\omega_{i,r})$ uses an uninformed uniform distribution over allowable values as a default. The uniform distribution assumption is the most common choice for situations where information is lacking. If available, the prior can be obtained from a Gaussian approximation of the real distribution by asking the expert for approximations of the average, variance, and covariance of the minimum and maximum altitudes.

$P(f_r(c_i)|\omega_{i,r})$ is the probability that an airspace will be placed at a certain altitude, given the altitude constraint. We assume that the expert (in the demonstration trace) always adheres to safety, and that the expert has no other inherent preferences for airspace altitude placement. Whereas in reality the airspace is placed at a particular location given the mission goals and interaction with other airspaces, inferring these intentions is out of the scope of the CL. Thus, we assume that the expert's placement of each airspace is with uniform probability within the safe altitude band.

This assumption has two nice properties. First, it enables the Bayesian update described above to assign a zero probability to any constraint that is inconsistent with the expert's airspace placements, because we know that the expert's actions are always safe. Thus, any constraint database generated by the CL is guaranteed to be admissible. Second, $P(f_r(c_i)|\omega_{i,r})$ is greater for more constraining sets of constraints. This means that if we observe an airspace repeatedly placed at or below 50000 feet, our confidence that the true maximum altitude constraint is near to 50000 feet (as opposed to much higher) will grow. Thus, more evidence will produce more confident posterior belief distributions, which will result in tighter constraints. This property has the effect that constraints can be learned even in the absence of negative examples. This is fortunate because negative examples are not provided in the demonstration trace.

## 4.3. An Example

An example of the Bayesian Constraint Learning appears in Figure 5. Consider some type of airspace for which there is a lower bound on the airspace's minimum altitude, an upper bound on the maximum altitude, and a lower bound on the altitude band of the airspace (the difference between the maximum altitude and minimum altitude). From the demonstration trace, the CL observes several expertly placed instances of the airspace type in order to learn the associated constraints. A broad Gaussian prior distribution over constraint values is used for n = 0 observations. After one observation, all constraint settings inconsistent with the observed placement are ruled out (for example, the minimum bound cannot be greater than the observed minimum altitude), but their posterior distributions are still fairly broad. However, at three and eight observations we witness a tightening of the posterior belief over constraint values due to the properties mentioned in the previous paragraph. Finally, note that Gaussian distributions can be summarized with their mean and standard deviation.

In summary, the final outcome of Bayesian learning is a set of constraints, where each constraint contains value restrictions represented by bounds, and each bound is associated with a discrete probability distribution representing the uncertainty over its possible values. Some of these probabilities are associated with values input from the original demonstration trace. Recall that each such probability is associated with a probability point. Probability points, as well as the CL-learned constraints, are used by the Safety Checker, described next. Finally, note that although minimum/maximum altitudes are used as illustrative examples in this paper, the SCLC in fact learns a wide variety of constraints.

## 5. SAFETY CHECKING LEARNED CONSTRAINTS

After constraints have been learned by the CL, they are used to verify (partial) solutions. In the context of GILA and airspace deconfliction problems, recall that a partial solution consists of actions that modify (i.e., move) airspaces in order to resolve conflicts among the airspaces in a given problem. GILA's planning and learning agents generate such partial solutions from a demonstration trace.

For example, consider the demonstration trace shown previously in Figure 4. Given this trace, the Symbolic Agent might solve the high-level task modify the altitude for a new problem using the same action sequence found in the demonstration trace: Select-ACM, Begin-Altitude-Modification, Set-ACM-Minimum-Altitude, Set-ACM-Maximum-Altitude, and Commit-Altitude-Change – see rows 35 to 39 in Figure 4. The other agents use the same trace to infer *alternative* partial solutions. The actions in these partial solutions should be checked in order to verify whether they violate any constraints. For example, if the Case-Based Agent sets the maximum altitude of the F4 to 40000 feet, then this change needs to be checked to see if it violates any constraints. The Safety Checker (SC) portion of the SCLC is responsible for this verification task. The SC also checks the final solution composed by the executive.

### 5.1. Safety Checker Overview

To illustrate the Safety Checker's procedure, we use the airspace deconfliction task. Each of the three agents in GILA has the job of proposing a sequence of deconfliction actions (which constitutes a candidate partial plan). These actions are intended to be applied to an ACO, which consists of spatio-temporal trajectories of all airspaces. To verify that the candidate partial deconfliction satisfies all safety constraints, the SC is invoked. The SC applies the sequence of deconfliction actions that constitutes a proposed solution. By applying this solution, the SC develops a hypothetical scenario – a modified ACO. The SC then uses its 4D Spatio-Temporal Reasoner to verify whether each constraint is still satisfied (preserved) 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 normalized to a value in the range [0.0, 1.0]. This severity is called the *Expected Degree of Violation (EDoV)*, and it is further described below. Specific information about the violation states which aircraft in the ACO caused the violation, e.g., fighter **F4**. The EDoV is used by the agents to rank violations. This ranking guides the agents so that they concentrate problem resolution on more severe violations first.

The three agents, Symbolic, Case-Based, and Decision-Theoretic, as well as the executive, use the violation reports from the SC for assigning blame and to repair and re-plan, until a solution is found that is violation-free or has an acceptable violation level.[3]

## 5.2. Inferring the Expected Degree of Violation (EDoV)

Recall that our Bayesian constraint learning approach results in a posterior probability distribution for each parameter value within a constraint template. For example, the CL's posterior belief over the upper bound on the maximum altitude of a fighter may be represented by a Gaussian distribution with a mean of 30000 feet and a standard deviation of 5000 feet. Let $P_f$ be the set of $<$ *value, probability* $>$ pairs (recall that these were called "probability points" in Section 3) representing the discretized distribution for constraint parameter $f$ learned and provided by the CL. Let $< x, p >$ be a pair that appears in $P_f$. We assume that $P_f(x)$ represents the probability $p$ of the value $x$ within the given pair from the discrete distribution.

The goal of the Safety Checker is to test whether the updated ACO (modified by the proposed solutions) still satisfies the safety constraints. To do this, it examines values in the newly-revised ACO and sees whether they are acceptable. To verify each change made in the hypothetically-revised ACO, the SC inputs the specific, instantiated constraint along with the distribution over the corresponding parameter value ($P(f)$), and the airspace position value seen in the revised ACO. Let $v$ denote an action taken in a solution proposed by one of the planning agents. Then the SC executes the following algorithm to calculate the EDoV for each value $v$:

(1) Consider a *simple constraint* $\omega = R$ (where $R$ is an instantiation of constraint $\omega$). These are constraints learned by the CL from a demonstration trace. As an example, suppose $\omega$ specifies a lower bound on the minimum altitiude of **F4**. One possible instantiation $R$ of $\omega$ specifies that the lower bound in question is 20000 feet for **F4**, which becomes the value of $x$. On the other hand, a proposed solution fragment might place an **F4** at 30000 feet, which is the value of $v$. The EDoV captures the expected error of the proposed solution.

We have the following definition for the Expected Degree of Violation for $R$:

$$EDoV_{unnormalized}(R) = \sum_x P_f(x) \cdot max(0, (x - v)) \quad (3)$$

for a minimum threshold and

$$EDoV_{unnormalized}(R) = \sum_x P_f(x) \cdot max(0, (v - x)) \quad (4)$$

for a maximum threshold. In general, for a value that is not a maximum or minimum, the expected error is:

$$EDoV_{unnormalized}(R) = \sum_x P_f(x) \cdot |x - v|. \quad (5)$$

(2) The SC also builds complex constraints from simple constraints using conjunctions or disjunctions. Currently we do not include negation in the constraints. For conjunctions within complex constraints, i.e., $(R_1 \cap \cdots \cap R_n)$ where each $R_i$ is a simple constraint, we have:

$$EDoV_{unnormalized}(R_1 \cap \cdots \cap R_n) = \left[\sum_{i=1}^{n} EDoV_{unnormalized}(R_i)\right] /n \quad (6)$$

and for disjunctions we have:

$$EDoV_{unnormalized}(R_1 \cup \cdots \cup R_n) = max_i \left[EDoV_{unnormalized}(R_i)\right]. \quad (7)$$

(3) Normalize this expected value to a number between 0 and 1. An assumed absolute (mechanical) maximum altitude $\alpha$ for all aerial vehicles is derived from limited domain knowledge available to GILA and to human subjects. $EDoV_{normalized} = \frac{EDoV_{unnormalized}}{|\alpha - \beta|}$, where $\beta$ is the most probable

[3]When the GILA system cannot find a solution in the allotted time or when the number of iterations exceeds a preset limit, it returns failure.

value of $x$ from the probability distribution. For a "maximum constraint" we add one standard deviation to this value and for a "minimum constraint" we subtract one standard deviation, allowing for noise. This value is output as the *Expected Degree of Violation (EDoV).*

To illustrate, consider the following example. Suppose we wish to test the Bayesian-derived constraint $(e = (\mathbf{F4},\ f_{minalt}),\ P(e_{F4,minalt}))$. After learning, the posterior probability distribution $P(e_{F4,minalt})$ has the following discrete probability points:

- $p(F4, 36000) = 0.125$
- $p(F4, 30000) = 0.250$
- $p(F4, 24000) = 0.125$
- $\ldots$

Furthermore, suppose that the altitude found in the modified (by a partial plan) ACO is 34000. Then we can calculate the EDoV as:

$$0.125 \cdot max(0, 36000 - 34000)\ +\ 0.250 \cdot max(0, 30000 - 34000)\ +$$

$$0.125 \cdot max(0, 24000 - 34000)\ +\ \ldots\ =\ 250.$$

The value 250 is normalized to a number between 0 and 1, in this case 0.041, and is output by the SC as the EDoV of the constraint associated with the given proposed solution. Note that this methodology is identical for relational safety constraints.

During plan composition, the executive module compares each violation's EDoV to the safety violation threshold, discarding any plan fragments that exceed that threshold, thereby pruning and reducing the size of the search space. The threshold value is set by subject matter experts.

## 6. EXPERIMENTAL RESULTS

In a recent experiment performed by the DARPA BlueForce that compared the quality of deconfliction solutions generated by GILA against the quality of solutions generated by novice human airspace managers, GILA was able to perform slightly better than the average. In particular, when GILA was compared against twelve human novices, the mean score (out of 100%) for the humans was 90.6% and the mean score for GILA was 92%. For fairness of comparison, the humans and GILA received the same background knowledge and inputs, according to careful and rigorous measurements made by the BlueForce. The quality metric included considerations such as the number of solution steps that differed between the expert and the novice/GILA. For the details of these experiments on GILA using SCLC, see (Zhang *et al.*, 2009).

We performed further experiments to better understand the impact of the SCLC within GILA. More specifically, we did an experimental investigation of the following experimental hypothesis:

**HYPOTHESIS:** GILA *with* the SCLC generates airspace-deconfliction steps that are more similar to those of the expert than GILA *without* the SCLC.

Again, fairness was ensured by having the original GILA (with the SCLC) and the ablated version use identical inputs. Three airspace deconfliction scenarios were employed. For each scenario, we used a demonstration trace generated by an expert airspace manager from the DARPA BlueForce. These traces are similar to the demonstration trace example shown previously in Figure 4. In a typical run, about 20 to 30 new airspaces were added to the original ACO. This resulted in 10 to 15 conflicts in the scenario.

We performed a suite of cross-validation experiments using several combinations of these scenarios. In each combination, one scenario and its corresponding demonstration trace was designated for learning, and another scenario was used as the target problem that GILA needed to solve with the learned knowledge. In order to determine the quality of our solution, we compared it to the demonstration trace associated with the target problem. For each case, we ran GILA with and without the SCLC, thereby allowing us to evaluate the impact of constraint learning/enforcement on the overall GILA system behavior.

For each scenario, GILA had to choose which airspaces should be moved, and in what manner they should be moved, to eliminate spatio-temporal conflicts. This led to the following two performance metrics for our experiments in order to test our hypothesis above:

- *Metric 1:* We compared all airspaces moved by GILA and the expert by grouping them as *true positives*, i.e., those moves performed by both the GILA and the expert, *false positives*, i.e., those moves that were only done by GILA but not the expert, and *false negatives*, i.e., those that were done by the expert but not by GILA.
- *Metric 2:* We compared all (airspace, type) move pairs done by GILA and the expert. Type can be altitude, time, or geometry. For example, a move pair can be as (F4, Altitude), which means that the move changes the altitude of the airspace F4. As a performance score, we measured how many pairs were in agreement between GILA and the expert. This is a more specific measure of how much GILA is in agreement with the expert.

The score of GILA, with versus without the SCLC, is given by the following formula:

$$\frac{TP}{TP + FP + FN},$$

where $TP$, $FP$, and $FN$ are the number of true positives, false positives, and false negatives in an experiment, respectively. The maximum possible score is 1.0, corresponding to complete agreement between GILA and the expert. The lowest score, 0.0, occurs when GILA and the expert choose completely disjoint sets of airspace modifications.

Across all of our five experimental cases, the system generated the following results with the SCLC: $TP = 30$, $FP = 18$, and $FN = 22$. Based on this outcome, GILA's score using the first metric was 0.429 when the SCLC was included. The score of the system dropped to 0.375 when the SCLC was excluded, with the following results: $TP = 27$, $FP = 20$, and $FN = 25$.

Using the second metric, the relative scores were 0.293 and 0.265 when GILA included or excluded the SCLC, respectively. These results suggest the value added by including the SCLC: GILA was able to perform more similarly to the expert by learning safety constraints and checking the solutions generated by the system against those constraints. This suggests that the learning and enforcement of safety constraints helps GILA choose correct deconfliction actions, by penalizing potentially unsafe actions that would be entertained otherwise.

## 7. RELATED WORK

We first address research related to the Constraint Learner. 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 *et al.*, 1985). Similarly, (Shavlik, 1985) shows how the general physics principle of Momentum Conservation can be acquired through the explanation of a "cancellation 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 *et al.*, 2006), and to learning autonomous control of helicopter flight (Abbeel and Ng, 2005).

Our learning framework, when seen in the context of multiple agents, 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 architecture is quite different. These architectures are directed toward integration in a psychologically plausible way, but their mechanisms are more centralized. Unlike these other cognitive architectures, GILA does not have a single, homogeneous, unifying computational mechanism, and it does not require sharing of common representations. GILA is more of a diverse, ensemble approach to modeling the acquisition and application of domain expertise.

Our research is also strongly related to learning control rules for search/planning. This area has a long history, e.g., see (Minton and Carbonell, 1987), and has more recently evolved into the learning of constraints (Huang *et al.*, 2000) for constraint-satisfaction planning (Kautz and Selman, 1999).

Next, we address research related to the Safety Checker. One area of related work is on "safe planning." 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 and Etzioni, 1994; Gordon, 2000). Our safety constraints have a similar motivation.

The Safety Checker is also related to formal verification, such as model checking (Clarke *et al.*, 1999). However our work has a more novel twist that is more akin to the recent work by Chockler and Halpern, in which formal verification is extended to include a "*degree* of blame" (Chockler and Halpern, 2004). Chockler and Halpern present a theoretical framework for the degree of blame in relation to an agent's epistemic state. Unlike traditional models that treat causality as binary (true/false), Chockler and Halpern take into account a more refined *expected degree* of responsibility, taken over the epistemic state of an agent. The SC's EDoV is also an expected degree of responsibility in the sense that expectation is taken over the violation and is then used by the executive to assign blame to individual agents. The novelty of our SC approach is its application of expected degree of responsibility in the context of a multi-agent system.

A final novelty of both the CL and the SC is that they are learning and applying constraints in an unusual context of very few examples, a minimized domain theory, and multiple learners. This is in sharp contrast to the traditional learning paradigm that consists of one learner and many training examples, or few examples but a rich domain theory. Our context poses an enormous challenge, i.e., that of maximizing the information gleaned from a minimal amount of data and little background knowledge. The heterogeneity of the ensemble of agents helps in this respect because the bias of each agent can counteract the biases of the others. Another way that our approach maximizes its gain from little data is by simultaneously learning at multiple levels of abstraction. The GILA results demonstrate the value of our approach.

## 8. CONCLUSIONS AND FUTURE WORK

This paper has described a new framework for learning and applying safety constraints in an important, real-world airspace deconfliction problem. Learning occurs from observation of 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 problem-solving knowledge. An implementation of our approach in a multi-agent system developed for the DARPA Integrated Learning Program demonstrated its effectiveness at facilitating the production of safe plans.

The next step in this research will be to extract the SCLC from GILA so that it can run as a standalone module. This will enable us to run extensive experiments to test hypotheses about our constraint methodology, and it can lead to further algorithmic improvements. Another future direction will be to reason explicitly about the system's biases and the incompleteness of its knowledge. An example of the former would be if the executive were to collect a history of the types of conflicts that each agent is good/poor at solving and then use this history to prioritize which agent(s) to suggest first for solving each new conflict. An example of the latter would be to adopt the approach of (Garland and Lesh, 2002), which identifies and mitigates the effects of knowledge incompleteness.

## 9. ACKNOWLEDGMENTS

This work was 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 DARPA. Special thanks go to Dutch Van Sloten and others on the DARPA Blueforce for helping us to understand and appreciate the job of airspace planning and deconfliction.

## REFERENCES

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, **7**(1), 39–59.

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

Botea, A., Enzenberger, M., Mueller, M., and Schaeffer, J. (2005). Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, **24**, 581–621.

Chockler, H. and Halpern, J. Y. (2004). Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, **22**, 93–115.

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

Garland, A. and Lesh, N. (2002). Plan evaluation with incomplete action descriptions. In *AAAI/IAAI*, pages 461–467.

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)*, pages 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)*, pages 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)*, pages 228–235.

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

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

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, **6**(3), 21–45.

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

Spears, W. M. and Gordon, D. F. (2000). Evolution of strategies for resource protection problems. In *in Advances in evolutionary computing: theory and applications*, pages 367–392. Springer Verlag.

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

Xuan, P. and Lesser, V. (2002). Multi-agent policies: from centralized ones to decentralized ones. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1098–1105, New York, NY, USA. ACM.

Yoon, S. and Kambhampati, S. (2007). Hierarchical strategy learning with hybrid representations.

Zhang, X., Yoon, S., DiBona, P., Appling, D., Ding, L., Doppa, J. R., Green, D., Guo, J., Kuter, U., Levine, G., MacTavish, R., McFarlane, D., Michaelis, J., Mostafa, H., Ontanon, S., Parker, C., Radhakrishnan, J., Rebguns, A., Song, Z., Trewhitt, E., Zafar, H., Zhang, C., Corkill, D., DeJong, G., Dietterich, T., Kambhampati, S., Lesser, V., McGuinness, D., Ram, A., Spears, D., Tadepalli, P., Whitaker, E., Wong, W.-K., Hendler, J., Hofmann, M., and Whitebread, K. (2009). An ensemble learning and problem solving architecture for airspace management. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference (IAAI)*.