Inferring and Applying Safety Constraints to Guide an Ensemble of Planners for Airspace Deconfliction

Antons Rebguns[*] and **Derek Green**[*] and **Geoffrey Levine**[†]

Ugur Kuter[‡] and **Diana Spears**[*]

Abstract

This paper presents a Bayesian approach to learning flexible safety constraints in a coordinated, multi-planner ensemble, along with stochastic and active experimentation approaches for assigning degrees of blame when these constraints are violated. The blame is subsequently translated and conveyed to planners, for the purpose of improved overall system performance. To illustrate the advantages of our framework, we provide and discuss examples on a real test application for Airspace Control Order (ACO) planning and deconfliction, which is a benchmark application in the DARPA Integrated Learning Program.

Introduction

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-sensitive situations. As an example, in the research described herein we tackle the Airspace Planning and Deconfliction task that is crucial to many civilian and military airspace applications, and is being used by DARPA as a benchmark application. This task is normally performed by a human expert, called the *airspace manager*. The task involves positioning each airspace trajectory so that there will be no spatio-temporal conflicts, and so that safety constraints such as maximum altitudes are not violated. The term "deconfliction" means setting the trajectories of two entities, such as aircraft, so that they do not intersect in space and time, i.e., to prevent a crash.

Such real-world applications require a variety of capabilities, e.g., planning under uncertainty and time, learning knowledge for planning, reasoning about world dynamics, and coordinating all of the above in an integrated AI system. One approach to addressing such a problem is to develop a single, centralized system that provides all of these capabilities. However, monolithic centralized systems tend to be too brittle and complex, and are notoriously difficult to debug. Therefore, we have adopted the approach of developing a distributed AI system that consists of loosely-coupled learner/planner components. Although these components learn to improve their planning, for succinctness hereafter we simply call them "planners." When executed as an ensemble, these components possess the capabilities mentioned above for finding solutions to challenging real-world applications. Furthermore, the loosely-coupled ensemble architecture leads to increasing robustness and the potential for synergy between components.

The use of an ensemble solves the problems of robustness and real-world capabilities, but it introduces a new challenge. In particular, due to the typical paucity of domain planning knowledge in realistic scenarios, generating a course of action (also called a "solution" or, equivalently, a "plan") via such a group of loosely-coupled, automated planners is challenging. Fortunately, the use of constraints can ameliorate this situation to a large extent. Therefore, the bulk of our research has been focused on the learning and application of constraints in the context of an ensemble of planners.

The focus of this paper is on our approach, called the *Safety Constraint Learner/Checker (SCLC)*, for automated learning and application of planning knowledge in the form of safety constraints. Examples of such safety constraints include the following: "The altitude of an aircraft over the course of its trajectory should never exceed its maximum," and "An aircraft trajectory should never be moved so that it intersects a no-fly zone."

SCLC consists of a learner and a checker, as follows:

1. The SCLC's *Constraint Learner (CL)* automatically infers safety constraints from problem-solving demonstrations in a scenario. A *demonstration trace* consists of a sequence of actions and/or decisions taken by an expert in solving a planning problem. By analyzing such traces of expert behavior for evidence of certain features in the scenarios, CL can help planners to more closely mimic the expert's behavior. In particular, CL uses a Bayesian learning technique to generate safety constraints about the scenario (i.e., it generates constraints on the acceptable behaviors in the scenario that, if violated, are guar-

[*]University of Wyoming, Computer Science Department, Laramie, WY 82071, USA. Author Emails: {anton,derekg,dspears}@cs.uwyo.edu

[†]University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, IL 61801, USA. Author Email: levine@uiuc.edu

[‡]University of Maryland, Institute for Advanced Computer Studies, College Park, MD 20742, USA. Author Email: ukuter@cs.umd.edu

anteed to cause incorrect solutions to the input problems). The planners use these safety constraints during planning. These constraints are also used by the SC, described next.

2. The SCLC's *Safety Checker (SC)* is responsible for verifying the correctness of plans/solutions in terms of their satisfaction or violation of the safety constraints. The SC inputs solutions from component planners, along with safety constraints output by the CL, and it outputs a degree of constraint violation and an assignment of blame. Both of these together may be used as diagnostic feedback in a learning/planning system for the purpose of proposing improved solutions.

At this point, the reader may be confused as to why the SC is needed if the planners already use the safety constraints during planning. The reason is that the planners might only output *partial* solutions/plans. These partial solutions are then composed (by an executive module) into one solution/plan, and that is what needs to be checked by the SC.

We have implemented the SCLC as a component in the *Generalized Integrated Learning Architecture (GILA)*, which is an ensemble planning system being developed as part of a large team effort, and funded by DARPA. GILA consists of an ensemble of three planners, the SCLC, and an executive module; these components of GILA are loosely coupled in an overall integrated system architecture. A highly simplified version of the GILA algorithm relevant to this paper is:

1. GILA takes as input an expert-generated demonstration trace, which is a training example of expert behavior. It also inputs a deconfliction scenario for learning.
2. The CL learns safety constraints from the trace.
3. Each of the three planners infers a solution fragment, using the demonstration trace, safety constraints, and learning scenario.
4. The executive module composes the solution fragments into one *general* proposed/candidate solution, which is applied to a new deconfliction scenario.
5. For the new scenario, the SC checks whether the proposed solution violates any of the constraints. If yes, then repeat starting at Step 3 using information output by the SC to guide the search for an acceptable solution to the new problem.

Although the SCLC approach is applied in the GILA project to an airspace planning and deconfliction task, it is *not* specific to this one task. It is sufficiently general to be applicable to a wide variety of planning tasks, and future work will focus on other tasks. The constraints are problem-specific, but the methodologies for learning and checking them are general. Nevertheless, this paper focuses on the airspace task, in both its illustrative examples and its experimental results. We now describe the task.

Implementation in a Real-World Airspace Deconfliction Task

The Airspace Planning and Deconfliction Task is normally executed by a human expert – the airspace manager. The airspace manager works with an *Airspace Control Order (ACO)*, which specifies all the locations and trajectories of airspace objects, and a set of *Airspace Control Measure Requests (ACMReqs)*, which specify the locations and trajectories of new airspace objects that need to be added to the original ACO. An example ACO is in Figure 1. The airspace manager starts with the initial ACO, and then applies the ACMReqs to the ACO in a way that does not cause any 4D spatio-temporal conflicts or constraint violations. This involves positioning each *Airspace Control Measure (ACM)* so that these goals are satisfied. An ACM is a trajectory for an aircraft, tanker, missile, or other military entity. Rephrased in terms of ACMs, an ACMReq is a request to include another ACM in the ACO.

The overall task objective is to derive a solution/plan to include the ACMReqs in the ACO, while deconflicting all entities and satisfying the safety constraints. In other words, there are three subtasks: planning, deconfliction, and constraint satisfaction.

DARPA's GILA project aims to develop an ensemble of integrated AI systems in order to automate the airspace manager's task. GILA and a human airspace manager get identical inputs; this includes an example problem with a corresponding demonstration trace, but *only one*. This is a small hint without which the automation would not be possible. The objective of the SCLC within GILA is to automate the constraint satisfaction subtask. The planners, on the other hand, derive partial solutions to the planning and deconfliction problems, subject to the safety constraints. An airspace manager's task 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. They say that in some respects it is analogous to solving a huge jigsaw puzzle. Nevertheless, with a puzzle each piece has one correct location. With airspaces, on the other hand, there are multiple good (or even optimal) solutions. This complicates the problem, thereby making it exceptionally difficult to automate. 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 is a considerable accomplishment. The SCLC appears to have played a notable role in this success, as described in the experimental results section below.

Preliminaries

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 here.

For effective learning of safety constraints for planning, we assume the existence of a *domain ontology*, O, that specifies conceptual knowledge about the underlying problem domain and the relationships within that domain. The domain ontology specifies *object classes* and their properties in the world. An object class, C, defines a set of entities that belong together because they share a domain property. An example object class is the set of all fighter planes. An example domain property could be the maximum altitude that a fighter plane is allowed to fly.

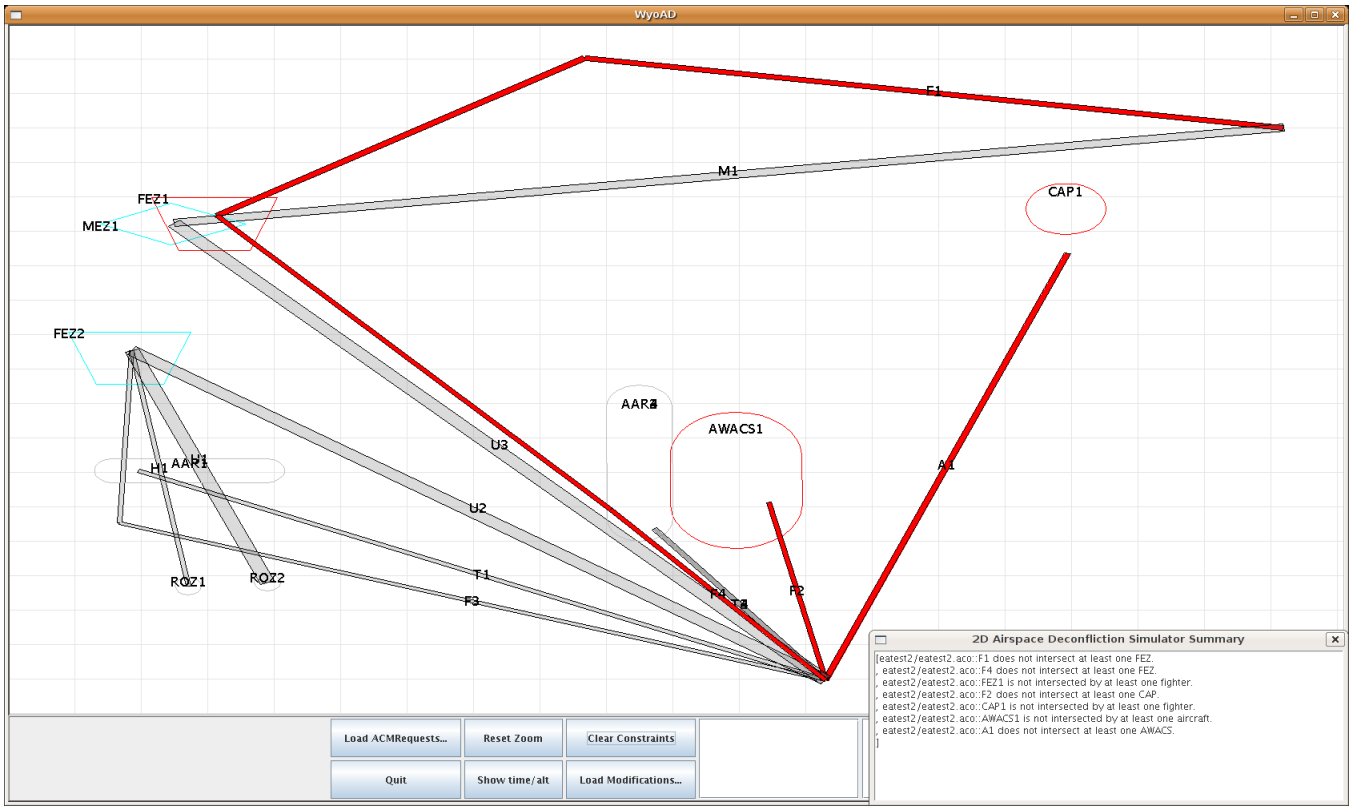

Figure 1: An example of an ACO deconfliction problem instance.

```
33  Select-Conflict  ACM ID #1: F4 ACM ID #2: AWACS1
34  Get-Conflict-Details  ACM ID #1: F4 ACM ID #2: AWACS1
35  Select-ACM  ACM ID: F4
36  Begin-Altitude-Modification ACM ID: F4 Usage: AIRCORR
37  Set-ACM-Min-Altitude  ACM ID: F4 Altitude: 34000
38  Set-ACM-Max-Altitude ACM ID: F4 Altitude: 35000
39  Commit-Altitude-Change  ACM ID: F4
40  Get-Conflicts ACMREQ ID: ACMREQ1  ACMConflictList(ACMConflict("AWACS1", "F5"))
41  Select-Conflict  ACM ID #1: F5 ACM ID #2: AWACS1
42  Get-Conflict-Details  ACM ID #1: F5 ACM ID #2: AWACS1
43  Select-ACM  ACM ID: F5
44  Begin-Altitude-Modification ACM ID: F5 Usage: AIRCORR
45  Set-ACM-Min-Altitude  ACM ID: F5 Altitude: 20000
46  Set-ACM-Max-Altitude ACM ID: F5 Altitude: 25000
47  Commit-Altitude-Change  ACM ID: F5
48  Get-Conflicts ACMREQ ID: ACMREQ1  ACMConflictList()
```

Figure 2: An example demonstration trace for deconflicting airspaces.

Formally, an object property is a function $f : C \rightarrow \Re$. For example, if **F1** is a fighter, then its maximum altitude could be described as $f_{maxalt}(\textbf{F1}) = 60,000$ ft. Similarly, we also entertain binary domain relations $g : C \times C \rightarrow \Re$ that define properties on pairs of airspaces. For example, $g_{distance}(\textbf{F1}, \textbf{F2}) = 5$ nm, represents that airspaces **F1** and **F2** are located 5 nautical miles from each other. Below, we show how these properties can be rephrased in the form of safety constraints. In addition to constraints, there is also background knowledge in the form of facts. The *domain theory*, D, consists of both O and the set of all known facts.

Let A be the set of all possible actions in the planning domain. An action might be setting a maximum altitude, for example. A *demonstration trace* T is defined as a sequence of actions. Figure 2 shows an example of a demonstration trace for deconflicting airspaces.

Let C be an object class, let F be a set of object properties, G the set of binary relational properties (where we use F and G because they are functions), and let T be a demonstration trace. We define two types of *safety constraints*. An *object (safety) constraint* as a four-tuple of the form (c, f, lb, ub), where c is an object in C, f is a unitary property in F, and lb and ub are the lower and the upper bounds on the possible values of $f(c)$. Similarly, a *relational (safety) constraint* is a five-tuple, (c_1, c_2, g, lb, ub), where c_1 and c_2 are in C, g is a binary relation in G, and lb and ub are lower and upper bounds on $g(c_1, c_2)$.

As an example, suppose we wish to represent the lower and upper bounds on the minimum altitude of the fighter **F1** as 5,000 and 10,000 feet respectively. Formally this would be expressed as (**F1**,$f_{minalt}, 5000, 10000$). The lower and upper bounds delineate the lowest and highest (respectively) 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.

The domain ontology describes an abstraction hierarchy over the object classes in the domain. For example, **F1** and **F2** can be abstracted to the class "fighter." Constraints can be learned simultaneously at two or more levels of abstraction. Greater abstraction implies increased succinctness. Therefore, the more abstract version of a constraint may be preferable for use when it exists.

A *constraint database*, χ, consists of a finite set of constraints. Given a demonstration trace T, χ is *admissible* if the following holds: for all $(c, f, lb, ub) \in \chi$, if $f(c)$ appears in T, $lb \leq f(c) \leq ub$ and for all $(c_1, c_2, g, lb, ub) \in \chi$ if $g(c_1, c_2)$ appears in T, $lb \leq g(c_1, c_2) \leq ub$.

A *constraint learning problem* is a tuple (D, T, C, F, G), where D is the domain theory, T is a demonstration trace, C is an object class from O, and F and G are finite sets of object and relational domain properties on the objects in C. Our notion of a *constraint database* χ that solves the constraint learning problem is one that specifies an admissible constraint on the value of each object property in F associated with each object in C, and relational property in G associated with each object pair in $C \times C$ that appears in the demonstration trace T.

Bayesian Learning of Safety Constraints

Learning safety constraints can be quite challenging in general. Here, we focus on learning the parameter values for the constraints. Constraint templates are provided by the system designer a priori, and it is the job of the learner to infer the values of parameters within these templates.[1] 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 inference is based on examples of maximum altitudes in the demonstration trace.

This section presents our method, called Constraint Learner (CL), for learning an admissible constraint database for a given learning problem. The CL takes as input a learning problem (D, T, C, F, G) and returns a constraint database χ for that problem. The demonstration trace T contains specific deconfliction and constraint enforcement decisions that an expert has made on a set of objects and their properties. CL notes every object in C and their properties, F and G, that appear in the trace. Entities appearing in the demonstration trace provide evidence on 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.

Learning Object and Relational Constraints. In the Bayesian framework, learning 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, $\{f_r(c_i), f_s(c_j), g_u(c_k, c_l), ...\}$. The goal is to learn an admissible constraint database $\chi = \{(c_i, f_r, P(f_r)), (c_j, f_s, P(f_s)), (c_k, c_l, g_u, P(g_u)), ...\}$, where $P(f_r)$ and $P(f_s)$ are discrete probability distributions over the values of properties f_r and f_s. This is the Bayesian representation of constraints.

We estimate $P(\chi|T)$, the posterior probability of χ, 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 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 objects and properties are independent, we have $P(\chi|T) = \prod_{c \in C, f \in F} P((c, f, P(f))|T) \times \prod_{c_1 \in C, c_2 \in C, g \in G} P((c_1, c_2, g, P(f))|T)$.

The above assumption enables us to decompose the general problem into manageable subproblems. Consider one such subproblem, the case of learning the constraint $\omega_{i,maxalt} = (c_i, f_{maxalt}, P(f_{maxalt}))$ 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. Bayesian learning 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 $P(\omega_{i,maxalt}|f^k_{maxalt}(c_i))$.

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

$$P(\omega_{i,r}|f^k_r(c_i)) = \frac{P(f_r(c_i)|\omega_{i,r}) \times P_{prior}(\omega_{i,r})}{P(f_r(c_i))} \quad (1)$$

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 the constraint $\omega_{i,r}$.

These values can usually be defined using reasonable assumptions. For instance, in the context of our airspace deconfliction example, $P_{prior}(\omega_{i,r})$ 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.

$P(f_r(c_i)|\omega_{i,r})$ 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 puts the airspace with uniform probability within the safe

[1] In the future, CL will learn the templates.

[2] More precisely, if the property is the maximum altitude, then $P(f_{maxalt})$ is actually a probability distribution over the upper bound on $maxalt$. Likewise, if the property is the minimum altitude, then $P(f_{minalt})$ is a distribution over the lower bound on $minalt$. We omit this technical detail in the paper for the sake of succinctness and readability.

region of placements. This assumption enables us to start learning without any a priori background knowledge about the expert's behavior and the underlying domain. Furthermore, 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 expert's actions are always correct. 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 restrictive sets of constraints. This means that if we observe an airspace repeatedly placed at or below 50,000 feet, our confidence that the true maximum altitude constraint is near to 50,000 feet (as opposed to much higher) will grow. Thus, more evidence will produce more confident posterior belief distributions, which will result in tighter constraints.

Note that the safety constraints mimic the expert's behavior, but they may or may not reflect the true bounds. Since the CL's only knowledge is the demonstration trace, this is the best the CL can do. Fortunately, the trace typically provides conservative bounds with respect to the true bounds, though this may not always be the case.

Safety Checking Learned Constraints

After the constraints have been learned, they are then used by the planners to generate constrained candidate partial plans. These partial plans are then composed into one proposed solution. Because the candidate partial plans are generated by multiple independent components (planners), they may be inconsistent. For this reason, the Safety Checker (SC) portion of the SCLC is called to verify that the proposed solution obeys the learned safety constraints. Any constraint violations are reported back to the planners for remediation.

As an example, consider the situation where each component planner has the job of proposing a sequence of deconfliction actions (which constitute 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 plan satisfies all safety constraints, the SC is invoked. The SC applies the sequence of steps/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 or not. Any violations are reported for evaluation. Violation reports include the violated constraint, specific information about the violation, optional advice for plan repair, blame assignment (all deconfliction actions involved will receive a share of blame), and the degree (severity) of violation normalized to a value in the range [0.0, 1.0]. Specific information about the violation states 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 severe violations. It also makes it possible to ignore less severe violations in the event that no completely safe plan is discovered within the allotted time.

The other planners use this feedback from the SC to repair and re-plan, until a solution is found that is violation-free or has an acceptable violation level. Figure 1, shown previously to highlight an example of an ACO instance, is taken as a screen shot of our 4D Spatio-Temporal Reasoner performing safety checking on the shown ACO.

Inferring the Expected Degree of Violation

Recall that our Bayesian constraint learning approach results in a probability distribution over the parameter values for a constraint template. For example, it might learn that the maximum altitude of a fighter is best represented as a Gaussian distribution with a mean of 30,000. The CL also records the set of all specific maximum altitudes that were obtained from the demonstration trace. We call this set A_{DT}. Recall from above that an example of an expert-generated maximum altitude can be represented as $f^k_{maxalt}(c_i)$. 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 accomplish this verification of the revised ACO, the SC inputs the specific, instantiated constraint along with the distributions over the parameter values (such as $P(f_{maxalt})$), the set A_{DT}, and the value seen in the ACO. It then executes the following algorithm for simple constraints:

1. Take the difference between the value in the ACO being tested and each of the expert-generated (from the demonstration trace) values $f^k_r(c_i)$ in A_{DT}.
2. Use the probability of each value in A_{DT} (found from the posterior distribution $P(f_r)$) to weight the differences.
3. Calculate the expected value over all the differences.
4. Normalize this expected value to a number between 0 and 1. 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 (**F1**, f_{maxalt}, $P(f_{maxalt})$). We use the current probability distribution $P(f_{maxalt})$ and the values from the demonstration trace collected in A_{DT} such as $f^3_{maxalt} = 38,000$, $f^5_{maxalt} = 37,000$, and $f^{16}_{maxalt} = 35,000$. This information yields the following maximum altitude values and their probabilities 0.21, 0.60, and 0.19, which are assumed to come from the posterior learned distribution $P(f_{maxalt})$):

- $p(38000) = 0.21$
- $p(37000) = 0.60$
- $p(35000) = 0.19$

Furthermore, suppose that the altitude found in the ACO is 41,000. Then we can calculate the EDoV as:

$$0.21 \cdot 3000 \; + \; 0.60 \cdot 4000 \; + \; 0.19 \cdot 6000 \; = \; 4170.$$

The 4170 is normalized to a number between 0 and 1, and is output by the SC as the EDoV of the constraint associated with the given proposed solution.

For more complicated constraints, e.g., those with conjunctions and/or disjunctions, the EDoV is built up from

the components of the constraint. For example, if multiple CAPs (combat air patrols) satisfy the minimum altitude for a CAP, then we take the average EDoV over all of these CAPs. Finally, note that this approach is used for both object and relational safety constraints.

Assigning Blame to Solution Steps

Recall that a solution proposed by the planners is composed and sent to the SC; it consists of a sequence of steps. For assigning precise blame to the planners, a percentage of blame is assigned to individual steps in a proposed solution. This percentage of blame to steps is then converted into a percentage of blame that goes to each planner – because each component may be responsible for having generated some of the individual plan steps in the solution. In other words, the SC needs to assign a percentage of blame to each step that leads to the violation of a particular constraint for the proposed solution. This seems straightforward, except that we also need to know whether certain *combinations* of steps *jointly* are to blame for a constraint violation. In response to these requirements, we have developed the following data structure as output by the SC:

BLAME FOR CONSTRAINT C7 WITH CANDIDATE SOLUTION S4	
STEP14	3% blame
STEP17	4% blame
STEP38	20% blame
STEPS14 and 17	7% blame
STEPS17 and 38	28% blame
STEPS14 and 38	3% blame
STEPS14 and 17 and 38	35% blame

The observant reader will have noticed that we are testing the power set of the set of all steps in the proposed solution, and that the total blame sums to 100%. There appears to be redundancy in this representation, but there is actually not. From this output, one may learn for example that there is a negative synergy between steps 17 and 38 because jointly they earn more blame than the sum of each alone.

The SC obtains this data structure by actively running experiments, i.e., by ablating steps to see the effect on the EDoV of the constraint. To obtain all of this information, the SC executes the following algorithm, which assumes that the selected constraint has been violated:

1. Select the constraint and the proposed solution on which to focus the experiments (or iterate through them).
2. Examine the proposed solution. Within that solution, find every step that is potentially relevant to causing the constraint to be violated. The set of all of these potentially relevant steps becomes the *Suspect Set*.
3. Run a factorial set of all possible experiments that ablate all possible subsets (i.e., the power set) of the Suspect Set from the Candidate Solution. Note that these subsets include multiple simultaneous culprits, thereby recording the information needed regarding combinations of steps that cause the constraint violation.
4. For each experiment, record the *difference* in EDoV when going from with-ablation to without-ablation. For example, suppose the EDoV for the proposed solution with STEP14 ablated is 0.08, and the EDoV for the proposed solution with STEP14 included is 0.27. Then the difference in EDoV is -0.19. In other words, this is the degradation in performance obtained by including the faulty STEP14.[3]
5. Normalize this difference in EDoV for each experiment by summing all of the differences in EDoV over all experiments, and then figuring out what fraction of reduced EDoV we get for this experiment.

Note that testing the power set can be computationally inefficient. Therefore, we use *relevance* to make the testing efficient. In particular, we identify only the most relevant steps to put into the Suspect Set. The constraint is used for determining relevance, e.g., if the constraint gives a maximum altitude for fighters, then any step mentioning a fighter is considered to be relevant.

Experimental Results

In a recent experiment performed by DARPA BlueForce that compares the quality of deconfliction solutions generated by GILA and novice human airspace managers, GILA was able to perform slightly better than the average human solutions. 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 equivalent background knowledge/inputs, according to careful measurements made by BlueForce. The quality metric included considerations such as the number of solution steps that differed between the expert and the novice/GILA.

We performed further experiments to better understand the impact of the SCLC within GILA. In these experiments, we used three airspace deconfliction scenarios; all are similar to the demonstration trace example in Figure 2.

We ran GILA on several combinations of these scenarios. In each, one scenario and 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 interpret the effect 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:

- *Metric 1:* We compared all airspaces moved by GILA and the expert by grouping them as *true positives*, i.e., those

[3]Note that any step (or steps) whose inclusion *reduces or does not change* the EDoV is removed from the Suspect Set and is also removed from consideration because it is harmless or perhaps even beneficial.

moves performed by both 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 1 determines whether GILA moves the correct airspace, according to the human expert.

- *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 (F1, Altitude), which means that the move changes the altitude of the airspace F1. As a performance score, we measured how many pairs were in agreement between GILA and the expert. Metric 2 determines whether GILA selects the correct parameter values, according to the human expert. This is a more specific metric than the first one.

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 of the underlying problem domain and checking the solutions generated by the system against those constraints. In conclusion, the learning and enforcement of safety constraints helps GILA choose correct deconfliction actions, by penalizing potentially unsafe actions that would be entertained otherwise.

Regarding execution time, the entire planning and deconfliction process took the human novices about 3.5 hours on average, and took GILA about 3 hours on average. Therefore, task automation is promising.

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, Mahadevan, & Steinberg 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, 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 architecture is quite different. These architectures are directed toward integration in a psychologically plausible way, but their mechanism are more centralized. Unlike these other cognitive architectures, GILA does not have a single, homogeneous, unifying computational mechanism; nor does it 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 prior research in learning control rules for search/planning. This 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).

Next, we address research related to the Safety Checker. There is related work 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 & Etzioni 1994; Gordon 2000). Our safety constraints have a similar motivation.

Another related area of research is credit/blame assignment. This topic has a long history that includes classifier systems such as the bucket brigade (Wilson & Goldberg 1989), temporal difference and reinforcement learning (Russell & Norvig 2003), and testing methodologies for system fault diagnosis, e.g., (Pipitone, DeJong, & Spears 1991). Our approach is different from all of these – because the ultimate goal of our blame assignment is to improve the performance of a heterogeneous ensemble of planners. Furthermore, our approach is focused on safety constraints.

The Safety Checker is also related to formal verification, such as model checking (Clarke & Grumberg & Peled 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 & Halpern 2004). In particular, the EDoV is a form of degree of blame. Chockler and Halpern present a theoretical framework for the degree of blame in relation to an agent's epistemic state. Our notion of degree of blame, on the other hand, is more tailored to the combination of procedural solutions (i.e., plans) and safety constraints. In other words, our conceptual framework for "degree of blame" bridges the gap between "safe planning" and the Chockler and Halpern idea of refining the blame (from binary to degrees) assigned during formal verification.

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 and multiple learners. This is in sharp contrast to the traditional learning paradigm that consists of one learner and many training examples. It poses an enor-

mous challenge, i.e., that of maximizing the information gleaned from a minimal amount of data. The heterogeneity of the ensemble of planners helps in this respect because the bias of each planner 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.

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 planning knowledge. An implementation of our approach in a multi-planner 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. We can also test how performance changes as the problem scales.

Acknowledgments

This work is funded by DARPA GILA Contract # FA8650-06-C-7605. The opinions expressed in this paper are those of the authors and do not necessarily reflect the opinions of DARPA. Special thanks go to Dutch Van Sloten and the DARPA BlueForce for helping us to understand and appreciate the job of airspace planning and deconfliction, and to LMCO for excellent project management and integration.

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 *Proc. of the European Space Agency Workshop on On Board Autonomy*. 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.

Chockler, H.; Halpern, J. 2004. Responsibility and blame: A structural-model approach. *JAIR* 22:93–115.

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.

Clarke, E. M.; Grumberg, O.; Peled, D. 1999. *Model Checking*. MIT Press.

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 *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 318–325.

Kuter, U.; Levine, G.; Green, D.; Rebguns, A.; DeJong, G.; and Spears, D. 2007. Learning constraints via demonstration for safe planning. In *Proc. of the AAAI'07 Workshop on Acquiring Planning Knowledge via Demonstration*.

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 *Proc. 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.

Reps, T.; Teitelbaum, T. 1989. *The Synthesizer Generator.* Springer-Verlag.

Russell, S.; Norvig, P. 2003. *Artificial Intelligence: A Modern Approach.* Prentice-Hall.

Pipitone, F.; DeJong, K.; Spears, W. 1991. An artificial intelligence approach to analog systems diagnosis. In *Testing and Diagnosis of Analog Circuits and Systems*, 187–215.

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*, 1042–1047.

Wilson, S. W., and Goldberg, D. E. 1989. A critical review of classifier systems. In *ICGA*.