An Ensemble Architecture for Learning Complex Problem-Solving Techniques From Demonstration

Xiaoqin Shelley Zhang, University of Massachusetts at Dartmouth[1]
Sungwook Yoon, Arizona State University[2]
Phillip DiBona, Lockheed Martin Advanced Technology Laboratories[3]
Darren Scott Appling, Georgia Tech Research Institute[4]
Li Ding, Rensselaer Polytechnic Institute[5]
Janardhan Rao Doppa, Oregon State University[6]
Derek Green, University of Wyoming[7]
Jinhong K. Guo[3], Ugur Kuter, University of Maryland[8]
Geoff Levine, University of Illinois at Urbana[9]
Reid L. MacTavish, Georgia Institute of Technology[10]
Daniel McFarlane[3],James R Michaelis[5], Hala Mostafa, University of Massachusetts, Amherst[11]
Santiago Ontañón[10], Charles Parker[6], Jainarayan Radhakrishnan[10],Anton Rebguns[7], Bhavesh Shrestha[1], Zhexuan Song, Fujitsu Laboratories of America[12]
Ethan B. Trewhitt[4], Huzaifa Zafar[11], Chongjie Zhang[11], Daniel Corkill[11], Gerald DeJong[9], Thomas G. Dietterich[6], Subbarao Kambhampati[2], Victor Lesser[11],Deborah L. McGuinness[5], Ashwin Ram[10],Diana Spears[7], Prasad Tadepalli[6], Elizabeth T. Whitaker[4], Weng-Keen Wong[6], James A. Hendler[5], Martin O. Hofmann[3], Kenneth Whitebread[3]

We present a novel ensemble architecture for learning problem-solving techniques from a very small number of expert solutions and demonstrate its effectiveness in a complex real-world domain. The key feature of our "Generalized Integrated Learning Architecture" (GILA) is a set of heterogeneous independent learning and reasoning (ILR) components, coordinated by a central meta-reasoning executive (MRE). The ILRs are *weakly coupled* in the sense that all coordination during learning and performance happens through the MRE. Each ILR learns independently from a small number of expert demonstrations of a complex task. During performance, each ILR proposes partial solutions to subproblems posed by the MRE, which are then selected from and pieced together by the MRE to produce a complete solution. The heterogeneity of the learner-reasoners allows both learning and problem solving to be more effective because their abilities and biases are complementary and synergistic. We describe the application of this novel learning and problem solving architecture to the domain of airspace management, where multiple requests for the use of airspaces need to be deconflicted, reconciled and managed automatically. Formal evaluations show that our system performs as well as or better than humans after learning from the same training data. Furthermore, GILA outperforms any individual ILR run in isolation, thus demonstrating the power of the ensemble architecture for learning and problem solving.

General Terms: Design, Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Ensemble Architecture, Learning from Demonstration, Complex Problem-Solving

ACM Reference Format:
ACM Trans. Intell. Syst. Technol. 0, 0, Article 00 (20XX), 35 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Distribution Statement (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.

1. INTRODUCTION

We present GILA (Generalized Integrated Learning Architecture), a learning and problem-solving architecture that consists of an ensemble of subsystems that learn to solve problems from a very small number of expert solutions. Because human experts who can provide training solutions for complex tasks such as airspace management are rare and their time is expensive, our learning algorithms are required to be highly sample-efficient. Ensemble architectures such as bagging, boosting, and co-training have proved to be highly sample-efficient in classification learning [Breiman 1996; Freund and Schapire 1996; Blum and Mitchell 1998; Dietterich 2000b]. Ensemble architectures have a long history in problem solving as well, starting with the classic Hearsay-II system to the more recent explosion of research in multi-agent systems [Erman et al. 1980; Weiss 2000]. In this article, we explore an ensemble learning approach for use in problem solving. Both learning and problem solving are exceptionally complicated in domains such as airspace management, due to the complexity of the task, the presence of multiple interacting subproblems, and the need for near-optimal solutions. Unlike in bagging and boosting, where a single learning algorithm is typically employed, our learning and problem-solving architecture has multiple heterogeneous learner-reasoners that learn from the same training data and use their learned knowledge to collectively solve problems. The heterogeneity of the learner-reasoners allows both learning and problem solving to be more effective because their abilities and biases are complementary and synergistic. The heterogeneous GILA architecture was designed to enable each learning component to learn and perform without limitation from a common system-wide representation for learned knowledge and component interactions. Each learning component is allowed to make full use of its idiosyncratic representations and mechanisms. This feature is especially attractive in complex domains where the system designer is often not sure which components are the most appropriate, and different parts of the problem often yield to different representations and solution techniques. However, for ensemble problem solving to be truly effective, the architecture must include a centralized coordination mechanism that can divide the learning and problem-solving tasks into multiple subtasks that can be solved independently, distribute them appropriately, and during performance, judiciously combine the results to produce a consistent complete solution.

In this article, we present a learning and problem-solving architecture that consists of an ensemble of independent learning and reasoning components (ILRs) coordinated by a central subsystem known as the "meta-reasoning executive" (MRE). Each ILR has its own specialized representation of problem-solving knowledge, a learning component, and a reasoning component which are tightly integrated for optimal performance. We considered the following three possible approaches to coordinate the ILRs through the MRE during both learning and performance.

(1) *Independent learning and selected performance.* Each ILR independently learns from the same training data and performs on the test data. The MRE selects one out of all the competing solutions for each test problem.
(2) *Independent learning and collaborative performance.* The learning is independent as before. However, in the performance phase, the ILRs share individual subproblem solutions and the MRE selects, combines, and modifies shared subproblem solutions to create a complete solution.
(3) *Collaborative learning and performance.* Both learning and performance are collaborative, with multiple ILRs sharing their learned knowledge and their solutions to the test problems.

Roughly speaking, in the first approach, there is minimal collaboration only in the sense of a centralized control that distributes the training examples to all ILRs and selects the final solution among the different proposed solutions. In the second approach, learning is still separate, while there is stronger collaboration during the problem solving in the sense that ILRs solve individual subproblems, whose solutions are selected and composed by the MRE. In the third approach, there is collaboration during both learning and problem solving; hence a shared language would be required for communicating aspects of learned knowledge and performance solution if each ILR uses a different internal knowledge representation. An example of this approach is the POIROT system [Burstein et al. 2008], where all components use one common representation language and the performance is based on one single learned hypothesis.

The approach we describe in this article, namely, *independent learning with limited sharing and collaborative performance* is closest to the second approach. It is simpler than the third approach where learning is collaborative, and still allows the benefits of collaboration during performance by being able to exploit individual strengths of different ILRs. Since there is no requirement to share the learned knowledge, each ILR adopts an internal knowledge representation and learning method that is most suitable to its own performance. Limited sharing of learned knowledge does happen in this version of the GILA architecture, though it is not required[1].

The ILRs use shared and ILR-specific knowledge in parallel to expand their private internal knowledge databases. The MRE coordinates and controls the learning and the performance process. It directs a collaborative search process, where each search node represents a problem-solving state and the operators are subproblem solutions proposed by ILRs. Furthermore, the MRE uses the learned knowledge provided by ILRs to decide the following: (1) which subproblem to work on next, (2) which subproblem solution (search node) to select for exploration (expansion) next, (3) when to choose an alternative for a previous subproblem that has not been explored yet, and (4) when to stop the search process and present the final solution. In particular, GILA offers the following features:

— Each ILR learns from the same training data independently of the other ILRs, and produces a suitable hypothesis (solution) in its own language.
— A blackboard architecture [Erman et al. 1980] is used to enable communication among the ILRs and the MRE and to represent the state of learning/performance managed by the MRE.
— During the performance phase, the MRE directs the problem-solving process by subdividing the overall problem into subproblems and posting them on a centralized blackboard structure.
— Using prioritization knowledge learned by one of the ILRs, the MRE directs the ILRs to work on one subproblem at a time. Subproblems are solved independently by each ILR, and the solutions are posted on the blackboard.
— The MRE conducts a search process, using the subproblem solutions as operators, in order to find a path leading to a conflict-free goal state. The path combines appropriate subproblem solutions to create a solution to the overall problem.

There are several advantages of this architecture.

— *Sample Efficiency.* This architecture facilitates rapid learning, since each example may be used by different learners to learn from different small hypothesis spaces. This is especially important when the training data is sparse and/or expensive.
— *Semi-supervised learning.* The learned hypotheses of our ILRs are diverse even though they are learned from the same set of training examples. Their diversity is due to multiple independent learning algorithms. Therefore, we can leverage unlabeled examples in a co-training framework [Blum and Mitchell 1998]. Multiple learned hypotheses improve the solution quality, if the MRE is able to select the best from the proposed subproblem solutions and compose them.
— *Modularity and Extensibility.* Each ILR has its own learning and reasoning algorithm; it can use specialized internal representations that it can efficiently manipulate. The modularity of GILA makes it easier to integrate new ILRs into the system in a plug-and-play manner, since they are not required to use the same internal representations.

This work has been briefly presented in [Zhang et al. 2009]. However, this article provides significantly more details about the components, the GILA architecture, as well as discussions of lessons

[1]There are two ILRs sharing their learned constraint knowledge. This is easy because they use the same internal representation format for constraint knowledge; therefore, no extra communication translation effort is needed (see Section 4.3.1 for more details).

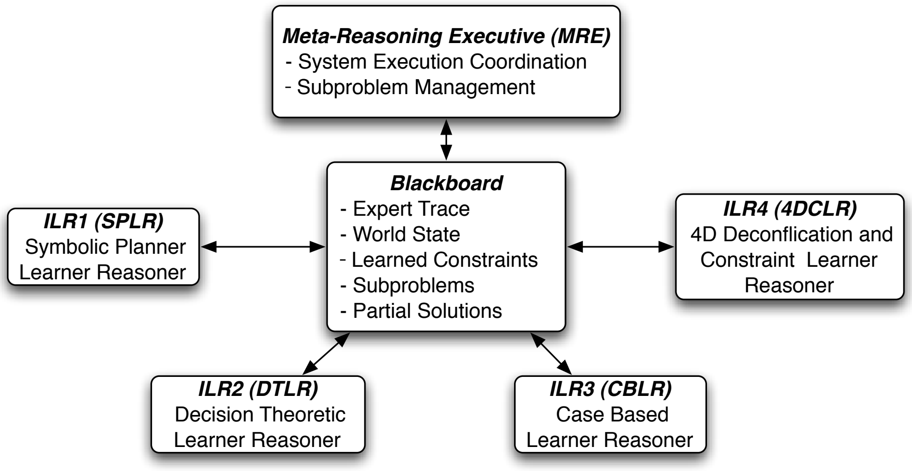

Meta-Reasoning Executive (MRE)
- System Execution Coordination
- Subproblem Management
Blackboard
- Expert Trace
- World State
- Learned Constraints
- Subproblems
- Partial Solutions
ILR1 (SPLR)
Symbolic Planner Learner Reasoner
ILR4 (4DCLR)
4D Deconflication and Constraint Learner Reasoner
ILR2 (DTLR)
Decision Theoretic Learner Reasoner
ILR3 (CBLR)
Case Based Learner Reasoner

Fig. 1: Ensemble learning and problem solving architecture

learned and additional experimental results about the effect of demonstration content and the effect of practice. In Section 2, we present the ensemble learning architecture for complex problem solving, which is implemented by GILA. We then introduce the airspace management domain (Section 3), in which GILA has been extensively evaluated. Components in GILA include the MRE (Section 5) and four different ILRs: the symbolic planner learner-reasoner (SPLR) (Section 4.1), the decision-theoretic learner-reasoner (DTLR) (Section 4.2), the case-based learner-reasoner (CBLR) (Section 4.3), and the 4D-deconfliction and constraint learner-reasoner (4DCLR) (Section 4.4). In rigorously evaluated comparisons (Section 6), GILA was able to outperform human novices who were provided with the same background knowledge and the same training examples as GILA, and GILA used much less time than human novices. Our results show that the quality of the solutions of the overall system is better than that of any individual ILR. Related work is presented in Section 7. Our work 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. In Section 8, we summarize the lessons learned from this work and discuss how GILA can be transferred to other problem domains.

2. ENSEMBLE LEARNING AND PROBLEM-SOLVING ARCHITECTURE

In this section, we will give an overview of the GILA architecture, presenting the reasons behind our choice of this architecture and explaining its usefulness in a variety of different settings.

2.1. Problem Statement

Given a small set of training demonstrations, pairs of problems and corresponding solutions $\{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^{m}$ of task $\mathcal{T}$, to solve a complex problem, we want to learn the general problem-solving skills for the task $\mathcal{T}$.

2.2. GILA's Ensemble Architecture

Most of the traditional ensemble learning algorithms for classification, such as bagging or boosting, use a single hypothesis space and a single learning method. We use multiple hypothesis spaces and multiple learning methods in our architecture corresponding to each Independent Learner-Reasoner (ILR), and a Meta Reasoning Executive (MRE) that combines the decisions from the ILRs. Figure 1 shows GILA's ensemble architecture.

Table I: The Four Independent Learner-Reasoners (ILRs)

Name	Hypothesis Representation	Performance Functions
SPLR	decision rules (what to do) value functions (how to do)	propose subproblem solutions
DTLR	cost function	propose subproblem solutions provide evaluations of subproblem solutions
CBLR	feature-based cases	propose subproblem solutions rank subproblems
4DCLR	safety constraints	generate resulting states of applying subproblem solutions check safety violations

Meta Reasoning Executive (MRE): The MRE is the decision maker in GILA. It makes decisions such as which subproblem sp_i to focus on next (search-space ordering) and which subproblem solution to explore among all the candidates provided by ILRs (evaluation).

Independent Learner-Reasoner (ILR): We developed four ILRs for GILA, as shown in Figure 1. Each ILR learns how to solve subproblems sp_i from the given set of training demonstrations $\{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^{m}$ for task $\mathcal{T}$. Each ILR uses a different hypothesis representation and a unique learning method, as shown in Table I.

The first ILR is a *symbolic planner learner-reasoner (SPLR)* [Yoon and Kambhampati 2007], which learns a set of decision rules that represent the expert's reactive strategy (what to do). It also learns detailed tactics (how to do it) represented as value functions. This hierarchical learning closely resembles the reasoning process that a human expert uses when solving the problem. The second ILR is a *decision-theoretic learner-reasoner (DTLR)* [Parker et al. 2006], which learns a cost function that approximates the expert's choices among alternative solutions. This cost function is useful for GILA decision-making, assuming that the expert's solution optimizes the cost function subject to certain constraints. The DTLR is especially suitable for the types of problems that GILA is designed to solve. These problems generate large search spaces because each possible action has numerical parameters whose values must be considered. This is also the reason why a higher-level search is conducted by the MRE, and a much smaller search space is needed in order to find a good solution efficiently. The DTLR is also used by the MRE to evaluate the subproblem solution candidates provided by each ILR. The third ILR is a *case-based learner-reasoner (CBLR)* [Muñoz-Avila and Cox 2007]. It learns and stores a feature-based case database. The CBLR is good at learning aspects of the expert's problem solving that are not necessarily explicitly represented, storing the solutions and cases, and applying this knowledge to solve similar problems. The last ILR is a *4D-deconfliction and constraint learner-reasoner (4DCLR)*, which consists of a *Constraint Learner (CL)* and a *Safety Checker (SC)*. The 4DCLR learns and applies planning knowledge in the form of safety constraints. Such constraints are crucial in the airspace management domain. The 4DCLR is also used for internal simulation to generate an expected world state; in particular, to find the remaining conflicts after applying a subproblem solution. The four ILR components and the MRE interact through a blackboard using a common ontology [Michaelis et al. 2009]. The blackboard holds a representation of the current world state, the expert's execution trace, some shared learned knowledge such as constraints, subproblems that need to be solved, and proposed partial solutions from ILRs.

We view solving each problem instance of the given task $\mathcal{T}$ as a state-space search problem. The start state S consists of a set of subproblems $sp_1, sp_2, \ldots sp_k$. For example, in the airspace management problem, each subproblem sp_i is a conflict involving airspaces. At each step, the MRE chooses a subproblem sp_i and then gives that chosen subproblem to each ILR for solving. ILRs publish their solutions for the given subproblem on the blackboard, and the MRE then picks the best solution using the learned knowledge for evaluation. This process repeats until a goal state is found or a preset time limit is reached. Since the evaluation criteria are also being learned by ILRs,

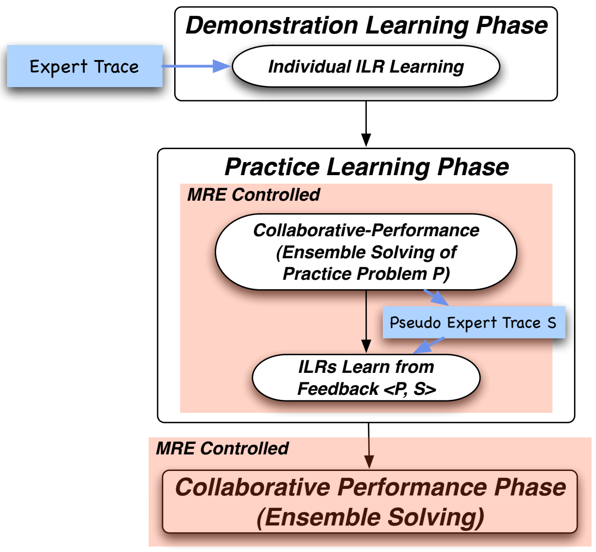

Demonstration Learning Phase
Expert Trace
Individual ILR Learning
Practice Learning Phase
MRE Controlled
Collaborative-Performance (Ensemble Solving of Practice Problem P)
Pseudo Expert Trace S
ILRs Learn from Feedback <P, S>
MRE Controlled
Collaborative Performance Phase (Ensemble Solving)

Fig. 2: GILA's system process

learning to produce satisfactory solutions of high quality depends on how well the whole system has learned.

Connections to Search-Based Structured Prediction: Our approach can be viewed as a general version of *Search-Based Structured Prediction*. The general framework of search-based structured prediction [Daumé III and Marcu 2005; Daumé III et al. 2009] views the problem of labeling a given structured input x by a structured output y as searching through an exponential set of candidate outputs. LaSo (Learning as Search optimization) was the first work in this paradigm. *LaSo* tries to rapidly learn a heuristic function that guides the search to reach the desired output y based on all the training examples. Xu et al. extended this framework to learn beam search heuristics for planning problems [Xu et al. 2007]. In the case of greedy search [Daumé III et al. 2009], the problem of predicting the correct output y for a given input x can be seen as making a sequence of smaller predictions $y_1, y_2, \ldots, y_T$ with each prediction y_i depending on the previous predictions. It reduces the structured prediction problem to learning a multi-class classifier h that predicts the correct output y_t at time t based on the input x and partial output $y_1, y_2, \ldots, y_{t-1}$. In our case, each of these smaller predictions y_i corresponds to solutions of the subproblems sp_i, which can be more complex (structured outputs) than a simple classification decision.

2.3. System Process

GILA's system process is divided into three phases: *demonstration learning*, *practice learning* and *collaborative performance*, as shown in Figure 2. During the demonstration learning phase, a complete, machine-parsable trace of the expert's interactions with a set of application services is captured and 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 demonstration learning and in collaboration during the practice learning. During the practice learning phase, GILA is given a practice problem (i.e., a set of airspaces with conflicts) and a goal state (with no remaining conflicts) but it is not told how this goal state was achieved (via actual modifications to those airspaces). The MRE then directs all ILRs to collaboratively attempt to solve this practice problem and generate a solution that is referred to as a "pseudo expert trace." ILRs can learn from this pseudo expert trace (assuming it is successful), thus indirectly sharing their learned knowledge through practice. In the collaborative performance phase, GILA solves an evaluation problem based on the knowledge it has already learned. A sequential learning feature has been implemented in GILA, so that each ILR can build upon its previous learned knowledge by loading a file that contains its learned knowledge when the system starts.

2.3.1. Demonstration Learning - Individual ILR Learning. The system is provided with a set of training examples (demonstrations) $\{\langle\mathcal{P}_i, \mathcal{S}_i\rangle\}_{i=1}^{m}$ of task $\mathcal{T}$ and the corresponding training examples $\{\langle\mathcal{P}_i, \mathcal{R}_i\rangle\}_{i=1}^{m}$ of ranking the subproblems when performing task $\mathcal{T}$. Learning inside the ensemble architecture happens as follows. First, the system learns a ranking function $\mathcal{R}$ using a rank-learning algorithm. This function $\mathcal{R}$ provides an order in which subproblems should be solved. Then each ILR i learns a hypothesis h_{ILR_i} from the given training examples; this process is called *Individual ILR Learning*. We will describe the learning methodology of each ILR in Section 4. Recall that ILRs are diverse because they use different hypothesis representations and different learning methods, as shown in Table I.

ALGORITHM 1: ENSEMBLE SOLVING PROCESS

```
Input: problem instance $\mathcal{P}$ of task $\mathcal{T}$;
       learned hypothesis of each ILR: $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$;
       ranking function $\mathcal{R}$ to rank subproblems;
Output: solution of the given problem $sol$.
 1: Initialize the start state $s = sp_1, sp_2, \ldots sp_k$
 2: root node $n$ = new Node($s$)
 3: Add node $n$ to the open list
 4: Create evaluation function $\mathcal{E}$ using $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$.
 5: repeat
 6:    node $n'$ = best node popped from the open list based on evaluation $\mathcal{E}(s')$, $s'$ = state($n'$)
 7:    if $s'$ is goal state then
 8:       $sol$ = sequence of subproblem solutions applied from start state $s$ to goal state $s'$
 9:       return solution of the given problem: $sol$
10:    else
11:       $sp_{focus}$ = highest ranked subproblem in current state $s'$ based on ranking $\mathcal{R}(s')$
12:       for each ILR $i = 1$ to $n$ do
13:          Solve subproblem: $sol_{ILR_i} = SolveH(h_{ILR_i}, s', sp_{focus})$
14:          new resulting state $s_i$ = applying $sol_{ILR_i}$ to current state $s'$
15:          add new Node($s_i$) to the open list
16:       end for
17:    end if
18: until open list is empty or a preset time limit is reached
19: return no solution found
```

2.3.2. Ensemble Solving - Collaborative Performance. Algorithm 1 describes how a new problem instance $\mathcal{P}$ for task $\mathcal{T}$ is solved with collaborative performance. The start state s is initialized as the set of subproblems $sp_1, sp_2, \ldots sp_k$. The highest ranked subproblem sp_{focus} is chosen based on the learned ranking function $\mathcal{R}$. The MRE informs all ILRs of the current focused subproblem sp_{focus} and each ILR i publishs its solution(s) sol_{ILR_i}, which may be a solution to a different subproblem if one ILR cannot find a solution to the current focused subproblem sp_{focus}. New states, resulting from applying each of these subproblem solutions to the current state, are generated by the 4DCLR through internal simulation. These new states are evaluated based on an evaluation function $\mathcal{E}$, which is created using the knowledge learned by ILRs. The MRE then selects the best state to explore n', according to $\mathcal{E}$. This process is repeated until reaching a goal state, i.e., a state where all subproblems are solved, or a preset time limit is reached. If a goal state is found, then a solution is returned, which is the sequence of subproblem solutions applied from the start state to the goal state; otherwise, the system reports *no solution found*.

This ensemble solving process is a best-first search, using the subproblem solutions provided by ILRs as search operators. This process can be viewed as a hierarchical search since each ILR is searching for subproblem solutions in a lower-lever internal search space with more details. The top-level search space is therefore much smaller because each ILR is only allowed to propose a limited number of subproblem solutions. The performance of this search process is highly dependent on how well each ILR has learned. A solution can only be found if, for each subproblem, at least one ILR has learned how to solve it. A better solution can be found when some ILRs have learned to solve a subproblem in a better way and also some ILRs have learned to evaluate problem states more accurately. A solution can be found quicker (with less search effort) if the learned ranking function can provide a more beneficial ordering of subproblems. The search can also be more efficient when a better evaluation function has been learned, which can provide an estimated cost closer to the real path cost. As a search process, the ensemble solving procedure provides a practical approach for all ILRs to collaboratively solve a problem without directly communicating their learned knowledge, which is in heterogeneous representations, as shown in Table I. Each ILR has unique advantages, and the ensemble works together under the direction of the MRE to achieve the system's goals, which cannot be achieved by any single ILR. The conjecture that no single ILR can perform as well as the multi-ILR system is supported by experimental results presented in Section 6.3.1.

ALGORITHM 2: PRACTICE LEARNING

```
Input: $\mathcal{L}_p = \{\langle \mathcal{P}_i, \mathcal{S}_i \rangle\}_{i=1}^{m}$: the set of training examples for solving problems of task $\mathcal{T}$ (demonstrations);
       $\mathcal{L}_r = \{\langle \mathcal{P}_i, \mathcal{R}_i \rangle\}_{i=1}^{m}$: the set of training examples for learning to rank subproblems;
       $\mathcal{U}$ = set of practice problem instances for task $\mathcal{T}$.
Output: the learned hypothesis of each ILR: $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$ and ranking function $\mathcal{R}$.

 1: Learn hypotheses $h_{ILR_1}, h_{ILR_2}, \ldots, h_{ILR_n}$ from solved training examples $\mathcal{L}_p$
 2: Learn Ranking function $\mathcal{R}$ from $\mathcal{L}_r$
 3: $\mathcal{L}_{new} = \mathcal{L}_p$
 4: repeat
 5:    for each problem $\mathcal{P} \in \mathcal{U}$ do
 6:       $\mathcal{S}$ = Ensemble-Solve($\mathcal{P}$, $h_{ILR_1}, h_{ILR_2}, \ldots, h_{ILR_n}$, $\mathcal{R}$)
 7:       $\mathcal{L}_{new} = \mathcal{L}_{new} \bigcup \langle \mathcal{P}, \mathcal{S} \rangle$
 8:    end for
 9:    Re-learn $h_{ILR_1}, h_{ILR_2}, \ldots, h_{ILR_n}$ from new examples $\mathcal{L}_{new}$
10: until convergence or maximum co-training iterations
11: return the learned hypothesis of each ILR $h_{ILR_1}, h_{ILR_2}, \ldots h_{ILR_n}$ and ranking function $\mathcal{R}$
```

2.3.3. Practice Learning. In practice learning, we want to learn from a small set of training examples, $\langle \mathcal{L}_p, \mathcal{L}_r \rangle$ for solving problems and for learning to rank subproblems respectively, and a set of unsolved problems $\mathcal{U}$. Our ideas are inspired by the iterative co-training algorithm [Blum and Mitchell 1998]. The key idea in co-training is to take two diverse learners and make them learn from each other using the unlabeled data. In particular, co-training trains two learners h_1 and h_2 separately on two views ϕ_1 and ϕ_2, which are conditionally independent of the other given the class label. Each learner will label some unlabeled data to augment the training set of the other learner, and then both learners are re-trained on this new training set. This process is repeated for several rounds. The difference or diversity between the two learners helps when teaching each other. As the co-training process proceeds, the two learners will become more and more similar, and the difference between the two learners becomes smaller. More recently, a result that shows why co-training without redundant views can work is proved in [Wang and Zhou 2007]. Wang and Zhou show that as long as learners are diverse, co-training will improve the performance of the learners.

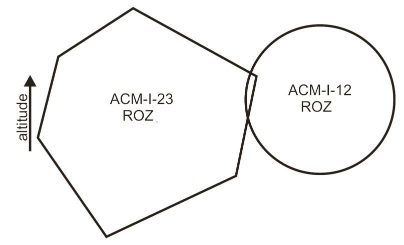
Any set of learning algorithms for problem solving could be used as long as they produce diverse models, which is an important requirement for practice learning to succeed [Blum and Mitchell 1998; Wang and Zhou 2007]. In our case, there are four different learners (ILRs) learning in a supervised framework with training demonstrations ($\mathcal{L}_p$, $\mathcal{L}_r$). The goal of supervised learning is to produce a model which can perfectly solve all the training instances under some reasonable time constraints. For example, our Decision Theoretic Learner and Reasoner (DTLR) attempts to learn the cost function of the expert in such a way that it ranks all good solutions higher than bad solutions by preserving the preferences of the expert. Each practice problem $\mathcal{P} \in \mathcal{U}$ is solved through collaborative performance – ensemble solving (Algorithm 2). The problem $\mathcal{P}$ along with its solution $\mathcal{S}$ is then added to the training set. The system re-learns from the new training set and this process repeats until convergence is achieved or the maximum number of co-training iterations has been reached.

3. DOMAIN BACKGROUND

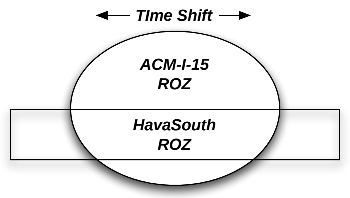
The domain of application used for developing and evaluating GILA is airspace management in an Air Operations Center (AOC). Airspace management is the process of making changes to requested airspaces so that they do not overlap with other requested airspaces or previously approved airspaces. The problem that GILA tackles is the following. Given a set of Airspace Control Measures Requests (ACMReqs), each representing an airspace requested by a pilot as part of a given military mission, identify undesirable conflicts between airspace uses and suggest changes 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. In order to resolve a conflict that involves two ACMs, the expert, who is also called the *subject matter expert (SME)*, first chooses one ACM and then decides whether to change its altitude (Figure 3(a)), change its time (Figure 3(b)), or move its position (Figure 3(c)). The complete modification process is an expert solution trace that GILA uses for training.

This airspace management problem challenges even the best human experts because it is complex and knowledge-intensive. Not only do experts need to keep track of myriad details of different kinds of aircraft and 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 system approach to airspace management requires painstaking knowledge engineering to build the system, as well as a team of human experts to maintain it when changes occur to the fleet, possible missions, safety protocols and costs of schedule changes. For example, flights need to be rerouted when there are forest fires occurring on their original paths. Such events require knowledge re-engineering. In contrast, our approach based on learning from an expert's demonstration is more attractive, especially if it only needs a very small number of training examples, which are more easily provided by the expert.

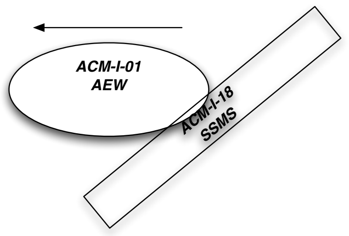
To solve the airspace management problem, GILA must decide in what order to address the conflicts during the problem-solving process and, for each conflict, which airspace to move and how to move the airspace to resolve the conflict and minimize the impact on the mission. Though there are typically infinitely many ways to resolve a particular conflict, some changes are better than others according to the expert's internal domain knowledge. However, such knowledge is not revealed directly to GILA in the expert's solution trace. The solution trace is the only input from which GILA may learn. In other words, learning is from examples of the expert performing the problem-solving task, rather than by being given the expert's knowledge. The goal of the system is to find good deconflicted solutions, which are qualitatively *similar to* those found by human experts. GILA's solutions are evaluated by experts by being compared to the solutions of human novices who also learn from the same demonstration trace.



(a) The expert removes the conflict by changing the altitude of ACM-I-23. The expert changes the altitude for "ROZ" usage when its shape is "polygon" and it conflicts with another "ROZ" ACM, whose shape is "circle".



(b) The expert shifts the time when "orbit" type "ROZ" ACM is in conflict with another "ROZ" ACM in "corridor" shape.



(c) The expert moves an ACM of "AEW" usage when it conflicts with an ACM of "SSMS" usage.

Fig. 3: Expert deconfliction examples. (ROZ:Restricted Operations Zone. AEW: Airborne Early Warning Area. SSMS: Surface-to-Surface Missile System)

4. INDEPENDENT LEARNING REASONING SYSTEMS (ILRS)

The GILA system consists of four different ILRs, and each learns in a unique way. The symbolic planner learner-reasoner (SPLR) learns decision rules and value functions, and it generates deconfliction solutions by finding the best fitting rule for the input scenario. The decision-theoretic learner-reasoner (DTLR) learns a linear cost function, and it identifies solutions that are near-optimal according to the cost function. The case-based learner-reasoner (CBLR) learns and stores a feature-based case database; it also adapts and applies cases to create deconfliction solutions. The 4D-deconfliction and constraint learner-reasoner (4DCLR) learns context-sensitive, hard constraints on the schedules in the form of rules. In this section, we describe the internal knowledge representations and learning/reasoning mechanisms of these ILRs and how they work inside the GILA system.

4.1. The Symbolic Planner Learner-Reasoner (SPLR)

The symbolic planner learner and reasoner (SPLR) represents its learned solution strategy as a *hybrid hierarchical representation machine* (HHRM), and it conducts learning at two different levels. On the top level, it employs decision rule learning to learn discrete relational symbolic actions (referred to as its *directed policy*). On the bottom level, it learns a value function that is used to provide precise values for parameters in top-level actions. From communication with the SMEs, it is understood that this hybrid representation is consistent with expert reasoning in Airspace Deconfliction. Experts first choose a top-level strategy by looking at the usage of the airspaces. This type of strategy is represented as a direct policy in the SPLR. For example, to resolve a conflict involving a missile campaign, experts frequently attempt to slide the time in order to remove the conflict. This is appropriate because a missile campaign targets an exact enemy location and therefore the geometry of the missile campaign mission cannot be changed. On the other hand, as long as the missiles are delivered to the target, shifting the time by a small amount may not compromise the mission objective. Though natural for a human, reasoning of this type is hard for a machine unless a carefully coordinated knowledge base is provided. Rather, it is easier for a machine to learn to "change time" *reactively* when the "missile campaign" is in conflict. From the subject matter expert's demonstration, the machine can learn what type of deconfliction strategy is used in which types of missions. With a suitable relational language, the system can learn good reactive deconfliction strategies [Yoon et al. 2002; Yoon and Kambhampati 2007; Khardon 1999; Martin and Geffner 2000]. After choosing the type of deconfliction strategy to use, the experts decide how much to change the altitude or time, and this is mimicked by the SPLR via learning and minimizing a value function.

4.1.1. Learning Direct Policy: Relational Symbolic Actions. In order to provide the machine with a compact language system that captures an expert's strategy with little human knowledge engineering, the SPLR adopts a formal language system with Taxonomic syntax [McAllester and Givan 1993; Yoon et al. 2002] for its relational representation, and an ontology for describing the airspace deconfliction problem. The SPLR automatically enumerates its strategies in this formal language system, and seeks a good strategy. The relational action selection strategy is represented with a decision list [Rivest 1987]. A decision list DL consists of ordered rules r. In our approach, a DL outputs "true" or "false" after receiving an input action. The DL's output is the disjunction of each rule's outputs, $\bigvee r$. Each rule r consists of binary features F_r. Each of the features outputs "true" or "false" for an action. The conjunction ($\bigwedge F_r$) of them is the result of the rule for the input action.

The learning of a direct policy with relational actions is then a decision list learning problem. The expert's deconfliction actions, e.g., *move*, are the training examples. Given a demonstration trace, each action is turned into a set of binary values, which is then evaluated against pre-enumerated binary features. We used a Rivest-style decision list learning package implemented as a variation of PRISM [Cendrowska 1987] from the Weka Java library. The basic PRISM algorithm cannot cover negative examples, but our variation allows for such coverage. For the expert's selected actions, the SPLR constructs rules with "true" binary features when negative examples are the actions that were available but not selected. After a rule is constructed, examples explained (i.e., covered) by the rule are eliminated. The learning continues until there are no training examples left. We list the empirically learned direct policy example from Figure 3(a), 3(b) and 3(c) in the following:

(1) (altitude 0 (Shape ? Polygon)) & (altitude 0 (Conflict ? (Shape ? Circle))) : When an airspace whose shape is "polygon" conflicts with another airspace whose shape is "circle", change the altitude of the first airspace. Learned from Figure 3(a).
(2) (time 0 (use ? ROZ)) & (time 0 (Conflict ? (Shape ? Corridor))) : When an airspace whose usage is "ROZ" conflicts with another airspace whose shape is "corridor", change the time of the first airspace. Learned from Figure 3(b).
(3) (move 0 (use ? AEW)) & (move 0 (Conflict ? (use ? SSMS))) : When an airspace whose usage is "AEW" conflicts with another airspace whose usage is "SSMA", move the position of the first airspace. Learned from Figure 3(c).

To decide which action to take, the SPLR considers all the actions available in the current situation. If there is no action with "true" output from the DL, it chooses a random action. Among the actions with "true" output results, the SPLR takes the action that satisfies the earliest rule in the rule set. All rules are sorted according to their machine learning scores in decreasing order, so an earlier rule is typically associated with higher confidence. Ties are broken randomly if there are multiple actions with the result "true" from the same rule.

4.1.2. Learning a Value Function. Besides choosing a strategic deconfliction action, specific values must be assigned. If we opted to change the time, by how much should it be changed? Should we impose some margin beyond the deconfliction point? How much margin is good enough? To answer these questions, we consulted the expert demonstration trace, which has records concerning the margin. We used linear regression to fit the observed margins. The features used for this regression fit are the same as those used for direct policy learning; thus, feature values are Boolean. The intuition is that the margins generally depend on the mission type. For example, missiles need a narrow margin because they must maintain a precise trajectory. The margin representation is a linear combination of the features, e.g., Move Margin = $\sum w_i \times F_i$ (margin of move action). We learned weights w_i with linear regression.

4.1.3. Learning a Ranking Function. Besides the hierarchal learning of deconfliction solutions, the SPLR also learns a ranking function $\mathcal{R}$ to prioritize conflicts. The SPLR learns this ranking function $\mathcal{R}$ using decision tree learning. First, experts show the order of conflicts during demonstration. Each pair of conflicts is then used as a training example. An example is true if the first member of a pair is given priority over the second. After learning, for each pair of conflicts (c_1, c_2), the learned decision tree answers "true" or "false." "True" means that the conflict c_1 has higher priority. The rank of a conflict is the sum of "true" values against all the other conflicts, with ties broken randomly. The rank of a conflict is primarily determined by the missions involved. For example, in Table II, the first conflict that involved a missile corridor (SSMS) is given a high priority due to the sensitivity to changing a missile corridor.

4.2. The 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 a cost function subject to some constraints. The goal of the DTLR is to learn a close approximation of the expert's cost function, and this learning problem is approached as an instance of structured prediction [Bakir et al. 2007]. Once the cost function is learned, the performance algorithm of the DTLR uses this function to try to find a minimal cost solution with iterative-deepening search.

4.2.1. Learning a Cost Function via Structured Prediction. The cost function learning is formalized in the framework of structured prediction. A structured prediction problem is defined as a tuple $\{\mathcal{X}, \mathcal{Y}, \Psi, L\}$, where $\mathcal{X}$ is the *input space* and $\mathcal{Y}$ is the output space. In the learning process, a *joint feature function* $\Psi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^n$ defines the joint features on both inputs and outputs. The *loss function*, $L : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re$, quantifies the relative preference of two outputs given some input. Formally, for an input $\mathbf{x}$ and two outputs $\mathbf{y}$ and $\mathbf{y}'$, $L(\mathbf{x}, \mathbf{y}, \mathbf{y}') > 0$ if $\mathbf{y}$ is a better choice than $\mathbf{y}'$ given input $\mathbf{x}$ and $L(\mathbf{x}, \mathbf{y}, \mathbf{y}') \leq 0$ otherwise. We use a margin-based loss function used in the *logitboost* procedure [Friedman et al. 1998], defined as $L(\mathbf{x}, \mathbf{y}, \mathbf{y}') = log(1 + e^{-m})$, where m is the margin of the training example (see Section 4.2.2 for details).

The decision surface is defined by a linear scoring function over the joint features $\Psi(\mathbf{x}, \mathbf{y})$ given by the inner product $\langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle$, where $\mathbf{w}$ is the vector of learned model parameters, and the best $\mathbf{y}$ for any input $\mathbf{x}$ has the highest score. The specific goal of learning is, then, to find $\mathbf{w}$ such that $\forall i : \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle = \mathbf{y}_i$.

In the case of ACO scheduling, an input drawn from this space is a combination of an ACO and a deconflicted ACM to be scheduled. An output $\mathbf{y}$ drawn from the output space $\mathcal{Y}$ is a schedule of the deconflicted ACM. The joint features are x-y coordinate change, altitude change and time change for each ACM, and other features such as changes in the number of intersections of the flight paths with the enemy territory.

4.2.2. Structured Gradient Boosting (SGB) for Learning a Cost Function. The DTLR's *Structured Gradient Boosting* (SGB) algorithm [Parker et al. 2006] is a gradient descent approach to solving the structured prediction problem. Suppose that there is some training example $\mathbf{x}_i \in \mathcal{X}$ with the correct output $\mathbf{y}_i \in \mathcal{Y}$, $\widehat{\mathbf{y}}_i$ defined as the highest scoring incorrect output for $\mathbf{x}_i$ according to the current model parameters. That is,

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

Then the *margin* is defined as the amount by which the model prefers $\mathbf{y}_i$ to $\widehat{\mathbf{y}}_i$ as an output for $\mathbf{x}_i$. The margin m_i for a given training example is $m_i = \langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle$, where $\delta\,(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)$ is the difference between the feature vectors $\Psi\,(\mathbf{x}_i, \mathbf{y}_i)$ and $\Psi\,(\mathbf{x}_i, \widehat{\mathbf{y}}_i)$. The margin determines the loss as shown in Step 5 of the pseudo code of Algorithm 3. The parameters of the cost function are adjusted to reduce the gradient of the cumulative loss over the training data (see [Parker et al. 2006] for more details).

ALGORITHM 3: STRUCTURED GRADIENT BOOSTING

Input: $\{\langle x_i, y_i \rangle\}$: the set of training examples;
B: the number of boosting iterations.
Output: weights of the cost function $\mathbf{w}$.

```
1:  Initialize the weights $\mathbf{w}$ = 0
2:  **repeat**
3:      **for** each training example $(x_i, y_i)$ **do**
4:          solve *Argmax:* $\widehat{\mathbf{y}}_i = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle$
5:          compute *training loss:* $L(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \log(1 + e^{-m_i})$, where $m_i = \langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle$
6:      **end for**
7:      compute *cumulative training loss:* $L = \sum_{i=1}^{n} L(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)$
8:      find the gradient of the cumulative training loss $\nabla L$
9:      update *weights:* $\mathbf{w} = \mathbf{w} - \alpha \nabla L$
10: **until** convergence or $B$ iterations
11: **return** the weights of the learned cost function $\mathbf{w}$
```

The problem of finding $\widehat{\mathbf{y}}_i$, which is encountered during both learning and performance, is called the *Argmax* problem. A discretized space of operators, namely, the altitude, time, radius and x-y coordinates, is defined based on the domain knowledge to produce various possible plans to deconflict each ACM. A simulator is used to understand the effect of a deconfliction plan. Based on their potential effects, these plans are evaluated using the model parameters, and the objective is to find the best scoring plan that resolves the conflict. Exhaustive search in this operator space would be optimal for producing high-quality solutions, but has excessive computational cost. Iterative Deepening Depth First (IDDFS) search is used to find solutions by considering single changes before multiple changes, and smaller amounts of changes before larger amounts of changes, thereby trading off the quality of the solution with the search time. Note that the length of the proposed deconfliction plan (number of changes) is getting iterative deepened in IDDFS. Since a fixed discrete search space of operators is used, the search time is upper bounded by the time needed to search the entire space of operators.

4.2.3. Illustration of Gradient Boosting. Figure 4 provides a geometrical illustration of the gradient boosting algorithm. There are four training examples $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and (x_4, y_4). As explained in the previous section, the features depend on both the input x and the output y, i.e., $\Psi(x, y)$. The data points are represented corresponding to features $\Psi(x_i, y_i)$ of the training examples x_i with respect to their true outputs y_i with $\oplus$, i.e., positive examples and data points corresponding to features $\Psi(x_i, \hat{y}_i)$ of the training examples x_i with respect to their best scoring outputs $\hat{y}_i$ with $\ominus$, i.e., negative examples. Note that the locations of the positive points do not change, unlike the negative points whose locations change from one iteration to another, i.e., the best scoring negative outputs $\hat{y}_i$ change with the weights, and hence the feature vectors of the negative examples $\Psi(x_i, \hat{y}_i)$ change. Three boosting iterations of the DTLR learning algorithm are shown in Figure 4, one row per iteration. In each row, the left figure shows the current hyperplane (cost function) along with the negative examples according to the current cost function, and the right figure shows the cost function obtained after updating the weights in a direction that minimizes the cumulative loss over all training examples, i.e., a hyperplane that separates the positive examples from negative

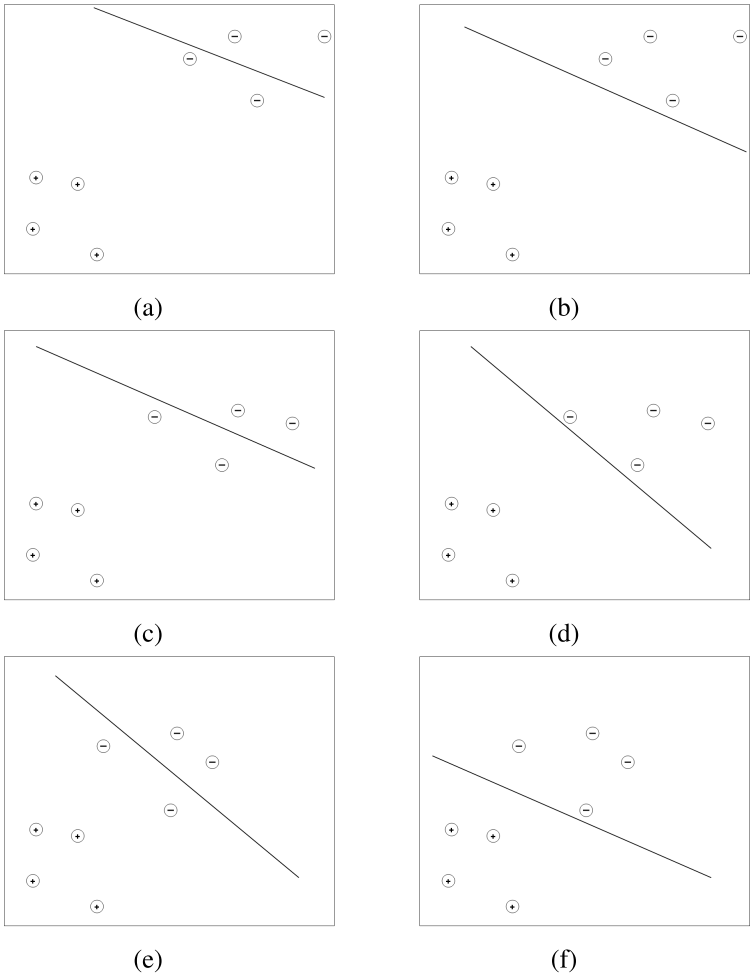


Fig. 4: In all of these figures, $\oplus$ corresponds to the feature vectors of the training examples with respect to their true outputs, $\ominus$ corresponds to the feature vectors of the training examples with respect to their best scoring negative outputs and the separating hyperplane corresponds to the cost function. (a) Initial cost function and negative examples before 1st iteration; (b) Cost function after 1st boosting iteration; (c) Cost function and negative examples before 2nd iteration; (d) Cost function after 2nd boosting iteration; (e) Cost function and negative examples before 3rd iteration; (f) Cost function after 3rd boosting iteration.

ones (if such a hyperplane exists). As the boosting iterations increase, our cost function is moving towards the true cost function and it will eventually converge to the true cost function.

4.2.4. What Kind of Knowledge Does the DTLR Learn?. We explain, through an example case, what was learned by the DTLR from the expert's demonstration and how the knowledge was applied while solving problems during performance mode. Before training, weights of the cost function are initialized to zero. For each ACM that was moved to resolve conflicts, a training example is created for the gradient boosting algorithm. The expert's plan that deconflicts the problem ACM corresponds to the correct solution for each of these training examples. Learning is done using the gradient boosting algorithm described before, by identifying the highest scoring incorrect solution and computing a gradient update to reduce its score (or increase its cost). For example, in one expert's demonstration that was used for training, all the deconflicting plans are either altitude changes or x-y coordinate changes. Hence, the DTLR learns a cost function that prefers altitude and x-y coordinate changes to time and radius changes.

During performance mode, when given a new conflict to resolve, the DTLR first tries to find a set of deconflicting plans using Iterative Deepening Depth First (IDDFS) search. Then, it evaluates each of these plans using the learned cost function and returns the plan with minimum cost. For example, in Scenario F, when trying to resolve conflicts for *ACM-J-15*, it found six plans with a single (minimum altitude) change and preferred the one with minimum change by choosing to increase the minimum altitude by 2000 (as shown on row #9 in Table II).

4.3. The Case-Based Learner-Reasoner (CBLR)

Case-based reasoning (CBR) [Aamodt and Plaza 1994] consists of solving new problems by reasoning about past experience. Experience in CBR is retained as a collection of *cases* stored in a case library. Each of these cases contains a past problem and the associated solution. Solving new problems involves identifying relevant cases from the case library and reusing or adapting their solutions to the problem at hand. To perform this adaptation process, some CBR systems, such as the CBLR, require additional adaptation knowledge.

Figure 5 shows the overall architecture of the CBLR, which uses several specialized case libraries, one for each type of problem that the CBLR can solve. A *Prioritization Library* contains a set of cases for reasoning about the priority, or order, in which conflicts should be solved. A *Choice Library* is used to determine which ACM will be moved, given a conflict between two ACMs. Finally, a *Constellation Library* and a *Deconfliction Library* are used within a hierarchical process. The *Constellation Library* is used to characterize the neighborhood surrounding a conflict. The neighborhood provides information that is then used to help retrieve cases from the *Deconfliction Library*. For each of these libraries, the CBLR has two learning modules: one capable of learning cases and one capable of learning adaptation knowledge. The case-based learning process is performed by observing an expert trace, extracting the problem descriptions, features and solutions, and then storing them as cases in a case library. Adaptation knowledge in the CBLR is expressed as a set of transformation rules and a set of constraints. Adaptation rules capture how to transform the solution from the retrieved case to solve the problem at hand, and the constraints specify the combinations of values that are permitted in the solutions being generated.

4.3.1. Learning in the CBLR. Each case library contains a specialized case learner, which learns cases by extracting them from an expert trace. Each case contains a problem description and an associated solution. Figure 6 shows sample cases learned from the expert trace.

Prioritization

Using information from the expert trace available to the CBLR, the Priority Learner constructs prioritization cases by capturing the order in which the expert prioritizes the conflicts in the trace. From this, the CBLR learns prioritization cases, storing one case per conflict. Each case contains a description of the conflict, indexed by its features, along with the priority assigned by the expert. The CBLR uses these cases to build a ranking function $\mathcal{R}$ to provide the MRE with a priority order for deconfliction. The order in which conflicts are resolved can have a significant impact on the quality of the overall solution.

Choice

Given a conflict between two ACMs, the CBLR uses the Choice Case Library to store the identifier of the ACM that the expert chose to modify. Each time the expert solves a conflict in the trace, the CBLR learns a choice case. The solution stored with the case is the ACM that is chosen to be moved. The two conflicting ACMs and the description of the conflict are stored as the problem description for the case.

Hierarchical Deconfliction

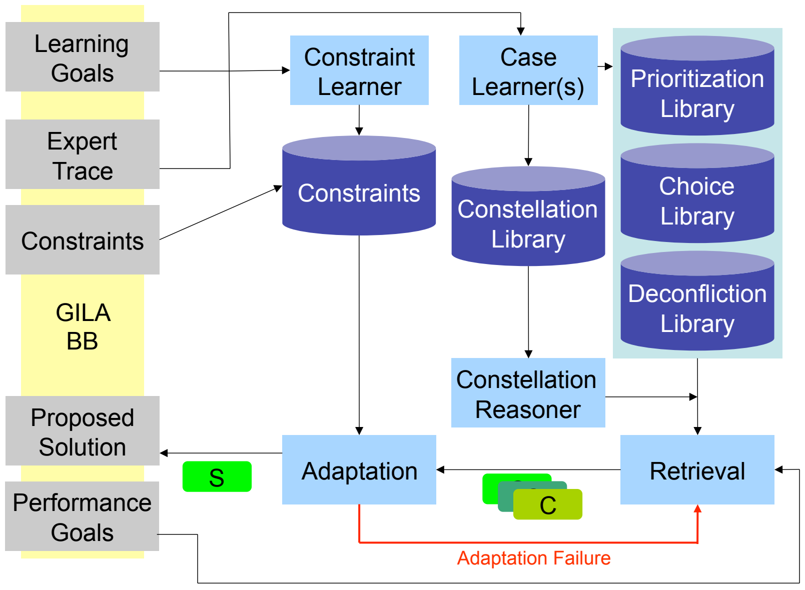
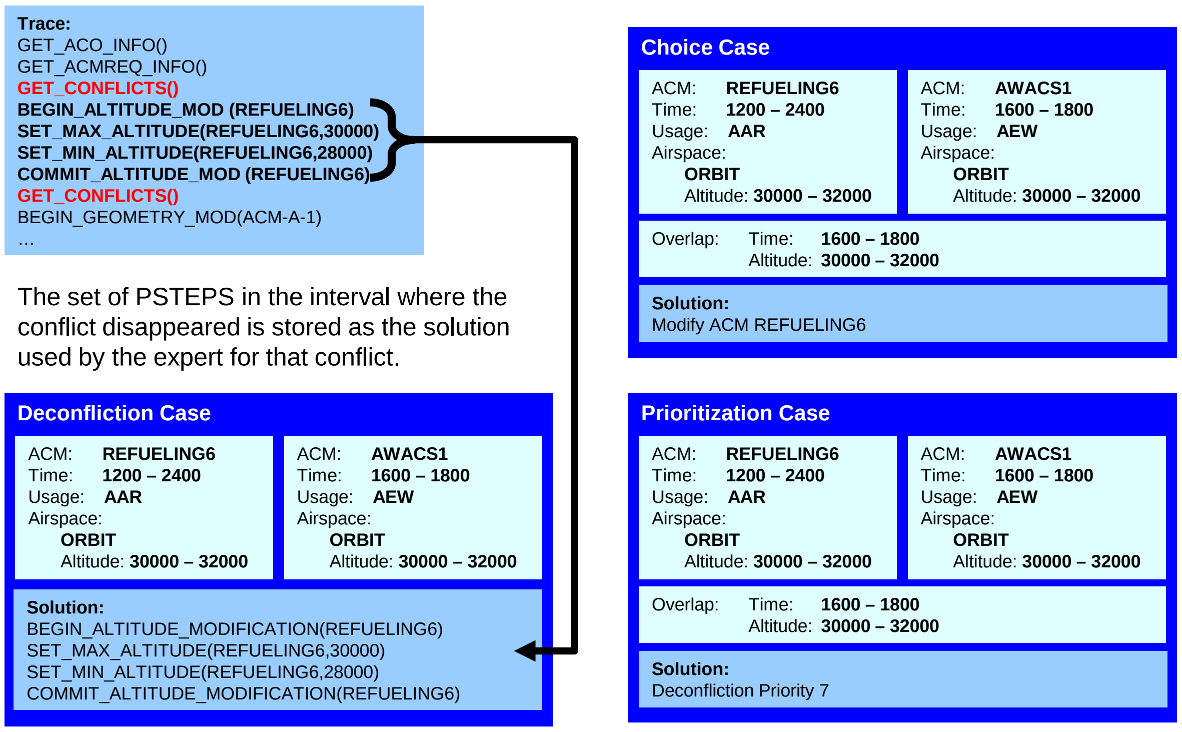


Fig. 5: The architecture of the Case-Based Learner-Reasoner (CBLR)



Trace:
GET_ACO_INFO()
GET_ACMREQ_INFO()
GET_CONFLICTS()
BEGIN_ALTITUDE_MOD (REFUELING6)
SET_MAX_ALTITUDE(REFUELING6,30000)
SET_MIN_ALTITUDE(REFUELING6,28000)
COMMIT_ALTITUDE_MOD (REFUELING6)
GET_CONFLICTS()
BEGIN_GEOMETRY_MOD(ACM-A-1)
...

The set of PSTEPS in the interval where the conflict disappeared is stored as the solution used by the expert for that conflict.

Choice Case

ACM: **REFUELING6** Time: **1200 – 2400** Usage: **AAR** Airspace: **ORBIT** Altitude: **30000 – 32000**	ACM: **AWACS1** Time: **1600 – 1800** Usage: **AEW** Airspace: **ORBIT** Altitude: **30000 – 32000**

Overlap: Time: **1600 – 1800**
Altitude: **30000 – 32000**

Solution:
Modify ACM REFUELING6

Deconfliction Case

ACM: **REFUELING6** Time: **1200 – 2400** Usage: **AAR** Airspace: **ORBIT** Altitude: **30000 – 32000**	ACM: **AWACS1** Time: **1600 – 1800** Usage: **AEW** Airspace: **ORBIT** Altitude: **30000 – 32000**

Solution:
BEGIN_ALTITUDE_MODIFICATION(REFUELING6)
SET_MAX_ALTITUDE(REFUELING6,30000)
SET_MIN_ALTITUDE(REFUELING6,28000)
COMMIT_ALTITUDE_MODIFICATION(REFUELING6)

Prioritization Case

ACM: **REFUELING6** Time: **1200 – 2400** Usage: **AAR** Airspace: **ORBIT** Altitude: **30000 – 32000**	ACM: **AWACS1** Time: **1600 – 1800** Usage: **AEW** Airspace: **ORBIT** Altitude: **30000 – 32000**

Overlap: Time: **1600 – 1800**
Altitude: **30000 – 32000**

Solution:
Deconfliction Priority 7

Fig. 6: A set of sample cases learned from the expert trace.

The CBLR uses a hierarchical reasoning process to solve a conflict using a two-phase approach. The first phase determines what method an expert is likely to use when solving a deconfliction problem. It does this by describing the "constellation" of the conflict. "Constellation" refers to the neighborhood of airspaces surrounding a conflict. The choices for deconfliction available to an airspace manager are constrained by the neighborhood of airspaces, and the Constellation Case Library allows the CBLR to mimic this part of an expert's decision-making process. A constellation case consists of a set of features that characterize the degree of congestion in each dimension (latitude-longitude, altitude and time) of the airspace. The solution stored is the dimension within which the expert moved the ACM for deconfliction (e.g., change in altitude, orientation, rotation, radius change, etc.).

The second CBLR phase uses the Deconfliction Case Library to resolve conflicts once the deconfliction method has been determined. A deconfliction case is built from the expert trace by extracting the features of each ACM. This set of domain-specific features was chosen manually based on the decision criteria of human experts. In addition to these two sets of features (one for each of the two conflicts in a pair), a deconfliction case includes a description of the overlap.

The solution is a set of PSTEPs that describe the changes to the ACM, as illustrated in Figure 6. A PSTEP is an atomic action in a partial plan that changes an ACM. These PSTEPs represent the changes that the expert made to the chosen ACM in order to resolve the conflict. Whereas the constellation phase determines an expert's likely deconfliction method, the deconfliction phase uses that discovered method as a highly-weighted feature when searching for the most appropriate case in the deconfliction case library. It then retrieves the most similar case based on the overall set of features and adapts that case's solution to the new deconfliction problem. We refer to this two-phase process as *hierarchical deconfliction*.

In order to learn adaptation knowledge, the CBLR uses transformational plan adaptation [Muñoz-Avila and Cox 2007] to adapt deconfliction strategies, using a combination of adaptation rules and constraints. Adaptation rules are built into the CBLR. This rule set consist of five common-sense rules that are used to apply a previously successful deconfliction solution from one conflict to a new problem. For example, "If the overlap in a particular dimension between two airspaces is X, and the expert moved one of them X+Y units in that dimension, then in the adapted PSTEP we should compute the overlap Z and move the space Z+Y units." If more than one rule is applicable to adapt one PSTEP, the adaptation module will propose several candidate adaptations, as explained later.

During the learning process, one challenge in extracting cases from a demonstration trace involves the identification of the sequence of steps that constitutes the solution to a particular conflict. The expert executes steps using a human-centric interface, but the resulting sequence of raw steps, which is used by the ILRs for learning, does not indicate which steps apply to which conflict. The CBLR overcomes this limitation by executing each step in sequence, comparing the conflict list before and after each step to determine if a conflict was resolved by that step.

Constraints are learned from the expert trace both by the CBLR and by the Constraint Learner (CL) inside the 4DCLR. To learn constraints, the CBLR evaluates the range of values that the expert permits. For instance, if the expert sets all altitudes of a particular type of aircraft to some value between 10,000 and 30,000 feet, the CBLR will learn a constraint that limits the altitude of aircraft type to a minimum of 10,000 feet and a maximum of 30,000 feet. These simple constraints are learned automatically by a constraint learning module inside the CBLR. This built-in constraint learner makes performance more efficient by reducing the dependence of the CBLR on other modules. However, the CL in the 4DCLR is able to learn more complex and accurate constraints, which are posted to the blackboard of GILA, and these constraints are used by the CBLR to enhance the adaptation of its solutions.

4.3.2. Problem Solving in the CBLR. The CBLR uses all the knowledge it has learned (and stored in the multiple case libraries) to solve the airspace deconfliction problems. The CBLR is able to solve a range of problems posted by the MRE. For each of the case-retrieval processes, the CBLR uses a weighted Euclidean distance to determine which cases are most similar to the problem at hand. The weights assigned to each feature are based on the decision-making criteria of human experts, and were obtained via interviews with subject-matter experts. Automatic feature weighting techniques [Wettschereck et al. 1997] were evaluated, but without good results given the limited amount of available data.

During the performance phase of GILA, a prioritization problem is sent by the MRE that includes a list of conflicts, and the MRE asks the ILRs to rank them by priority. The assigned priorities will determine the order in which conflicts will be solved by the system. To solve one such problem, the CBLR first assigns priorities to all the conflicts in the problem by assigning each conflict the priority of the most similar priority case in the library. After that, the conflicts are ranked by priority (and ties are solved randomly).

Next, a deconfliction problem is presented to each of the ILRs so that they can provide a solution to a single conflict. The CBLR responds to this request by producing a list of PSTEPs. It solves a conflict using a three-step process. First, it decides which ACM to move using the choice library. It then retrieves the closest match from the Constellation Library and uses the associated solution as a feature when retrieving cases from the Deconfliction Library. It retrieves and adapts the closest n cases (where $n = 3$ in our experiments) to produce candidate solutions. It tests each candidate solution by sending it to the 4DCLR module, which simulates the application of that solution. The CBLR evaluates the results and selects the best solutions, that is, those that solve the target conflict with the lowest cost. A subset of selected solutions is sent to the MRE as the CBLR's solutions to the conflict.

Adaptation is only required for deconfliction problems, and is applied to the solution retrieved from the deconfliction library. The process for adapting a particular solution S, where S is a list of PSTEPs, as shown in Figure 6, works as follows:

(1) Individual PSTEP adaptation: Each individual PSTEP in the solution S is adapted using the adaptation rules. This generates a set of candidate adapted solutions AS.
(2) Individual PSTEP constraint checking: Each of the PSTEPs in the solutions in AS is modified to comply with all the constraints known by the CBLR.
(3) Global solution adaptation: Adaptation rules that apply to groups of PSTEPs instead of to individual PSTEPs are applied; some unnecessary PSTEPs may be deleted and missing PSTEPs may be added.

4.3.3. CBLR Results. During GILA development, we were required to minimize encoded domain knowledge and maximize machine learning. One strength of the case-based learning approach is that it learns to solve problems in the same way that the expert does, with very little pre-existing knowledge. During the evaluation, we performed "Garbage-in/Garbage-out" (GIGO) experiments that tested the learning nature of each module by teaching the system with incorrect approaches, then testing the modules to confirm that they used the incorrect methods during performance. This technique was designed to test that the ILR used knowledge that was learned rather than encoded. The case-based learning approach successfully learned these incorrect methods and applied them in performance mode. This shows that the CBLR learns from the expert trace, performing and executing very much like the expert does. This also allows the CBLR to learn unexpected solution approaches when they are provided by an expert. If such a system were to be transitioned with a focus on deconfliction performance (rather than machine learning performance), domain knowledge would likely be included.

The CBLR also responded very well to incremental learning tests. In these tests the system was taught one approach to problem solving at a time (Table V). When the system was taught to solve problems using only altitude changes, the CBLR responded in performance mode by attempting to solve all problems with altitude changes. When the system was taught to solve problems by making geometric changes, the CBLR responded in performance mode by using both of these methods to solve problems, confirming that the CBLR's problem-solving knowledge was learned rather than being previously stored as domain knowledge.

Moreover, the different ILRs in GILA exhibit different strengths and weaknesses, and the power of GILA consists exactly of harnessing the strong points of each ILR. For instance, one of the CBLR's strengths is that its performance during prioritization was close to the expert's prioritization. For this reason, the CBLR priorities were used to drive the deconfliction order of the GILA system in the final system evaluation.

4.4. The 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. A safety constraint example is: "The altitude of a UAV over the course of its trajectory should never exceed a maximum of 60000 feet". Constraints are "hard" in the sense that they can never be violated, but they are also context-sensitive, where the "context" is the task mission as exemplified in the ACO. For instance, a recommended minimum altitude of an aircraft may be raised if the problem being solved involves the threat of enemy surface-to-air missiles.

The 4DCLR consists of the following two components: (1) the *Constraint Learner (CL)*, which automatically infers safety constraints from the expert demonstration trace and outputs the constraints for use by the other ILRs in the context of planning, and (2) the *Safety Checker (SC)*, which is responsible for verifying the correctness of solutions/plans in terms of their satisfaction or violation of the safety constraints learned by the CL. The output of the Safety Checker is a degree of violation, which is used by the MRE in designing safe subproblem solutions.

The approach adopted in the 4DCLR is 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]. The Safety Checker, in particular, is related to formal verification, such as model checking [Clarke et al. 1999]. However, unlike traditional verification, which outputs a binary "success/failure," our GILA Safety Checker outputs a *degree* of constraint violation (failure). This is analogous to what is done in [Chockler and Halpern 2004]. The difference is that when calculating "degree" we not only calculate the probabilities over alternative states as Chockler and Halpern do, but we also account for physical distances and constraints.

4.4.1. The Constraint Learner and the Representations It Uses. We assume that the system designer provides constraint templates a priori, and it is the job of the Constraint Learner (CL) to infer the values of parameters within these templates. For example, a template might state that a fighter has a maximum allowable altitude, and the CL would infer what the value of that maximum should be. In the future, the CL will learn the templates as well.

Learning in the CL is Bayesian. A probability distribution is used to represent the uncertainty regarding the true value of each parameter. For each parameter, such as the maximum flying altitude for a particular aircraft, the CL begins with a prior probability distribution, $P_{f(c)}(\omega)$ or $P_{g(c_1,c_2)}(\omega)$, where c, c_1, $c_2 \in \mathcal{C}, f \in \mathcal{F}, g \in \mathcal{G}$, and ω is a safety constraint. 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, e, 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 applying Bayes' Rule to obtain a posterior distribution, $P_{f(c)}(\omega|e)$ or $P_{g(c_1,c_2)}(\omega|e)$, given the assumption for the likelihood 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. With a modest amount of evidence, this approach leads to tight distributions over the constraint parameters.

4.4.2. The Safety Checker and Its Outputs for the MRE. The Safety Checker (SC) takes candidate subproblem solutions from the ILRs as input, 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 by the candidate solution. A violation message is output by the SC that includes the violated constraint, the solution that violated the constraint, specific information about the nature of the violation in the context of the ACO 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*, or simply the *violation penalty*. The SC calculates this penalty by finding a normalized expected amount of violation, based on the constraint posterior distribution learned by the CL. Let $P_{f(c)}(\omega|E)$ represent the posterior distribution over the safety constraint governing property f applied to concept c, given expert trace E. An example might be the maximimum altitude property of the "Combat Air Patrol" airspace concept. Given a proposed solution that involves repositioning an airspace matching concept c, let v represent $f(c)$ in that solution (e.g., let it represent the maximum altitude of a "Combat Air Patrol" in the proposed solution). Then the safety violation penalty is calculated as:

$$penalty_{unnormalized} = \int P_{f(c)}(\omega|E) \cdot max(0, (v - \omega))d\omega\ . \tag{1}$$

For a minimum threshold, the unnormalized penalty would be:

$$penalty_{unnormalized} = \int P_{f(c)}(\omega|E) \cdot max(0, (\omega - v))d\omega\ . \tag{2}$$

The method is identical for relational constraints g. The unnormalized penalty is normalized based on the range of possible parameter values, so that violations in different dimensions (altitude versus horizontal distance versus time) can be compared (additional details in [Rebguns et al. 2009]). The MRE uses the violation penalty to discard subproblem solutions that are invalid because their penalty is above the *safety threshold*.

Why is the SC needed if the ILRs already use the safety constraints during planning? The reason is that the ILRs do not interact with one another during planning. Because each ILR may not have the domain knowledge, representational expressiveness, or learning and planning capabilities to solve the entire input problem, the ILRs output subproblem solutions, which are partial and incomplete solutions. The MRE subsequently composes these subproblem solutions into one final complete solution using search. This final solution needs to be checked because interactions between the subproblem solutions will not emerge until after they have been composed into a single solution, and these interactions might violate constraints.

5. THE META-REASONING EXECUTIVE (MRE)

In both the practice learning phase and the collaborative performance phase (see Figure 2), the system is required to solve a test problem using the learned knowledge. The Meta-Reasoning Executive (MRE) directs a collaborative performance process (Algorithm 1) during which the ILRs contribute to solving the test problem. This collaborative performance process is modeled as a search for a path from the initial state to a goal state (where the problem is fully solved). The complete solution is a combination of the partial solutions contributed by each ILR.

First, the given test problem is decomposed into a set of subproblems. In general, problem decomposition is a difficult task, and the quality of problem solving depends on how the problem is decomposed. In this application, GILA uses domain knowledge to decompose the original problem: given an ACO and a set of proposed ACMs, solving the problem consists of removing all existing conflicts; so the whole problem is then decomposed as a set of subproblems, and the purpose of each subproblem is to remove one conflict. These subproblems are interrelated, i.e., how a subproblem is solved may affect how others can be solved. Solving one subproblem can also generate new subproblems. To manage these interrelationships, the MRE conducts an internal search process,

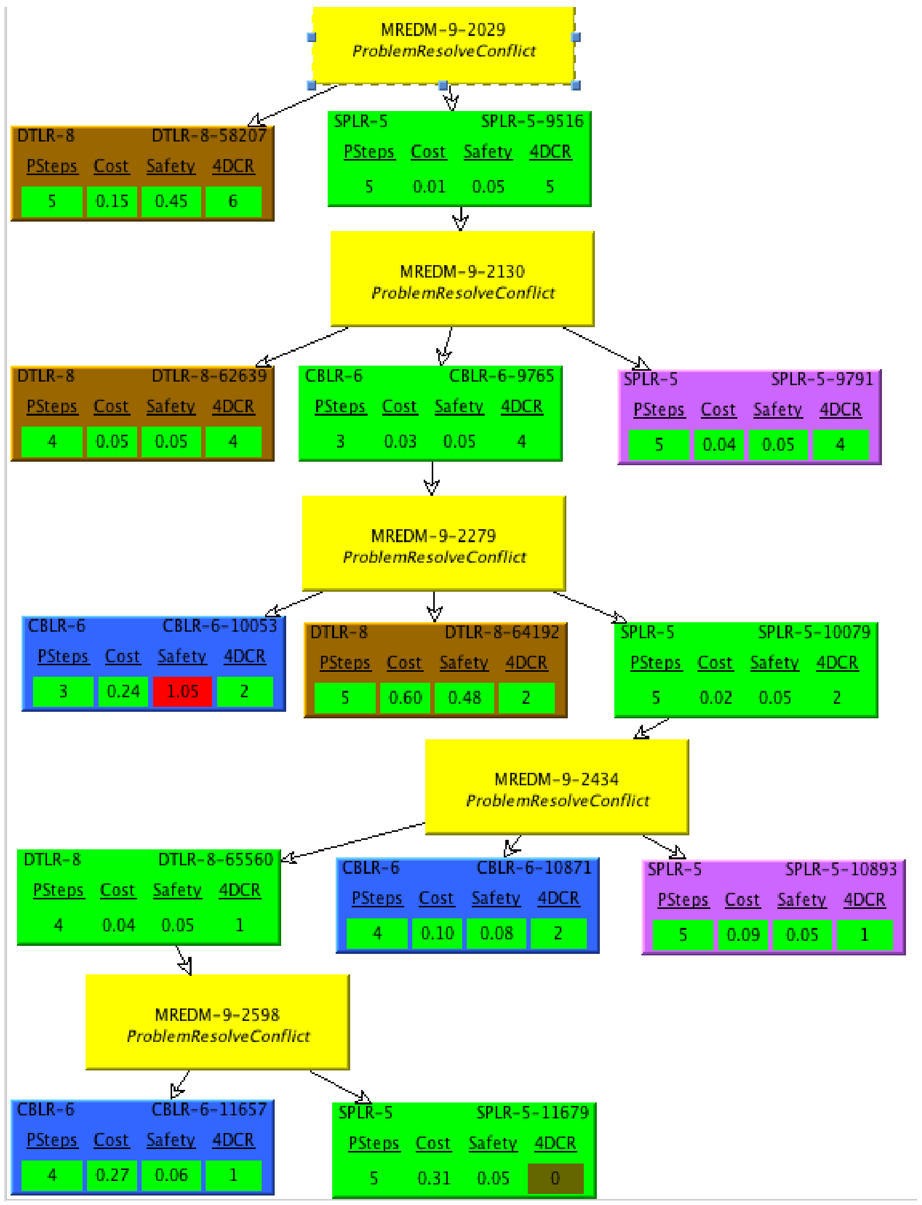


Fig. 7: Partial search tree example - GILA performs on Scenario F.

as described below. In addition, one ILR in particular – the CBLR – learns the priority of these subproblems and provides guidance on the ordering to solve them.

Next, the MRE posts these subproblems on the blackboard, and each ILR then posts its solutions to some of these subproblems. These subproblem solutions are treated as the search operators available at the current state. They are applied to the current state, which results in new states. New conflicts may appear after applying a subproblem solution. These new states are then evaluated and stored in an open list. The best state is selected to be explored next: if it is a goal state (no remaining conflicts), the problem is fully solved; otherwise, the MRE posts all subproblems that correspond to conflicts existing in this current state, and the previous process is repeated.

Figure 7 shows part of the search tree constructed when GILA is performing on Scenario F after learning from the expert demonstration that solves Scenario E, and practice on Scenario G. The nodes marked with “MREDM-9-XXXX” represent problem states and the nodes marked with “CBLR-6,” “SPLR-5” or “DTLR-8” represent subproblem solutions (sequences of ACM modifications) posted by the ILRs. The problem state node and subproblem solution nodes alternate. If a problem state node represents the problem state s, and one of its child nodes is a subproblem solution sol selected for exploration, a new problem state node is generated representing the result of applying sol to s. The ordering of nodes to be explored depends on the search strategy. A best-first search strategy is used in this work. The node n that contains state s with the best evaluation score $\mathcal{E}(s)$ is selected from the open list and explored next.

This search process is directed by the learned knowledge from the ILRs in the following two ways. First, GILA learns a ranking function $\mathcal{R}$ to decide which subproblems to work on initially. It is not efficient to have all ILRs provide solutions to all subproblems, as it takes more time to generate those subproblem solutions and also requires more effort to evaluate them. Because solving one subproblem could make solving the remaining problems easier or more difficult, it is crucial to direct the ILRs to work on subproblems in a facilitating order. Though multiple ILRs are learning this ranking function, the CBLR is the best one for this task. In the beginning of the search process, the MRE asks the CBLR to provide a priority ordering of the subproblems. Based on this priority list, the MRE suggests which subproblem to work on first. This suggestion is taken by all the ILRs as guidance to generate solutions for subproblems. The ILRs work on the subproblems simultaneously.

Second, GILA learns an evaluation function $\mathcal{E}$ to evaluate the problem state resulting from applying a proposed subproblem solution to the current state. This evaluation function $\mathcal{E}$ is constructed using the learned knowledge from the ILRs in the following ways:

(1) The Safety Checker (SC) checks for safety violations in a problem state. Some subproblem solutions may cause safety violations that make them invalid. The Safety Checker determines whether there is a safety violation and, if yes, how severe the violation is, which is represented by the violation penalty. If the violation penalty is greater than the safety threshold, the MRE discards the subproblem solution that causes this violation. For example, as shown in Figure 7, there is a red box for safety value “1.05” on Node “CBLR-6-10053.” The red safety box means that the problem state represented by this node has a violation penalty greater than the safety threshold (1.0 in this example); hence, the corresponding subproblem solution is discarded. Otherwise, if the violation penalty is less than or equal to the safety threshold, the following evaluations are performed.
(2) The DTLR derives the execution cost for a new problem state after applying a subproblem solution. The execution cost is learned by the DTLR to measure how close this subproblem solution is to the expert’s demonstration. Ideally, GILA is looking for a solution that best mimics the expert, which is a solution with minimal execution cost. However, this cost estimation is not exact due to various assumptions in its learning, such as discretization of the action space and inexact inference.
(3) Another ILR, the 4DCLR, performs an internal simulation to investigate the results of applying a subproblem 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, which is the state with no remaining conflicts.

If a subproblem solution does not solve any conflict at all, it is discarded; otherwise, the new problem state resulting from applying this subproblem solution is evaluated based on the following factors: the cost of executing all subproblem solutions selected on this path from the initial state to this current state ($cumulative\_exec\_cost$), safety violation penalties that would be present if the path were executed ($safety\_penalties$), and the estimated execution cost and violation penalties from this current state to a goal state, in other words, to resolve the remaining conflicts ($estimated\_remaining\_cost$). These factors are combined using a linear function with a set of weight parameters:

(1) execution.cost.weight (w1)
(2) safety.violations.weight (w2)
(3) solved.to.remaining.conflict.balance.weight (w3)

The estimation of execution cost and violation penalties for resolving the remaining conflicts is calculated as:

$$estimated\_remaining\_cost = \frac{actual\_cost}{number\_of\_resolved\_conflicts} * number\_of\_remaining\_conflicts$$

$$actual\_cost = w1 * cumulative\_exec\_cost + w2 * safety\_penalties$$

The $estimated\_total\_cost$ is calculated as:

$$estimated\_total\_cost = w3 * actual\_cost + (1 - w3) * estimated\_remaining\_cost$$

The values of the weight parameters $w1$, $w2$, and $w3$ can be varied to generate different search behaviors. Based on the $estimated\_total\_cost$, the MRE determines the ordering of nodes to be explored. The search process stops when a goal state is reached, i.e., when there are no remaining conflicts, or a preset time limit is reached. In Figure 7, there is a brown box with the 4DCR value of "0" on Node "SPLR-5-11679." This node represents a goal state because the 4DCLR reports that the number of remaining conflicts in the current state is "0."

6. EXPERIMENTAL SETUP AND RESULTS

The GILA system consists of an ensemble 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. The GILA system is composed of distributed GILA Nodes, which contain and manage the GILA components. Each node runs in parallel and the components (e.g., ILRs, MRE) are multithreaded within the node. Each node efficiently shares OWL data via the networked blackboard, and the MRE can queue problems for the ILRs, minimizing any idle time. The deployment of components to GILA Nodes is configurable – to optimize performance and scalability. There is no logical limit to the number of nodes or components in the GILA system.

6.1. Test Cases and Evaluation Criteria

The test cases for experiments were developed by subject matter experts from BlueForce, LLC. The experimental results were graded by these SMEs. One expert did the majority of the work, with help from one or two other experts. In the remainder of this section, we use "the expert" to refer to this group. Notice that the expert is independent of the GILA team and is not involved in designing the GILA system.

For the final evaluation, four scenarios, D, E, F and G, were developed. The evaluator randomly chose three of them, namely, E, F and G.In each test scenario, there are 24 Airspace Control Measures (ACMs). There are 14 conflicts among these ACMs as well as existing airspaces. Each test case consists of three test scenarios 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 an airspace. The quality of each step (action) inside the solution is judged according the following factors:

Table II: GILA's and an expert's deconfliction solutions for Scenario F

Expert's priority	Conflict	Expert's Solution	GILA Solution	GILA's priority
1	ACM-J-18 (SSMS) ACM-J-19 (SOF)	Move 19 from 36.64/-116.34, 36.76/-117.64 to 36.64/-116.34, 36.68/-117.54	Move 19 from 36.64/-116.34, 36.76/-117.64 to 36.6/-116.18, 36.74/-117.48	4
2	ACM-J-17 (SSMS) ACM-J-19 (SOF)	Already resolved	Move 17 from 37.71/-117.24, 36.77/-117.63 to 37.77/-117.24, 36.84/-117.63	1
3	ACM-J-12 (CASHA) ACM-J-23 (UAV)	Change alt of 12 from 16000-35500 to 17500-35500	Change alt of 23 from 1500-17000 to 1500-15125	7
4	ACM-J-11 (CASHA) ACM-J-24 (UAV)	Move 11 from 36.42/-117.87, 36.42/-117.68 to 36.43/-117.77, 36.43/-117.58	Change the alt of 11 from 500-35500 to 17000-35500	10
5	ACM-J-17 (SSMS) ACM-J-18 (SSMS)	Move 17 from 37.71/-117.24, 36.78/-117.63 to 37.71/-117.24, 36.85/36.85/-117.73	Resolved by 17/19 (Step 1)	
6	ACM-J-4 (AAR) ACM-J-21 (SSMS)	Move 4 from 35.58/-115.67, 36.11/-115.65 to 36.65/-115.43, 35.93/-115.57	Move 4 from 35.58/-115.67, 36.11/-115.65 to 35.74/-115.31, 36.27/-115.59	5
7	ACM-J-1 (AEW) ACM-J-18 (SSMS)	Move 1 from 37.34/-116.98, 36.86/-116.58 to 37.39/-116.89, 36.86/-116.58	Move 1 from 37.34/-116.98, 36.86/-116.58 to 37.38/-116.88, 36.9/-116.48	8
8	ACM-J-3 (ABC) ACM-J-16 (RECCE)	Move 16 from 35.96/-116.3, 35.24/-115.91 to 35.81/-116.24, 35.24/-115.91	Move 16 from 35.96/-116.3,35.24/-115.91 to 35.95/-116.12, 35.38/-115.65	3
9	ACM-J-3 (ABC) ACM-J-15 (COZ)	Change 3 alt from 20500-24000 to 20500-23000	Change the alt of 15 from 23500-29500 to 25500-29500	9
10	Hava South ACM-J-15 (COZ)	Change 15 time from 1100-2359 to 1200-2359	Change 15 time from 1100-2359 to 1215-2359	2
11	ACM-J-8 (CAP) ACM-J-15 (COZ)	Move 8 from 35.79/-116.73, 35.95/-116.32 to 35.74/-116.72, 35.90/-116.36	Move 8 from 35.79/-116.73, 35.95/-116.32 to 35.71/-116.7, 35.87/-116.3	6
12	ACM-J-3 (ABC) ACM-J-8 (CAP)	Already resolved	Resolved by 8/15 (Step 6)	
13	Hava South ACM-J-4 (AAR)	Already resolved	Resolved by 4/21 (Step 5)	
14	ACM-J-1 (AEW) ACM-J-10 (CAP)	Already resolved	Resolved by 1/18 (Step 8)	

— Whether this action solves a conflict.
— Whether the result of this action still satisfies the original purpose of the mission. For example, changing the flying altitude of a missile may still satisfy its original purpose; however, changing the destination of a missile would dissatisfy its original purpose.
— The simplicity of the action. A conflict may be solved in different ways, and a simple solution should use as few steps as possible and affect the fewest number of conflicts.
— Proximity to the problem area.
— Suitability for operational context.
— Originality of the action, which is how creative it is. For example, one action may solve two conflicts. Most of the time, this refers to solutions that the SMEs consider to be quite clever and, perhaps, something that they did not even consider.

Each factor is graded on a 0-5 scale. The score for each step is the average of all above factors. The final score for a solution is an average of the scores for each step, which is then multiplied by 20 to normalize it in the range [0,100].

GILA's solution and the expert's solution for Scenario F are both shown in Table II. Out of the 14 conflicts to be resolved, four of them are resolved as side-effects of solving the other conflicts in both GILA's and the expert's solution. Three of these four conflicts are solved the same way by both GILA and the expert. Among nine other conflicts for which both GILA and the expert provided direct modifications, seven are conflicts for which GILA chose to change the same ACM (in conflict) and make the same type of change as the expert, although the new values are slightly different from the values chosen by the expert. There are two conflicts for which GILA chose to change a different ACM (in conflict), but make the same type of change as the expert. There is one conflict for which GILA changed a different ACM and also made a different type of change. There are four conflicts to which GILA gave the same (or very similar – the difference was 1) priority as the expert. Based on the criteria described above, this GILA solution is scored by the expert with a score of 96 out of 100, as shown in Table III.

6.2. GILA Versus 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 baseline. The hypothesis we tested is:

HYPOTHESIS 1. *GILA has achieved 100% human novice performance, measured by the trimmed mean score, which is calculated by ignoring the two highest and two lowest scores.*

To compare the performance of GILA with novices, we first recruited human volunteers from engineers at Lockheed Martin. After eliminating those who had prior experience with airspace management, we got 33 people for the test. These 33 people were randomly grouped into six groups. Each group was given a demonstration case, a practice case and a test case on which to perform. We used three test cases in six combinations for demonstration, practice and performance. The test could have been more stable if each group could have worked on more than one combination; however, given the availability of the subjects' time, this could not be implemented in our test.

We started with an introduction of the background knowledge. Each of the participants was given a written document that listed all the knowledge that 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 shown a video of the expert demonstration traces on how to deconflict airspaces. Based on their observations, 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 performance test case on which they were expected to work. The test ended with an exit questionnaire.

Table III shows the scores achieved by GILA and human novices. The score for the 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 the solution, which is evaluated by the SME based on the six factors described in Section 6.1. To avoid any experimental bias, the scoring process was blind. The solution was presented in a manner that prevented the expert from determining whether it was generated by GILA or by a human. The maximum possible score for one solution was 100. For example, the first row in Table III shows that for experiment EFG (using Scenario E for demonstration, F for practice and G for performance), the average score for human novices is 93.84, while the score of the 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 (which ignores the two highest and two lowest scores)

Table III: Solution Quality: GILA versus Human Novices and Number of Solutions Contributed by Each ILR

Scenario			Human Novices	GILA	SPLR	CBLR	DTLR
Demo	Practice	Performance					
E	F	G	93.84	95.40	75%	25%	0
E	G	F	91.97	96.00	60%	30%	10%
F	E	G	92.03	95.00	64%	36%	0
F	G	E	91.44	95.80	75%	17%	8%
G	E	F	87.40	95.40	75%	25%	0
G	F	E	86.3	95.00	75%	17%	8%
Average			90.5	95.4	70%	24%	6%
STDEV			7.05	0.41			
95% Confidence Interval			[88.05, 92.87]	[95.10, 95.76]			

is 91.24. Hypothesis 1 that "GILA has achieved 100% human novice performance (measured by trimmed mean score)" is supported with 99.98% confidence using a t-test.

Here are some general observations of how human novices performed differently from GILA in solving an airspace management problem.

(1) GILA sometimes gave 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 the human reasoning process uses a piece of common knowledge that is missing from 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 started to forget factors (e.g., which ACM, which method to change, etc.) that needed to be taken into account. GILA demonstrated a clearly higher level of information management ability in working with multiple conflicts at the same time.

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

6.3. Effect of Collaborations Among Components

To test the importance of the collaboration among various ILRs, we performed two additional sets of experiments. The first set is to run GILA with only one ILR for solving conflicts. The second set is to evaluate the influence of the 4DCLR on GILA's performance.

6.3.1. GILA Versus Single ILRs for Solving Conflicts. In this set of experiments, GILA ran with only one ILR for solving conflicts. However, in all these experiments, the DTLR was still used for providing cost information for the MRE, and the 4DCLR was used for internal simulation and safety constraint checking. The hypothesis to test here is:

Table IV: Comparison of GILA and Single ILRs for Conflict Resolution (Test Scenario: EFG)

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

HYPOTHESIS 2. *No single ILR (for generating deconflicting solutions) can perform as well as GILA.*

Table IV shows that the DTLR is able to solve 5 conflicts out of 14 total conflicts, while the CBLR is able to solve 7 of them. Though the 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 the 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. Although the DTLR is unable to solve some conflicts because of its incomplete search, it learns a cost function used by the MRE to guide the overall problem solving process. The CBLR fails to solve conflicts if its case library does not contain similar conflicts. The above results support Hypothesis 2 positively. These experiments verify that the collaboration of multiple ILRs is indeed important to solve problems with high-quality solutions.

6.3.2. Performance of the 4DCLR. The performance improvement gained by including the 4DCLR in GILA has been experimentally tested. Here, we summarize the results; for details see [Rebguns et al. 2008]. Specifically, we did an experimental investigation of the following hypothesis:

HYPOTHESIS 3. *GILA* with *the 4DCLR generates airspace-deconfliction steps that are more similar to those of the expert than GILA* without *the 4DCLR.*

Two performance metrics were applied in testing this hypothesis. The first, more general, metric used was:[2]

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

The score of GILA, with versus without the 4DCLR, was provided 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 was 1.0, corresponding to complete agreement between GILA and the expert. The lowest score, 0.0, occurred when GILA and the expert chose completely disjoint sets of airspace modifications.

[2]See [Rebguns et al. 2008] for the more specific second metric.

Across five experimental cases, the system generated the following results with the 4DCLR: $TP = 30$, $FP = 18$ and $FN = 22$. Based on this outcome, GILA's score using the first metric was 0.429 when the 4DCLR was included. The score of the system dropped to 0.375 when the 4DCLR was excluded, with the following results: $TP = 27$, $FP = 20$ and $FN = 25$. Based on the above results, it can be concluded that the 4DCLR is helpful for GILA's improved performance.

6.4. Effect of Demonstration Content On GILA's Performance

Though GILA has achieved quite a good performance score after learning from the expert's demonstration, there is a remaining question of how important the expert's demonstration is. In other words, is GILA solving the problem mainly based on its built-in domain knowledge? (Although this prior domain knowledge was minimized, its presence could affect the performance.) To answer this question, we designed the following two sets of experiments.

6.4.1. Performance Without Learning. The hypothesis being tested here is:

HYPOTHESIS 4. *GILA performs much worse without learning from the expert's demonstration.*

In this set of experiments, we have both GILA and human novices perform on two scenarios, without learning from the expert's demonstration. As shown in Figure 8, GILA's performance time increases significantly (about 10 times slower) when it has to solve the same problem without learning. This is due to the fact that GILA has to rely on brute force search to solve the problem. Also, without learning, GILA performs poorly on some of the harder subproblems. The difference in the solution quality score does not seem to be significant; however, small differences in score can mean big differences to the mission. As the SME explains, "An unacceptable altitude in one conflict only brought that conflict resolution's score down to 4.5 [of 5.0]. Put this in the mix for the entire 14 conflicts, and the overall score would change from 4.83 down to 4.81.... yet this could begin a snowball effect that negatively impacted the entire day's ATO." The above analysis of results support Hypothesis 4 positively. Additionally, an analysis of GILA's solutions shows that the improved scores are due to learning. With learning, GILA's solution is nearly identical to the expert's solution.

In addition, we have compared novices and GILA, with and without learning, on two matched settings:

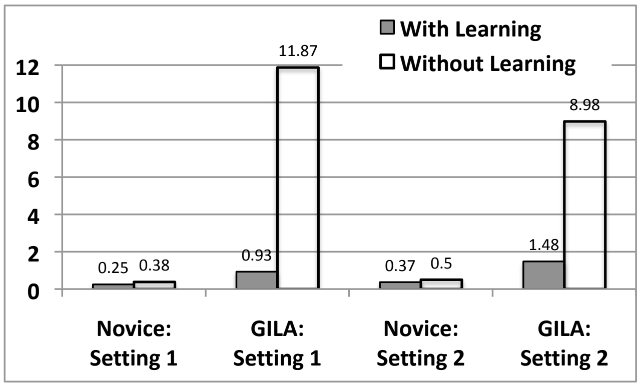
— Setting 1: perform on Scenario E without learning and on G with learning;
— Setting 2: perform on Scenario G without learning and on E with learning

As illustrated in Figure 8, when learning was not a part of the task, novices also showed a similar drop in performance in both settings. Without learning, novices took a little bit longer to solve the problem, but not as much as GILA. This is because humans often rely on common sense rather than excessive search to solve problems.

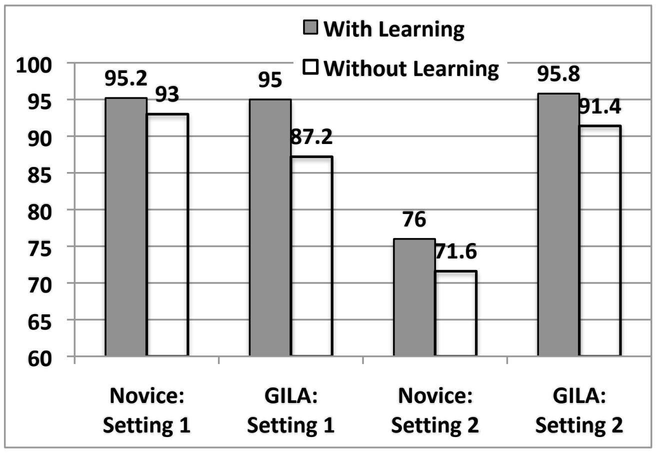
6.4.2. Learning from a Biased Expert Trace. To verify that GILA is actually learning what the expert demonstrates, we designed the following bias test. As we described in Section 3, in order to resolve a conflict, there are three types of changes an expert can make to an ACM: altitude change, geometry change and time change. In our bias test, GILA learned from three different expert traces:

(1) An expert trace that contains only altitude changes
(2) An expert trace that contains only altitude and geometry changes
(3) An expert trace that contains all three types of changes

We observed how GILA performed after learning from each of above traces. The hypothesis that was tested is:



(a) Performance Time



(b) Solution Quality Score

Fig. 8: Comparison of the impact of learning on novices and GILA

Table V: ILRs' Proposed Actions After Learning From Bias Trace

	Altitude				Altitude. Geometry				Altitude. Geometry, Time			
PSTEP	Demo Trace	CB LR	SP LR	DT LR	Demo Trace	CB LR	SP LR	DT LR	Demo Trace	CB LR	SP LR	DT LR
SetACMMinAltitude	9	24	35	12	7	6	11	7	4	3	11	5
SetACMMaxAltitude	10	29	35	8	7	7	11	4	6	5	11	4
SetACMPoint	-	-	-	54	10	8	12	-	5	3	14	-
SetStartTime	-	-	-	1	-	-	-	2	2	1	-	2
SetEndTime	-	-	-	1	-	-	-	2	1	-	-	2
SetRadius	-	-	-	14	-	-	-	-	-	-	-	-

HYPOTHESIS 5. *The content of the demonstration has substantial impact on how each ILR solves the problem.*

Table V shows how many times each ILR proposed a certain type of change after learning from a specific demonstration trace. Notice that the actions (PSTEPs) for altitude change are *SetACM-MinAltitude* and *SetACMMaxAltitude*, actions for geometry change are *SetACMPoint* and *SetRadius*, and actions for time change are *SetSatrtTime* and *SetEndTime*. Both the CBLR and the SPLR learned to strongly prefer the action choices demonstrated by the expert, and they did not generate any choice that they had not been seen in the demonstration. However, biased demonstrations led the DTLR to propose a more diverse set of changes. This was due to the DTLR's unique learning and reasoning mechanism. The DTLR always internally considers all possible changes. If there are only altitude changes in the demonstration, and they do not resolve conflicts during the DTLR's internal search, then it proposes other changes. The above results support Hypothesis 5 positively.

6.5. Effects of Practice

To study the effect of the internal practice phase, we compared GILA's performance score on six scenarios, A, B, C, E, F and G, with and without practice. The average score is 0.9156 with practice, and 0.9138 without practice. The improvement due to practice is small, which shows that GILA has not taken full advantage of practice. In fact, given the limited time working on this project, the following questions have not been addressed thoroughly in the practice phase:

— How should an ILR learn from the pseudo expert trace generated by practice? Currently, the pseudo expert trace is treated as another expert trace, and an ILR learns from it exactly as it learns from the original expert trace. However, this solution does not properly deal with the potentially inconsistent knowledge learned from these two traces.
— How should an ILR share its learned knowledge more actively with other ILRs? A pseudo expert trace actually provides more feedback to ILRs about what they have learned. By analyzing the pseudo expert trace an ILR can see, for each subproblem, whether its proposed solution has been selected. If not selected, the ILR can learn from the reason why it is not selected, and also learn from the actual selected solution.

Though the above questions have not been answered, even now practice shows good promise for improving solutions. For example, without practice, GILA moves an airspace across the FEBA (Forward Edge of the Battle Area) over into enemy territory, which is not safe. With practice, GILA finds a better way to solve the same conflict – by changing the altitude of one ACM involved in the conflict. Hence we are confident that the practice phase provides a good opportunity for GILA to exercise its learned knowledge and to improve its solution quality.

7. RELATED WORK

GILA is one of two efforts in the DARPA Integrated Learning Program to integrate multiple learning paradigms for learning a complex task from very few examples. The other effort is POIROT (Plan Order Induction by Reasoning from One Trial) [Burstein et al. 2008]. POIROT is an integrated framework for combining machine learning mechanisms to learn hierarchical models of web services procedures. Individual learners in POIROT share a common language (LTML – Learnable Task Modeling Language) in which to express their hypotheses (generalizations) and other inferences from the demonstration traces. LTML is based on ideas from OWL and PDDL. Modules can also formulate learning goals for other modules. There is a Meta-Controller that manages the learning goals following the goal-driven learning paradigm [Ram and Leake 1995]. The hypotheses generated are merged together into a single hypothesis, using a computational analogy-based method. POIROT incorporates ReSHOP, an HTN planner capable of interpreting planning domains in LTML, generating plans and executing them by invoking web service calls. While GILA's integration approach is "Independent learning with collaborative performance," POIROT's is closer to "Collaborative learning and performance." In fact, GILA's modules (ILRs) are capable of both learning and solving problems; in POIROT, on the other hand, modules only have learning capabilities. Thus, they collaborate during learning, whereas performance is accomplished by a separate module. In machine learning terms, we could see POIROT as using the different modules to explore the same generalization space using different biases, whereas in GILA, each module explores a different generalization space.

The MABLE system [Mailler et al. 2009] developed in the DARPA Bootstrapped Learning effort also uses heterogeneous learners that learn with very limited training (not just example based) and perform using their own representations and mechanisms. This system was developed later than GILA. Unlike GILA, the hierarchical breakdown of the overall learning into subproblems ("concepts") is provided to MABLE and the training associated with each concept is identified directly and the concepts are taught and tested in precedence order. MABLE faces challenges of identifying which learners ("learning strategies") are appropriate for the training "method" ("by example," "by telling," etc.) provided for a concept and extremely limited graded testing available to identify if a concept has been learned successfully.

FORR (For the Right Reason) [Epstein 1994] is another domain-independent ensemble learning architecture. This architecture assumes initial broad domain knowledge, and gradually specializes it to simulate expertise for individual problem classes. FORR contains multiple heuristic agents called "advisors" that collaborate on problem-solving decisions. A FORR-based program learns both from the performance of an external expert and from practice in its domain. This architecture has been implemented for game playing. The major difference between the FORR architecture and the GILA architecture is that FORR contains one single learner, and all advisors perform based on the same learned knowledge, whereas GILA contains multiple ILRs, and each learns using its own methods and proposes solutions based on its own internal learned knowledge. We believe that multiple diverse learning methods can be advantageous for capturing knowledge from various sources, especially when the expert demonstration examples are very few.

In addition to POIROT, MABLE and FORR, our work on GILA is related to several areas of research on the integration of learning methods (ensemble learning and multiagent learning) and on learning from demonstration. The rest of this section outlines the connections between GILA and those areas.

Ensemble learning focuses on constructing a set of classifiers and then solving new problems by combining their predictions [Dietterich 2000a]. Ensemble learning methods, such as Bagging [Breiman 1996] or Boosting [Freund and Schapire 1996], improve classification accuracy versus having an individual classifier, given that there is diversity in the ensemble. Thus, the focus on ensemble learning is to increase the classification accuracy. Moreover, except for a few exceptions, ensemble learning methods focus on creating multiple classifiers using the same learning method, but providing different training or feature sets. GILA, however, focuses on integrating different learning paradigms in order to reduce the number of training examples required to learn a complex task. Moreover, ensemble learning techniques have been studied for classification and regression tasks, whereas GILA operates on a planning task.

GILA's ILRs could be considered "agents." Multiagent learning (MAL) studies multiagent systems from a machine learning perspective [Stone and Veloso 2000]. Most recent work in MAL focuses on multiagent reinforcement learning. GILA, however, is closely related to work on distributed learning [Davies and Edwards 1995], where groups of agents collaborate to learn and solve a common problem. Work in this area focuses on both the integration of inductive inferences during learning [Davies 2001] (closely related to the POIROT project), and on the integration of solutions during problem solving [Ontañón and Plaza 2007] (which is closely related to the GILA project).

Learning from Demonstration, sometimes called "programming by demonstration" (PBD) or "imitation learning," has been widely studied in robotics [Bakker and Kuniyoshi 1996], and offers an alternative to manual programming. Lau et. al. [Lau 2001] proposed a machine learning approach for PBD based on Version Space algebra. The learning is conducted as a search in a Version Space of hypotheses, consistent with the demonstration example. Human demonstrations have also received some attention to speed up reinforcement learning [Schaal 1996], and as a way of automatically acquiring planning knowledge [Hogg et al. 2008], among others. Könik and Laird present a *Relational Learning from Observation* technique [Könik and Laird 2006] able to learn how to decompose a goal into subgoals, based on observing annotated expert traces. Könik and Laird's technique uses relational machine learning techniques to learn how to decompose goals, and the output is a collection of rules, thus showing an approach to learning planning knowledge from demonstrations. The main difference between GILA and these learning from demonstration techniques is that GILA analyzes expert demonstrations using multiple learning modules in order to learn as much knowledge as possible, and thus increase its sample efficiency.

8. CONCLUSIONS AND FUTURE WORK

In this article, we presented an ensemble architecture for learning to solve an airspace management problem. Multiple components, each using different learning/reasoning mechanisms and internal knowledge representations, learn independently from the same expert demonstration trace. A meta-reasoning executive component directs a collaborative performance process, during which it posts subproblems and selects partial solutions from the ILRs to explore. During this process, each ILR contributes to the problem-solving process without explicitly transferring its learned knowledge. This ensemble learning and problem-solving approach is efficient, as the experimental results show that GILA matches or exceeds the performance of human novices after learning from the same expert demonstration. The collaboration among various learner-reasoners is essential to success, since no single ILR can achieve the same performance as the GILA system. It has also been verified that the successful performance of GILA is primarily due to learning from an expert's demonstration rather than from knowledge engineered within the system, distinguishing GILA from a hand-engineered expert system.

The ensemble learning and problem-solving architecture developed in this work opens a new path for learning to solve complex problems from very few examples. Though this approach is tested within the domain of airspace management, it is primarily domain-independent. The collaborative performance process directed by the MRE is domain-independent, with the exception of the approach used to decompose the problem into subproblems. The learning and reasoning mechanisms inside each ILR are generally domain-independent, i.e., they can be transferred to other problem domains. Each ILR can be transferred to other domains as described below.

— The SPLR is specifically designed to be domain neutral. The policy language bias is "automatically" generated from any input domain; thus, transporting the SPLR to other domains would be a straightforward process with minimal human intervention.
— Whereas the CBLR case structure is domain-specific, the learning and reasoning components are domain-independent. Transferring the CBLR to another domain would require the identification of the most important features that would be used to represent a case in the new domain along with the representation of the set of steps used to solve a problem in the new domain. A similarity metric and adaptation rules that would operate on these features in the new domain would also be needed. The hierarchical relationships among case libraries would need to match the structure of the new domain.
— The learning algorithm of the DTLR is similarly domain-independent, whereas the features are domain-specific. To transfer to a different domain, the following three things need to be redefined: (1) a joint-feature function that gives the features defined on both input $\mathbf{x}$ and output $\mathbf{y}$ to successfully exploit the correlations between inputs and outputs, (2) a loss function that gives a discrepancy score between two outputs $\mathbf{y}$ and $\mathbf{y}$' for a given input $\mathbf{x}$, and (3) an argmax solver, which is an oracle that gives the best output $\hat{y}$ for a given input $\mathbf{x}$ according to the cost function.
— In terms of task generality, the 4DCLR is easily applicable to any physical-world application that involves physical constraints. Only the ontology and specific domain knowledge would need to

be replaced; the algorithms would remain the same. Generalization of the 4DCLR to abstract (non-physical) tasks is a topic for future investigation.

The GILA system can be extended in several ways. For example, GILA could be extended to eliminate the assumption that the expert's demonstration is perfect and that there is no disagreement among experts. Several experts may disagree on similar situations. Each ILR could be enhanced to handle this new challenge. For example, SPLR learning allows negative coverage. For priority and choice learning, the SPLR would choose to learn from the actions of the majority of experts. For margin learning, the SPLR would learn the average margins among experts. The CBLR can currently learn cases from multiple experts who agree or disagree. At performance time, a set of cases that are most similar to the current problem being solved are retrieved, and this set may contain two or more cases with radically different solutions. The CBLR will apply these solutions one at a time, and submit the solution that results in the highest quality airspace deconfliction (i.e., the lowest number of conflicts in the airspace). In the case of an imperfect expert (resulting in a learned case with an incorrect solution) the most similar case will be adapted and applied, and the resulting solution tested. In future work, in order to improve CBLR performance, a case that results in an incorrect solution would be identified, and another case would be adapted and applied in its place. We would also incorporate an introspective module that will reason about the quality of the cases learned, based on the solutions that they produce over time, and either remove lower quality cases or flag them so that they are only applied when higher quality cases are not successful. The DTLR can be improved by learning search control heuristics and an informed search algorithm that helps find higher quality solutions to its subproblems. The 4DCLR does not currently consider the situation of multiple experts who disagree, thereby resulting in inconsistent expert traces. In the future, we would like to extend the 4DCLR to address this, by weighting each expert's inputs based on his/her assessed level of expertise.

Another future direction is to introduce further collaboration among the different ILRs. How can each ILR learn from other ILRs more actively? In the work presented in this article, components are coordinated only by the meta-reasoning module at performance time. As part of our future work, we are exploring the possibility of coordinating the components at learning time by following ideas from goal-driven learning (GDL) [Ram and Leake 1995] (see [Radhakrishnan et al. 2009] for preliminary results). We can also provide feedback on each ILR's solution, including an explanation of why its solution was not selected, thereby allowing it to learn from solutions provided by other ILRs. Such collaborations would enhance the performance of the entire GILA system.

ACKNOWLEDGMENTS

Distribution Statement (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.

REFERENCES

AAMODT, A. AND PLAZA, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications 7,* 1, 39–59.

BAKIR, G. H., HOFMANN, T., SCHLKOPF, B., SMOLA, A. J., TASKAR, B., AND VISHWANATHAN, S. V. N., Eds. 2007. *Predicting Structured Data*. MIT Press, Cambridge, MA.

BAKKER, P. AND KUNIYOSHI, Y. 1996. Robot see, robot do: An overview of robot imitation. In *AISB96 Workhsop on Learning in Robots and Animals*. 3–11.

BLUM, A. AND MITCHELL, T. 1998. Combining labeled and unlabeled data with co-training. In *Annual Conference on Learning Theory(COLT)*.

BREIMAN, L. 1996. Bagging predictors. *Machine Learning 24,* 2, 123–140.

BURSTEIN, M. H., LADDAGA, R., MCDONALD, D., COX, M. T., BENYO, B., ROBERTSON, P., HUSSAIN, T. S., BRINN, M., AND MCDERMOTT, D. V. 2008. POIROT - integrated learning of web service procedures. In *AAAI*. 1274–1279.

CENDROWSKA, J. 1987. PRISM: an algorithm for inducing modular rules. *International Journal for Man-Machine Studies 27,* 4, 349–370.

CHOCKLER, H. AND HALPERN, J. Y. 2004. Responsibility and blame: A structural-model approach. *Journal of Artificial Intelligence Research (JAIR) 22*, 93–115.

CLARKE, E. M., GRUMBERG, O., AND PELED, D. 1999. *Model Checking*. MIT Press.

DAUMÉ III, H., LANGFORD, J., AND MARCU, D. 2009. Search-based structured prediction. *Machine Learning Journal 75,* 3, 297–325.

DAUMÉ III, H. AND MARCU, D. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *Proceedings of the 22nd International Conference on Machine Learning(ICML).*

DAVIES, W. AND EDWARDS, P. 1995. Distributed learning: An agent-based approach to data-mining. In *Working Notes of the ICML '95 Workshop on Agents that Learn from Other Agents*, D. Gordon, Ed. Tahoe City, CA.

DAVIES, W. H. E. 2001. The communication of inductive inference. Ph.D. thesis, University of Aberdeen.

DIETTERICH, T. 2000a. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, J. Kittler and F. Roli, Eds. Lecture Notes in Computer Science. Springer Verlag, 1 – 15.

DIETTERICH, T. G. 2000b. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning 40,* 2, 139–157.

EPSTEIN, S. L. 1994. For the right reasons: The FORR architecture for learning in a skill domain. *Cognitive Science Volume 18, Issue 3, July-September 1994, Pages 479-511 18,* 3, 479–511.

ERMAN, L. D., HAYES-ROTH, F., LESSER, V. R., AND REDDY, D. R. 1980. The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys 12,* 2, 213–253.

FREUND, Y. AND SCHAPIRE, R. E. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufmann, 148–156.

FRIEDMAN, J., HASTIE, T., AND TIBSHIRANI, R. 1998. Additive logistic regression: a statistical view of boosting. *Annals of Statistics 28*, 2000.

HOGG, C. M., MUÑOZ-AVILA, H., AND KUTER, U. 2008. HTN-MAKER: Learning htns with minimal additional knowledge engineering required. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence(AAAI)*. 950–956.

HUANG, Y., SELMAN, B., AND KAUTZ, H. 2000. Learning declarative control rules for constraint-based planning. In *Proceedings 17th International Conference on Machine Learning(ICML)*. 337–344.

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

KHARDON, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence Journal 113,* 1-2, 125–148.

KÖNIK, T. AND LAIRD, J. E. 2006. Learning goal hierarchies from structured observations and expert annotations. *Machine Learning 64,* 1-3, 263–287.

LAU, T. 2001. Programming by demonstration: a machine learning approach. Ph.D. thesis, University of Washington.

MAILLER, R., BRYCE, D., SHEN, J., AND O'REILLY, C. 2009. Mable: a framework for learning from natural instruction. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. AAMAS '09. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 393–400.

MARTIN, M. AND GEFFNER, H. 2000. Learning generalized policies in planning domains using concept languages. In *Proceedings of Seventh International Conference on Principles of Knowledge Representation and Reasoning*.

MCALLESTER, D. AND GIVAN, R. 1993. Taxonomic syntax for first order inference. *JACM 40,* 2, 246–283.

MICHAELIS, J. R., DING, L., AND MCGUINNESS, D. L. 2009. Towards the explanation of workflows. In *Proceedings of the IJCAI 2009 Workshop on Explanation Aware Computing (ExaCt)*.

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

MUÑOZ-AVILA, H. AND COX, M. 2007. Case-based plan adaptation: An analysis and review. *IEEE Intelligent Systems*.

ONTAÑÓN, S. AND PLAZA, E. 2007. Learning and joint deliberation through argumentation in multiagent systems. In *Proceedings of the 2007 International Conference on Autonomous Agents and Multiagent Systems*. 971–978.

PARKER, C., FERN, A., AND TADEPALLI, P. 2006. Gradient boosting for sequence alignment. In *Proceedings of the 21st National Conferenceo on Artificial Intelligence(AAAI)*. AAAI Press.

RADHAKRISHNAN, J., ONTAÑÓN, S., AND RAM, A. 2009. Goal-driven learning in the GILA integrated intelligence architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1205–1210.

RAM, A. AND LEAKE, D. 1995. *Goal-Driven Learning*. The MIT Press.

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 *Proceedings of CP/ICAPS COPLAS'08 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

REBGUNS, A., GREEN, D., SPEARS, D., LEVINE, G., AND KUTER, U. 2009. *Learning and verification of safety parameters for airspace deconfliction*. University of Maryland Technical Report CS-TR-4949/UMIACS-TR-2009-17.

RIVEST, R. L. 1987. Learning decision lists. *Machine Learning 2,* 3, 229–246.

SCHAAL, S. 1996. Learning from demonstration. In *Advances in neural information processing systems (NIPS)*. 1040–1046.

STONE, P. AND VELOSO, M. M. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots 8,* 3, 345–383.

WANG, W. AND ZHOU, Z. 2007. Analyzing co-training style algorithms. In *Proceedings of European Conference on Machine Learning(ECML)*.

WEISS, G., Ed. 2000. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press.

WETTSCHERECK, D., AHA, D. W., AND MOHRI, T. 1997. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review 11*, 273–314.

XU, Y., FERN, A., AND YOON, S. 2007. Discriminative learning of beam-search heuristics for planning. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI)*.

YOON, S., FERN, A., AND GIVAN, R. 2002. Inductive policy selection for first-order MDPs. In *Proceedings of Eighteenth Conference in Uncertainty in Artificial Intelligence(UAI)*.

YOON, S. AND KAMBHAMPATI, S. 2007. Hierarchical strategy learning with hybrid representations. In *Proceedings of AAAI 2007 Workshop on Acquiring Planning Knowledge via Demonstration*.

ZHANG, X., YOON, S., DIBONA, P., APPLING, D., DING, L., DOPPA, J., GREEN, D., GUO, J., KUTER, U., LEVINE, G., MACTAVISH, R., MCFARLANE, D., MICHAELIS, J., MOSTAFA, H., ONTAÑÓN, S., PARKER, C., RADHAKRISHNAN, J., REBGUNS, A., SHRESTHA, B., SONG, Z., TREWHITT, E., ZAFAR, H., ZHANG, C., CORKILL, D., DEJONG, G., DIETTERICH, T., KAMBHAMPATI, S., LESSER, V., AND ET AL. 2009. An ensemble learning and problem-solving architecture for airspace management. In *Proceedings of Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-09)*. Pasadena, CA, 203–210.

Received March 2011; revised July 2011; accepted November 2011