An Ensemble Learning and Problem Solving Architecture for Airspace Management *

Xiaoqin (Shelley) Zhang[†], Sungwook Yoon[‡], Phillip DiBona[§], Darren Scott Appling[¶], Li Ding[‖], Janardhan Rao Doppa[**], Derek Green[††], Jinhong K. Guo[§], Ugur Kuter[‡‡], Geoff Levine[★], Reid L. MacTavish[✳], Daniel McFarlane[§], James R Michaelis[‖], Hala Mostafa[⊛], Santiago Ontañón[✳], Charles Parker[**], Jainarayan Radhakrishnan[✳], Anton Rebguns[††], Bhavesh Shrestha[†], Zhexuan Song[◇], Ethan B. Trewhitt[¶], Huzaifa Zafar[⊛], Chongjie Zhang[⊛], Daniel Corkill[⊛], Gerald DeJong[★], Thomas G. Dietterich[**], Subbarao Kambhampati[‡], Victor Lesser[⊛], Deborah L. McGuinness[‖], Ashwin Ram[✳], Diana Spears[††] , Prasad Tadepalli[**], Elizabeth T. Whitaker[¶], Weng-Keen Wong[**], James A. Hendler[‖], Martin O. Hofmann[§], Kenneth Whitebread[§]

Abstract

In this paper we describe the application of a novel learning and problem solving architecture to the domain of airspace management, where multiple requests for the use of airspace need to be reconciled and managed automatically. The key feature of our "Generalized Integrated Learning Architecture" (GILA) is a set of integrated learning and reasoning (ILR) systems coordinated by a central meta-reasoning executive (MRE). Each ILR learns independently from the same training example and contributes to problem-solving in concert with other ILRs as directed by the MRE. Formal evaluations show that our system performs as well as or better than humans after learning from the same training data. Further, GILA outperforms any individual ILR run in isolation, thus demonstrating the power of the ensemble architecture for learning and problem solving.

Introduction

Air-space management is a complex knowledge-intensive activity that challenges even the best of human experts. Not only do the experts need to keep track of myriad details of different kinds of aircraft, their limitations and requirements, but they also need to find a safe, mission-sensitive, and cost-effective global schedule of flights for the day. An expert systems approach to airspace management requires painstaking knowledge engineering to build the system and a team of human experts to maintain it when changes occur to the fleet, possible missions, safety protocols, and costs of schedule changes. An approach based on learning from expert demonstrations is more attractive, especially if it can be based on a small number of training examples since this bypasses the knowledge engineering bottleneck (Buchanan and Wilkins 1993).

In this paper we describe a novel learning and problem solving architecture based on multiple *integrated learning and reasoning* (ILR) systems coordinated by a centralized controller called the *meta-reasoning executive* (MRE). Our architecture is directly motivated by the need to learn rapidly from a small number of expert demonstrations in a complex problem solving domain. Learning from a small number of examples is only possible if the learning system has a strong bias or prior knowledge (Mitchell 1997). However, a strong bias could lead to undesired solutions or no solution when the bias is not appropriate for solving the problem at hand. One way to guard against such inappropriate biases and to hedge one's bets is to consider the solutions of learners with different biases and combine them in an optimal fashion. Since learning is highly impacted by the knowledge representation, which in turn influences problem solving, we tightly integrate learning and problem solving in each ILR by allowing it to use its own internal knowledge representation scheme. The MRE is responsible for presenting training examples to the ILRs, posing problems to be solved, and selecting and combining the proposed solutions in a novel way to solve the overall problem. Thus the MRE is spared from understanding the internal representations of the ILRs and operates only in the space of subproblems and candidate solutions. While there have been highly successful uses of ensembles in classification learning including bagging and boosting (Polikar 2006), we are not aware of any work on ensemble learning in the context of complex problem solving. Unlike classification ensembles that use simple voting mechanisms to make the final decision, here the MRE needs to select the best among the proposed sub-problem solutions and assemble them into a final solution.

We developed a system called *Generalized Integrated Learning Architecture* (GILA) that implements the above

*Distribution Statement A (Approved for Public Release, Distribution Unlimited). This material is based upon work supported by DARPA through a contract with Lockheed Martin (prime contract #FA8650-06-C-7605). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, Lockheed Martin or the U.S. Government.

[†]University of Massachusetts at Dartmouth
[‡]Arizona State University
[§]Lockheed Martin Advanced Technology Laboratories
[¶]Georgia Tech Research Institute
[‖]Rensselaer Polytechnic Institute
[**]Oregon State University
[††]University of Wyoming
[‡‡]University of Maryland
[★]University of Illinois at Urbana
[✳]Georgia Institute of Technology
[⊛]University of Massachusetts, Amherst
[◇] Fujitsu Laboratories of America

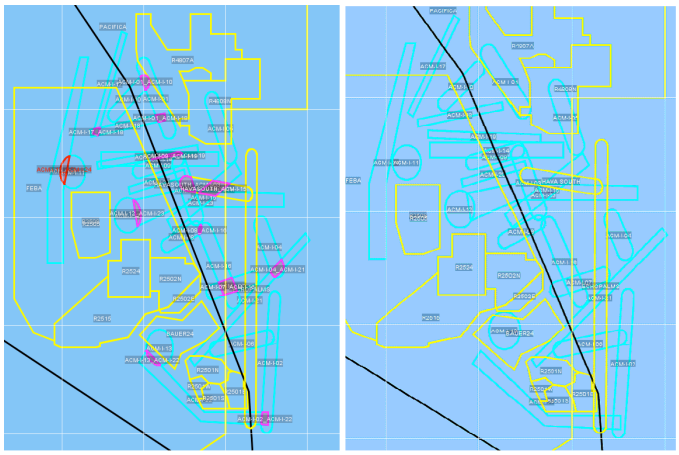

Figure 1: An example of an ACO (left) and its deconflicted solution (right). Yellow: exiting airspaces that cannot be modified. Light Blue: airspace requests that can be modified. Magenta/Red: conflicts.

framework and evaluated it in the domain of airspace management. Our system includes the MRE and 4 different ILRs with diverse biases including the decision-theoretic learner-reasoner (DTLR) that learns a linear cost function, symbolic planner learner and reasoner (SPLR) that learns decision rules and value functions, the case-based learner reasoner (CBLR) that learns and stores a feature-based case database, and the 4D-deconfliction and constraint learner (4DCLR) that learns hard constraints on the schedules in the form of rules. In rigorously evaluated comparisons, GILA was able to outperform human novices, who are provided with the same background knowledge and the same training examples as GILA, and do it in much less time. Our results show that the quality of the solutions of the overall system is better than that of the individual ILRs, all but one of which were unable to fully solve the problems by themselves. This demonstrates that the ensemble learning and problem solving architecture as instantiated by GILA is an effective approach to learning and managing complex problem-solving in domains such as airspace management.

Domain Background

The domain of application for the GILA project is 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 problem is: given a set of Airspace Control Means Requests (ACM-Reqs), each representing an airspace requested by a pilot as part of a given military mission, identify undesirable conflicts between airspace uses and suggest moves in latitude, longitude, time, or altitude that will eliminate them. An Airspace Control Order (ACO) is used to represent the entire collection of airspaces to be used during a given 24-hour period. Each airspace is defined by a polygon described by latitude and longitude points, an altitude range, and a time interval during which the air vehicle will be allowed to occupy the airspace. The process of deconfliction assures that any two vehicles' airspaces do not overlap or conflict. Figure 1 shows an example of an original ACO (left) as well as its deconflicted solution (right). In order to solve the airspace management problem, GILA must decide in what order to address the conflicts during the problem-solving process and, for each conflict, which airspace to move and how to move the airspace so as to resolve the conflict and minimize the impact on the mission. The main problem in airspace deconfliction is that there are typically infinitely many ways to successfully deconflict. The goal of the system is to find deconflicted solutions that are qualitatively *similar to* those found by human experts. Since it is hard to specify what this means, we take the approach of learning from expert demonstrations, and evaluate GILA's results by comparing them to human solutions.

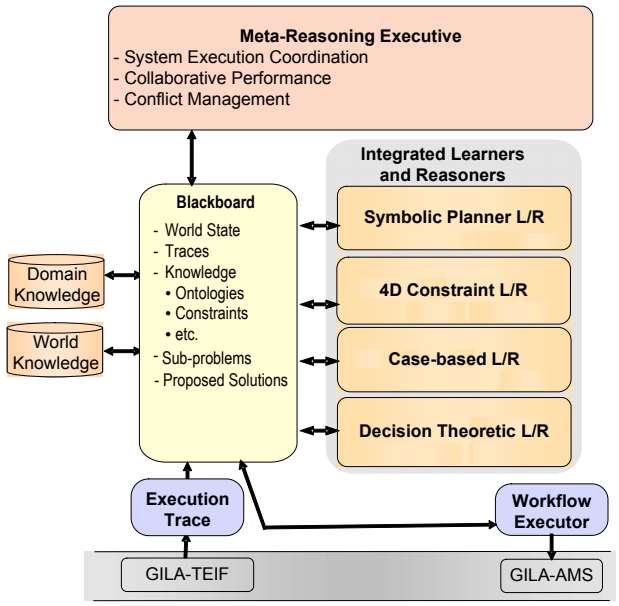

Figure 2: GILA System Architecture (GILA-TEIF: Training and Evaluation Interface; GILA-AMS: Air Management System)

Generalized Integrated Learning Architecture

GILA loosely integrates a number of tightly coupled learning and reasoning mechanisms called Integrated Learner-Reasoners (ILRs). ILR interactions are controlled by a meta-reasoning executive (MRE). Coordination is facilitated by a blackboard. GILA is self-learning, i.e. learns how to best apply its reasoning and learning modules via the MRE. Figure 2 shows GILA's architecture. Different components communicate through a common ontology. We will discuss the ontology, ILRs and MRE in detail in later sections.

As shown in Figure 3, the system process is divided into three sub-phases: demonstration learning, practice learning and performance phases. In the *demonstration learning* phase, a complete, machine-parsable trace of the expert's interactions with a set of application services is captured and is made available to the ILRs via the blackboard. Each ILR uses shared world, domain, and ILR-specific knowledge to expand its private models both in parallel during individual

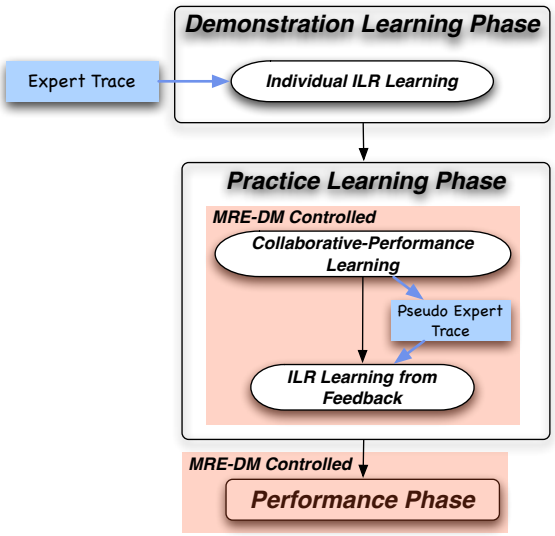

Figure 3: GILA System Process

learning and in collaboration. During the *practice learning* phase, GILA is given a practice problem (a set of airspaces with conflicts) and its goal state (with all conflicts resolved) but not told how this goal state is achieved (the actual modifications to those airspaces). The MRE then directs all ILRs to collaboratively solve this practice problem and generate a solution that is referred as "pseudo expert trace." ILRs can learn from this pseudo expert trace, thus indirectly sharing their learned knowledge through practice. In the *performance* phase, GILA solves a problem based on the knowledge it has already learned. The MRE decomposes the problem as a set of sub-problems (each sub-problem is a conflict to resolve), which are then ordered using the learned knowledge from ILRs. ILRs are requested to solve those sub-problems one at a time according to the ordering. The MRE evaluates the partial solutions to the sub-problems produced by different ILRs, and composes the best ones into a complete final solution. The evaluation criteria is also based on the knowledge learned by ILRs.

Ontology and Explanations

Background ontologies have been used to aid GILA's learning, problem-solving, and explanation processes. We developed a family of ontologies to support the goal of deconflicting requests in an Airspace Control Order (ACO). GILA ontologies are partitioned into two inter-dependent groups: (i) domain knowledge related to the ACO deconfliction problem, e.g., constraints, deconfliction plan steps, problem state description, and (ii) general-purpose knowledge of the world, e.g. explanation, problem-solving and temporal-spatial concepts. By adopting GILA ontologies, especially the explanation interlingua - Proof Markup Language (PML) (McGuinness et al. 2007), MRE and ILRs with drastically different implementations can communicate.

GILA ontologies are also used to drive the collaboration between ILRs and the MRE. Instead of using a peer-to-peer communications model, ILRs are driven by knowledge shared on a centralized blackboard. The ILRs use the blackboard to obtain the current system state and potential problems, guide action choice, and publish results. This communication model can be abstracted as a collaborative and monotonic knowledge base evolution process, run by an open hybrid reasoning system. Besides supporting runtime collaboration, GILA ontologies enable logging and explanation of the learning and problem-solving processes. While the initial implementation only used the blackboard system to encode basic provenance, an updated prototype that leverages the Inference Web (IW) (McGuinness and da Silva 2004) explanation infrastructure and SPARQL uses the blackboard system to log more comprehensive GILA component behavior. Given the complexity of a large hybrid system, explanations are critical to end user's acceptance of the final GILA recommendations and they can also be used by the internal components when trying to decide how and when to make choices.

Integrated Learning Reasoning systems (ILRs)

In this section, we describe how each ILR works inside the GILA system and how their internal knowledge representation formats and learning/reasoning mechanisms are different from each other.

Symbolic Planner Learner and Reasoner

Symbolic planner learner and reasoner (SPLR) represents its learned solution strategy with *hybrid hierarchical representation machine* (HHRM). There is anecdotal evidence to suggest that a hybrid representation is consistent with expert reasoning in the Airspace Deconfliction problem. Experts choose the top level solution strategy by looking at the usage of the airspaces. This type of activity is represented with direct policy representation. When, subsequently, the experts need to decide how much to change the altitude or time, they tend to use some value function, they attempt to minimize. SPLR investigated a method to learn and use hybrid hierarchical representation machine (HHRM) (Yoon and Kambhampati 2007).

An abstract action is either a parameterized action literal such as $A(x_1, \ldots, x_n)$, or as a minimizer of a real-valued cost function such as $A(\operatorname{argmin} V(x_1, \ldots, x_n))$.

HHRM for Airspace Management Domain Since our target domain is airspace management, we provide the set of action hierarchies that represents the action space for the airspace management problem. At the top of the hierarchy, we define three relational actions: 1) altitude-change (airspace), 2) geometrical-change (airspace), 3) width-change (airspace). Each of the top-level actions has a corresponding cost function to minimize. Each cost function is minimized with the following ground level physical actions. *SetAltitude*, *SetPoint* and *SetWidth*.

Learning HHRM Learning of HHRM from the example solution trajectory starts with partitioning the solution according to the top level actions. Note that the example solution presented to SPLR has only physical level actions, e.g., *SetAltiude(Airspace1, 10000ft)*. SPLR then partitions these

0. Learn-HHRM (Training-Sequence, Top-Level-Actions)
1. If Top-Level-Actions is null, then Return
2. Part ← Partition (Training-Sequence, Top-Level-Actions)
3. Learn-Top-Level-Actions (Part, Top-Level-Actions)
4. Learn-HHRM (Part, Lower-Level-Actions (Part))

Figure 4: Hybrid Hierarchical Representation Machine Learning.

sequences into top level action sequences, e.g. **altitude-change** (Airspace1). Then for each partitioned space, SPLR learns the top level actions. This process is recursively applied to each partition by setting the lower level actions as top level actions and then partitioning the sequence of the new top level actions. We conclude this section with the HHRM learning algorithm in Figure 4. We used features based on Taxonomic syntax (Yoon, Fern, and Givan 2002), derived directly from the domain ontology and the demonstration examples, to learn direct policy representation actions. Taxonomic syntax has empirically been proven to work very well with learning direct policies and, as the empirical evaluation demonstrates, this was also the case for SPLR. Linear regression was used to learn the value function actions. Further details of both these phases can be found in (Yoon and Kambhampati 2007).

Decision Theoretic Learner-Reasoner (DTLR)

The *Decision Theoretic Learner-Reasoner* (DTLR) learns a cost function over possible solutions to problems. It is assumed that the expert's solution optimizes the cost function subject to some constraints. The goal of the learner is to learn a close approximation of the expert's cost function. Once the cost function is learned, the performance system tries to find a solution that nearly minimizes the cost through iterative-deepening search.

We will formalize the cost function learning in the framework of structured prediction (Taskar, Guestrin, and Koller 2004; Tsochantaridis et al. 2005). We define a structured prediction problem as a tuple $\{\mathcal{X}, \mathcal{Y}, \Psi, \Delta\}$, where $\mathcal{X}$ is the *input space* and $\mathcal{Y}$ is the output space. In the case of ACO scheduling, an input drawn from $\mathcal{X}$ is a combination of ACO and a set of ACMReq's to be scheduled. An output $\mathbf{y}$ drawn from $\mathcal{Y}$ is a schedule of the deconflicted ACM's. The *loss function*, $\Delta : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re$, quantifies the relative preference of two outputs given some input. Formally, for an input $\mathbf{x}$ and two outputs $\mathbf{y}$ and $\mathbf{y}'$, $\Delta(\mathbf{x}, \mathbf{y}, \mathbf{y}') > 0$ iff $\mathbf{y}$ is a better choice than $\mathbf{y}'$ given input $\mathbf{x}$. Our goal, then, is to learn a function that selects the best output for every input. For each possible input $\mathbf{x}_i \in \mathcal{X}$, we define output $\mathbf{y}_i \in \mathcal{Y}$ to be the best output, so that $\forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \Delta(\mathbf{x}_i, \mathbf{y}_i, \mathbf{y}) > 0$.

We are aided in the learning process by the *joint feature function* $\Psi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^n$. In the ACO scheduling domain, these features are x-y coordinate change, *altitude* change, *time* change for each ACM, and the changes in the number of intersections of the flight path with the enemy territory. The decision function is the linear inner product $\langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle$ where $\mathbf{w}$ is the vector of learned model parameters. The specific goal of learning, then, is to find $\mathbf{w}$ such that $\forall i : \text{argmax}_{\mathbf{y} \in \mathcal{Y}} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle = \mathbf{y}_i$.

We now describe our *Structured Gradient Boosting* (SGB) algorithm (Parker, Fern, and Tadepalli 2006) which is a gradient descent approach to solving the structured prediction problem. Suppose that we have some training example $\mathbf{x}_i \in \mathcal{X}$ with the correct output $\mathbf{y}_i \in \mathcal{Y}$. We define $\widehat{\mathbf{y}}_i$ to be the best incorrect output for $\mathbf{x}_i$ according to the current model parameters.

$$\widehat{\mathbf{y}}_i = \text{argmax}_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \quad (1)$$

$$m_i = \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}) \rangle - \langle \Psi(\mathbf{x}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle \quad (2)$$

$$\text{Cumulative loss } L = \sum_{i=1}^{n} \log(1 + e^{-m_i}). \quad (3)$$

We define the *margin* m_i to be the amount by which the model prefers $\mathbf{y}_i$ to $\widehat{\mathbf{y}}_i$ as output for $\mathbf{x}_i$ (see Equation 2.) The loss on each example is determined by the logitboost function of the margin and is accumulated over all training examples as in Equation 3. The parameters $\mathbf{w}$ of the cost function are repeatedly adjusted in proportion to the negative gradient of the cumulative loss L over the training data.

The problem of finding $\widehat{\mathbf{y}}_i$ which is encountered during both learning and performance is called *inference*. We define a discretized space of operators namely, altitude, time, radius and x-y co-ordinates based on the domain knowledge to produce various possible plans to de-conflict each ACM in the ACMREQ. Since finding the best scoring plan that resolves the conflict is too expensive, we approximate it by trying to find a single change that resolves the conflicts and consider multiple changes only when it is necessary.

Case-Based Learner Reasoner (CBLR)

CBLR learns how to prioritize conflicts and deconflict airspaces by using case-based reasoning (CBR). CBR (Aamodt and Plaza 1994) consists of storing and indexing a set of past experiences (called *cases*) and reusing them to solve new problems, assuming that "similar problems have similar solutions." In GILA, CBLR uses the expert demonstration to learn cases. Unlike other learning techniques that attempt to generalize at learning time, learning in CBR systems consists of representing and retaining cases in the case libraries. All attempts at generalization are made during performance by adapting the solutions of retrieved cases to the problem at hand.

Learning in the CBLR CBLR learns knowledge to enable solving of both conflict prioritization and deconfliction problems and, for that purpose, it uses five knowledge containers. It uses a *prioritization library* which has a set of cases for reasoning about the priority with which conflicts should be solved and a *choice library*, which contains cases for reasoning about which airspace is to be moved in solving a given conflict. It has a *constellation library* used to characterize the neighborhood surrounding a conflict (i.e. its context) and a *deconfliction library* containing the specific steps that the expert used to solve the conflict. Finally, in order to adapt the deconflicted solutions, CBLR uses *adaptation knowledge*, stored as rules and constraints.

CBLR learns cases for the four case libraries by analyzing the expert trace. The order in which the expert solves conflicts provides the information required for prioritization cases, and the expert's choice of which airspace to modify

for a particular conflict is the basis for choice cases. Constraints are learned by observing the range of values that the expert uses for each variable (altitude, radius, width, etc.), and by incorporating any additional constraints that other components in GILA might learn.

Reasoning in the CBLR Given a set of conflicts, CBLR uses the prioritization library to assign to each conflict the priority of the most similar conflict in the library. The conflicts are then ranked according to these priorities, to obtain a final ordering. Thus, the bias of the CBLR is that similar conflicts have similar priorities. Because CBLR followed the approach of the expert so closely, evaluation shows priorities computed by the CBLR were of very high quality.

Given a conflict, CBLR uses a hierarchical problem solving process. First, the choice library is used to determine which of the two airspaces is to move. Then, using the constellation library, CBLR determines which deconfliction strategy (e.g. shift in altitude) is best suited to resolve the conflict given the context. Finally, cases from the deconfliction library are retrieved and adapted using the adaptation knowledge. Adaptation may produce several candidate solutions for each case retrieved and the validity of each candidate solution is evaluated with the help of the 4DCL. CBLR attempts different cases, until satisfactory deconflicted solutions are found. In addition to standard problem solving tests, incremental learning tests show that CBLR uses only the approaches that were demonstrated by the expert, thus showing CBLR's success in learning the expert's approach.

4D Constraint Learner/Reasoner (4DCLR)

The *4D Constraint Learner/Reasoner (4DCLR)* within GILA is responsible for automated learning and application of planning knowledge in the form of safety constraints. Example constraints are: "The altitude of a UAV over the course of its trajectory should never exceed a maximum of 60,000 feet," and "An aircraft trajectory should never be moved so that it intersects a no-fly zone."

The 4DCLR (Rebguns et al. 2008) has two components: (1) the *Constraint Learner (CL)*, which automatically infers constraints from expert demonstration traces, and (2) the *Safety Checker (SC)*, which is responsible for verifying the correctness of solution in terms of their satisfaction or violation of the safety constraints learned by the CL. Why is the SC needed if the ILR's already use the safety constraints during problem solving? The reason is that the ILRs output partial solutions to sub-problems. These partial solutions are then combined by the MRE into one complete final solution, and that is what needs to be checked by the SC.

Constraint Learner We assume that the system designer provides constraint templates a priori; it is the job of the CL to infer the values of parameters within these templates. Learning in the CL is Bayesian. For each parameter, such as the maximum flying altitude for a particular aircraft, the CL begins with a prior probability distribution. If informed, the prior might be a Gaussian approximation of the real distribution obtained by asking the expert for the average, variance, and covariance of the minimum and maximum altitudes. If uninformed, a uniform prior is used.

Learning proceeds based on evidence witnessed by the CL at each step of the demonstration trace. This evidence might be a change in maximum altitude that occurs as the expert positions and repositions an airspace to avoid a conflict. Based on this evidence, the prior is updated using Bayes' Rule and the assumption that the expert always moves an airspace uniformly into a "safe" region. After observing evidence, the CL assigns zero probability to constraint parameters that are inconsistent with the expert's actions, and assigns the highest probability to more constraining sets of parameters that are consistent with the expert's actions.

Safety Checker The SC inputs are candidate sub-problem solutions from the ILRs, the current ACO on which to try the candidate solutions, and the safety constraints output by the CL; it outputs a violation message. The SC uses its 4D Spatio-Temporal Reasoner to verify whether any constraint is violated. A violation message is output by the SC that includes the violated constraint, the solution that violated the constraint, the nature of the violation, and the expected degree (severity) of the violation, normalized to a value in the range [0, 1].

The expected degree of violation is called the *safety violation penalty*. The SC calculates this penalty by finding a normalized expectation of differences between expert actions (from the demonstration trace) and proposed actions in the candidate solution. The differences are weighted by their probabilities from the posterior distribution. The MRE uses the violation penalty to discard sub-problem solutions that are invalid because they have too many violations.

Meta-Reasoning Executive (MRE)

In both the practice learning phase and the performance phase, the system is required to solve a test problem using the learned knowledge. The MRE directs a collaborative performance learning process where ILRs are not directly sharing their learned knowledge but are all contributing to solving the test problem. This collaborative performance process is being modeled as a search process, searching for a path from the initial state to the goal state where the problem is fully solved. The complete solution is a combination of the partial solutions contributed by each ILR found on this path. The given test problem is decomposed as a set of sub-problems (conflicts). As the MRE posts these sub-problems on blackboard, each ILR will post its solutions to some of these sub-problems. These sub-problem solutions are the search operators available at the current state. The MRE then selects one of these solutions and applies it to the current state. As a result, a new state will be generated after applying this sub-problem solution. New conflicts may appear after applying a sub-problem solution. The new state is then evaluated: if it is a goal state, the problem is fully solved; otherwise, the MRE will post all sub-problems (conflicts) existing in the new state and the previous process is repeated. Figure 5 shows an example of a search tree. The yellow nodes represent problem states and blue/green nodes represent sub-problem solutions (sequences of ACM modifications) posted by the ILRs. The yellow and blue/green nodes alternate. When a sub-problem solution is selected to

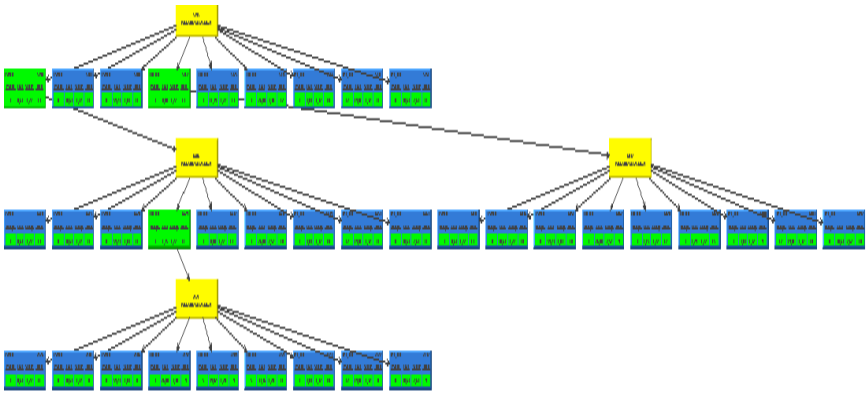

Figure 5: Search Tree Example

be explored, its color is changed from blue to green. If the yellow node represents the problem state p, and its (green) child node represents a solution s, the green node's (yellow) child represents the result of applying s to p. The ordering of nodes to be explored depends on the search strategy. A best-first search strategy is used in this work. This search process is directed by the learned knowledge from the ILRs in the following two ways.

First, GILA is learning to decide which sub-problems to work on initially. It is not efficient to have all ILRs provide solutions to all sub-problems, as it takes more time to generate those sub-problem solutions and also requires more effort to evaluate them. Since solving one sub-problem could make solving the remaining problems easier or more difficult, it is crucial to direct ILRs to work on sub-problems in a facilitating order. This ordering knowledge is learned by each ILR. In the beginning of the search process, the MRE asks ILRs to provide a priority ordering of the sub-problems. Based on each ILR's input, MRE will generate a priority list that suggests which sub-problem to work on first. This suggestion is taken by all ILRs as guidance to generate solutions for sub-problems.

Secondly, GILA learns to evaluate the proposed sub-problem solutions. Each sub-problem solution (green node on the search tree) is evaluated using the learned knowledge from ILRs in the following ways:

1. Check for safety violations in the sub-problem solution. Some sub-problem solutions may cause safety violations that make it invalid. The SafetyChecker learns to check if there is a safety violation and how severe the violation is, which is represented by a number called violation penalty. If the violation penalty is greater than the safety threshold, the MRE discards this sub-problem solution; otherwise, the following evaluations are performed.
2. DTLR derives the execution cost for a sub-problem solution which is expected to approximate the expert's cost function. However, because this is not exact due to various assumptions in its learning such as discretization of the action space and inexact inference.
3. Another ILR, the 4DCL, performs an internal simulations to investigate the results of applying a sub-problem solution to the current state. The resulting new problem state is evaluated and the number of remaining conflicts is returned to the MRE as an estimate of how far it is from the goal state when there are no remaining conflicts.

If a sub-problem solution does not solve any conflict at all it is discarded; otherwise a sub-problem solution is evaluated based on the following factors: the cost of executing all sub-problem solutions selected on this path, safety violation penalties that would be present if the path were executed, and the estimated execution cost & violation penalties of resolving remaining conflicts. These factors are combined using a linear function with a set of weight parameters. The values of these weight parameters can be varied to generate different search behaviors.

Experimental Setup and Results

The GILA system consists of a set of distributed, loosely-coupled components, interacting via a blackboard. Each component is a standalone software module that interacts with the GILA System using a standard set of domain-independent APIs (e.g., interfaces). The distributed nature of the design allows components to operate in parallel, maximizing efficiency and scalability. Figure 6 shows the design of GILA software architecture.

Test Cases and Evaluation Criteria

The testing cases for experiments were developed by Subject Matter Experts (SMEs) from BlueForce LLC. The experiment results were graded by the SME. In each testing case, there are 24 Airspace Control Measures (ACMs). There are 14 conflicts among these ACMs as well as existing airspaces. Figure 1 shows an example of a test case and its solution. Each testing scenario consists of three test cases for demonstration, practice and performance respectively.

The core task is to remove conflicts between ACMs and to configure ACMs such that they do not violate constraints on time, altitude or geometry of airspace. The quality of each step (action) inside the solution is judged according the following factors:

- Whether this action solves a conflict
- Whether this is the requested action
- The simplicity of the action
- Proximity to the problem area
- Suitability for operational context
- Originality of the action

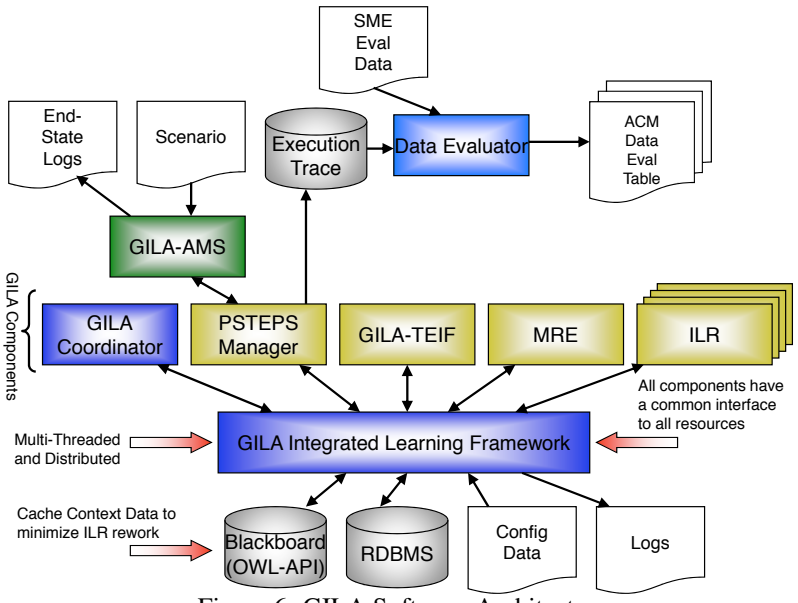

Figure 6: GILA Software Architecture

Each factor is graded on a 0-5 scale. The score for each step is the average of all six factors. The final score for a solution is an average of the score of each step and then multiplied by 20 to make it between 0-100.

GILA vs. Human Novice Performance Comparison

Comparative evaluation of the GILA system is difficult because we have not found a similar man-made system that can learn from a few demonstrations to solve complicated problems. Hence we chose the human novices as our base-line, which is both challenging and exciting.

To compare the performance of GILA with novices, we grouped 33 people into six groups. Each group was given a demonstration case, a practice case and a testing case to perform on. The 33 people were engineers from Lockheed Martin with no priori airspace management experience. We used three testing cases in six combinations for demonstration, practice and performance. We started with an introduction of the background knowledge. Each of the participants was given a written document that lists all the knowledge GILA had before learning. They also received GUI training on how to use the graphical interface designed to make human testing fair in comparison with GILA testing. After the training, each participant was handed a questionnaire to validate that they had gained the basic knowledge to carry out the test. The participants were then given a video demonstration of the expert traces on how to deconflict the airspaces. Based on their observation, they practiced on the practice case, which only had the beginning and the ending states of the airspaces without the detailed actions to deconflict them. Finally, the participants were given a testing case to work on. The test ended with an exit questionnaire.

Table 1 shows the scores achieved by GILA and human novices. The score for human novices shown in the table is the average score of all human novices in a group who are working on the same testing scenario. The score of a solution represents the quality of a solution, which is evaluated by SMEs based on the six factors described earlier. The maximum possible score for one solution is 100. For example, the first row in Table 1 shows that for experiment scenario EFG (using test case E for demonstration, F for practice and G for performance), the average score for human novices is 93.84 while the score of GILA solution is 95.40. It is shown that based on the average of all six experiments, GILA has achieved 105% of human novices' performance. The trimmed mean score of human novices (ignore two highest scores and two lowest scores) is 91.24. The hypothesis that "GILA has achieved 100% human novice performance (measured by trimmed mean score)" is supported with 99.98% confidence using t-test. Here are some general observations of how human novices perform differently from GILA in solving an airspace management problem.

Table 1: Solution Quality: GILA v.s. Human Novices and Solution Number Contributed by ILR

Scenario	Human Novices	GILA	SPLR	CBLR	DTLR
EFG	93.84	95.40	75%	25%	0
EGF	91.97	96.00	60%	30%	10%
FEG	92.03	95.00	64%	36%	0
FGE	91.44	95.80	75%	17%	8%
GEF	87.40	95.40	75%	25%	0
GFE	86.3	95.00	75%	17%	8%
Average	90.5	95.4	70%	24%	6%

1. GILA sometimes gives uneven solutions, for example 35001 ft instead of 35000 ft. Novices can infer from the expert trace that 35000 is the convention. It seems that human reasoning process uses a piece of common knowledge, which is missing in GILA's knowledge base.

2. Overall, novices lacked the ability to manage more than one piece of information. As the complexity of the conflicts increased, they start to forget factors (which ACM, which method to change, etc.) that needed to be taken into account. GILA demonstrated a clearly higher level of information management ability to working with multiple conflicts at the same time.

Table 2: Comparison of GILA and Single-ILR for Conflict Resolving (Test Scenario: EFG)

	GILA	SPLR	DTLR	CBLR
Conflict Solved	14	14	5	7
Quality Score	95.40	81.2	N/A	N/A

The last three columns of Table 1 show the percentage of contribution made by each ILR in the final solution output by GILA. Note that 4DCL is not in this list since it does not propose conflict resolutions but only checks safety and constraint violations. On the average, SPLR clearly dominates the performance by contributing 70% final solution, followed by CBLR which contributes 24% and finally DTLR which contributes 6%. One reason SPLR's performance is so good is that its rule language which is based on taxonomic syntax is very natural and appropriate to capture the kind of rules that people seem to be using. Secondly, its lower level value function captures nuanced differences between different parameter values for the ACM modification operators. Third, it does more exhaustive search during the performance phase than the other ILRs to find the best possible ACM modifications. CBLR does well when its training cases are similar to the test cases, and otherwise does poorly. DTLR suffers from its approximate search and coarse discretization of the search space. It sometimes completely fails to find a solution that passes muster by the constraint learner although such solutions do exist in the finer search space.

GILA vs. Single ILR for Solving Conflicts

To test the importance of the collaboration among various ILRs, we performed additional experiments by running GILA with only one ILR for solving conflicts. However, in all these experiments, DTLR was still used for providing cost information for MRE, and 4DCLR was used for constraint and safety checking. Table 2 shows that DTLR is able to solve five conflicts out of 14 total conflicts while CBLR is able to solve seven of them. Though SPLR is able to solve all 14 conflicts, the quality score of its solution (81.2) is significantly lower than the score achieved by GILA as a whole (95.4). The lower score for SPLR-only solution is caused by some large altitude and time changes, including moving the altitude above 66000. Though there are multiple alternatives to resolving a conflict, usually an action that minimizes change is preferred over those with larger changes. Such large-change actions were not in the solution produced using all ILRs because other ILRs proposed alternative actions, which were preferred and chosen by the MRE. Even when DTLR's cost function is good, it is unable to solve some conflicts because of its incomplete search. CBLR fails to solve conflicts if its case base does not contain similar conflicts. These experiments verify that the collaboration of multiple ILRs is indeed important to solve problems with high-quality solutions.

Conclusions

In this paper we presented an ensemble architecture for learning to solve the airspace management problem. Multiple components, each using a different learning/reasoning mechanism and internal knowledge representations, learn independently from the same expert demonstration trace. The meta-reasoning executive component directs a collaborative performance process, in which it posts sub-problems and selects partial solutions from ILRs to explore. During this process, each ILR contributes to the problem-solving without explicitly transferring its learned knowledge. Experimental results show that GILA matches or exceeds the performance of human novices after learning from the same expert demonstration. This system can be used to provide advice for human operators to deconflict airspace requests. It is also straightforward to use this architecture for a new problem domain since the MRE and the core learning algorithms inside ILRs are domain-independent, and also there is no need to maintain a common knowledge representation. This ensemble learning and problem-solving architecture opens a new path to learning to solve complex problems from very few examples.

References

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

Buchanan, B. G., and Wilkins, D. C. 1993. *Readings in Knowledge Acquisition and Learning :Automating the Construction and Improvement of Expert Systems*. San Mateo, CA: Morgan Kaufmann.

McGuinness, D. L., and da Silva, P. P. 2004. Explaining answers from the semantic web: the inference web approach. *J. Web Sem.* 1(4):397–413.

McGuinness, D. L.; Ding, L.; da Silva, P. P.; and Chang, C. 2007. Pml 2: A modular explanation interlingua. In *Proceedings of AAAI'07 Workshop on Explanation-Aware Computing*.

Mitchell, T. M. 1997. *Machine Learning*. New York, NY: Mc Graw Hill.

Parker, C.; Fern, A.; and Tadepalli, P. 2006. Gradient boosting for sequence alignment. In *Proceedings of the 21st National Conferenceo on Artificial Intelligence*. AAAI Press.

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

Rebguns, A.; Green, D.; Levine, G.; Kuter, U.; and Spears, D. 2008. Inferring and applying safety constraints to guide an ensemble of planners for airspace deconfliction. In *CP/ICAPS COPLAS'08 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

Taskar, B.; Guestrin, C.; and Koller, D. 2004. Max margin markov networks. In *NIPS*.

Tsochantaridis, I.; Joachims, T.; Hofmann, T.; and Altun, Y. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research* 6:1453–1484.

Yoon, S., and Kambhampati, S. 2007. Hierarchical strategy learning with hybrid representations. In *AAAI 2007 Workshop on Acquiring Planning Knowledge via Demonstrations*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *In Proceedings of Eighteenth Conference in Uncertainty in Artificial Intelligence*.