

MIT Open Access Articles

*Human-machine collaborative
optimization via apprenticeship scheduling*

The MIT Faculty has made this article openly available. **Please share**
how this access benefits you. Your story matters.

Citation: Gombolay, Matthew, et al., "Human-machine collaborative optimization via apprenticeship scheduling." Journal of Artificial Intelligence Research 63 (Sept. 2018): p. 1-49 doi 10.1613/JAIR.1.11233 ©2018 Author(s)

As Published: 10.1613/JAIR.1.11233

Publisher: AI Access Foundation

Persistent URL: <https://hdl.handle.net/1721.1/125878>

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike



Human-Machine Collaborative Optimization via Apprenticeship Scheduling

Matthew Gombolay

GOMBOLAY@MIT.EDU

*Massachusetts Institute of Technology,
77 Massachusetts Avenue,
Cambridge, MA 02114 USA*

Reed Jensen

RJENSEN@LL.MIT.EDU

Jessica Stigile

JESSICA.STIGILE@LL.MIT.EDU

*MIT Lincoln Laboratory,
244 Wood Street,
Lexington, MA 02420 USA*

Toni Golen

TGOLEN@BIDMC.HARVARD.EDU

Neel Shah

NTSHAH@BIDMC.HARVARD.EDU

*Beth Israel Deaconess Medical Center,
330 Brookline Avenue,
Boston, MA 02215 USA*

Sung-Hyun Son

SSON@LL.MIT.EDU

*MIT Lincoln Laboratory,
244 Wood Street,
Lexington, MA 02420 USA*

Julie Shah

JULIE_A_SHAH@CSAIL.MIT.EDU

*Massachusetts Institute of Technology,
77 Massachusetts Avenue,
Cambridge, MA 02114 USA*

Abstract

Coordinating agents to complete a set of tasks with intercoupled temporal and resource constraints is computationally challenging, yet human domain experts can solve these difficult scheduling problems using paradigms learned through years of apprenticeship. A process for manually codifying this domain knowledge within a computational framework is necessary to scale beyond the “single-expert, single-trainee” apprenticeship model. However, human domain experts often have difficulty describing their decision-making processes, causing the codification of this knowledge to become laborious. We propose a new approach for capturing domain-expert heuristics through a pairwise ranking formulation. Our approach is model-free and does not require enumerating or iterating through a large state space. We empirically demonstrate that this approach accurately learns multifaceted heuristics on a synthetic data set incorporating job-shop scheduling and vehicle routing problems, as well as on two real-world data sets consisting of demonstrations of experts solving a weapon-to-target assignment problem and a hospital resource allocation problem. We also demonstrate that policies learned from human scheduling demonstration via apprenticeship learning can substantially improve the efficiency

of a branch-and-bound search for an optimal schedule. We employ this human-machine collaborative optimization technique on a variant of the weapon-to-target assignment problem. We demonstrate that this technique generates solutions substantially superior to those produced by human domain experts at a rate up to 9.5 times faster than an optimization approach and can be applied to optimally solve problems twice as complex as those solved by a human demonstrator.

1. Introduction

Resource scheduling and optimization is a costly, challenging problem that affects almost every aspect of our lives. In healthcare, for example, patients with non-urgent needs who experience prolonged wait times have higher rates of treatment noncompliance and missed appointments (Kehle, Greer, Rutks, & Wilt, 2011; Pizer & Prentice, 2011). In military engagements, the weapon-to-target assignment problem requires warfighters to deploy the minimal amount of resources in order to mitigate as many threats as possible while maximizing the duration of survival (Lee, Su, & Lee, 2003).

The problem of optimal task allocation and sequencing with upper- and lowerbound temporal constraints (i.e., deadlines and wait constraints) is NP-Hard (Bertsimas & Weismantel, 2005), and domain-independent approaches to real-world scheduling problems quickly become computationally intractable (Boese, Kahng, & Muddu, 1994; Streeter & Smith, 2006; Do & Kambhampati, 2003). However, human domain experts are able to learn from experience to develop strategies, heuristics and rules-of-thumb to effectively respond to these problems. The challenge we pose is to autonomously learn the strategies employed by these domain experts; this knowledge can be applied and disseminated more efficiently with such a model than with a “single-expert, single-apprentice” model.

Researchers have made significant progress toward capturing domain-expert knowledge from demonstration (Berry, Gervasio, Peintner, & Yorke-Smith, 2011; Abbeel & Ng, 2004; Konidaris, Osentoski, & Thomas, 2011b; Zheng, Liu, & Ni, 2015; Odom & Natarajan, 2015; Vogel, Ramach, Gupta, & Raux, 2012; Ziebart, Maas, Bagnell, & Dey, 2008). In one recent work (Berry et al., 2011), an AI scheduling assistant called PTIME learned how users preferred to schedule events. PTIME was subsequently able to propose scheduling changes when new events occurred by solving an integer program. Two limitations to this work exist, however: PTIME requires users to explicitly rank their preferences about scheduling options to initialize the system, and also uses a complete solver that, in the worst-case scenario, must consider an exponential number of options.

Research focused on capturing domain knowledge based solely on user demonstration has led to the development of inverse reinforcement learning (IRL) (Abbeel & Ng, 2004; Konidaris et al., 2011b; Zheng et al., 2015; Odom & Natarajan, 2015; Vogel et al., 2012; Ziebart et al., 2008). IRL serves the dual purpose of learning an unknown reward function for a given problem and learning a policy to optimize that reward function.

However, there are two primary drawbacks to IRL for scheduling problems, computational tractability and the need for an environment model. The classical apprenticeship learning algorithm, developed by Abbeel and Ng in 2004, requires repeated solving of a Markov decision process (MDP) until a convergence criterion is satisfied. However, enumerating a large state space, such as those common to large-scale scheduling problems involving hundreds of tasks and tens of agents, can quickly become computationally intractable due to memory limitations. Approximate dynamic programming approaches exist that essentially reformulate the problem as regression (Konidaris

et al., 2011b; Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, et al., 2015), but the amount of data required to regress over a large state space remains challenging, and MDP-based scheduling solutions exist only for simple problems (Wu, Xu, Zhang, & Liu, 2011; Wang & Usher, 2005; Zhang & Dietterich, 1995).

IRL also requires a model of the environment for training. At its most basic, reinforcement learning uses a Markovian transition matrix that describes the probability of transitioning from an initial state to a subsequent state when taking a given action. In order to address circumstances in which environmental dynamics are unknown or difficult to model within the constraints of a transition, researchers have developed Q-Learning and its variants, which have had much recent success (Mnih et al., 2015). However, these approaches require the ability to “practice,” or explore the state space by querying a black-box emulator to solicit information about how taking a given action in a specific state will change that state.

Another prior method involves directly learning a function that maps states to actions (Chernova & Veloso, 2007; Terrell & Mutlu, 2012; Huang & Mutlu, 2014). For example, Ramanujam and Balakrishnan trained a discrete-choice model using real data collected from air traffic controllers, and showed how this model can accurately predict the correct runway configuration for an airport (Ramanujam & Balakrishnan, 2011). Sammut et al. (Sammut, Hurst, Kedzier, & Michie, 1992) applied a decision tree model for an autopilot to learn to control an aircraft from expert demonstration. Action-driven learning techniques offer great promise for learning policies from expert demonstrators, but they have not been applied to complex scheduling problems. However, in order for these methods to succeed, the scheduling problem must be modeled in a way that allows for efficient computation of a scheduling policy.

In this paper, we propose a technique, which we call “apprenticeship scheduling,” to capture this domain knowledge in the form of a scheduling policy. Our objective is to learn scheduling policies through expert demonstration and validate that schedules produced by these policies are of comparable quality to those generated by human or synthetic experts. Our approach efficiently utilizes domain-expert demonstrations without the need to train with an environment emulator. Rather than explicitly modeling a reward function and relying upon dynamic programming or constraint solvers – which become computationally intractable for large-scale problems of interest – our objective is to use action-driven learning to extract the strategies of domain experts in order to efficiently schedule tasks.

The key to our approach is the use of pairwise comparisons between the actions taken (e.g., schedule agent a to complete task τ_i at time t) and the set of actions not taken (e.g., unscheduled tasks at time t) to learn the relevant model parameters and scheduling policies demonstrated by the training examples. Our approach was inspired by cognitive studies of human decision-making, in which learning through comparisons – and, in particular, paired comparisons – was identified as a foundation of human multi-criteria decision-making (Saaty, 2008; Lombrozo, 2006). Rather than explicitly query human experts about their preferences, our approach functions more like a human apprentice who learns by observing a sequence of actions performed by a demonstrator. Our approach automatically computes pairwise comparisons of the features describing the action taken at each moment in time relative to the corresponding set of actions not taken, producing sets of both positive and negative training examples. We formulate the apprenticeship scheduling problem as one of learning a pairwise preference model, and construct a classifier that is able to predict the rank of all possible actions and, in turn, predict which action the expert would ultimately take at each moment in time.

We validated our approach using both a synthetic data set of solutions for a variety of scheduling problems and two real-world data sets of demonstrations by human experts solving a variant of the weapon-to-target assignment problem (Lee et al., 2003), known as anti-ship missile defense (ASMD), and a hospital resource allocation problem (Gombolay, Yang, Hayes, Seo, Liu, Wadhwan, Yu, Shah, Golen, & Shah, 2016). The synthetic and real-world problem domains we used to empirically validate our approach represent two of the most challenging classes within the taxonomy established by Korsah, Stentz, and Dias (2013).

The first problem we considered was the vehicle routing problem with time windows, temporal dependencies and resource constraints (VRPTW-TDR). Depending upon parameter selection, this family of problems encompasses the traveling salesman (Type 1), job-shop scheduling, multi-vehicle routing and multi-robot task allocation problems, among others. We found that apprenticeship scheduling accurately learns multifaceted heuristics that emulate the demonstrations of experts solving these problems. We observed that an apprenticeship scheduler trained on a small data set of 15 scheduling demonstrations selected the correct scheduling action with up to 95% accuracy. We also empirically characterized the extent to which our method is robust to errors that humans – even experts – may commonly make. We found that our method is able to learn a high-quality representation of the demonstrator’s underlying heuristic from a “noisy” expert demonstrator that selects an incorrect action up to 20% of the time.

Next, we observed that apprenticeship scheduling learned a policy for ASMD that outperformed the average ASMD domain expert for a statistically significant portion of problem scenarios ($p < 0.05$) when trained on 15 perfect expert-generated schedules. Third, we trained a decision support tool to assist nurses in managing resources – including patient rooms, staff and equipment – in a Boston hospital. We found that 90% of the high-quality recommendations generated by the apprentice scheduler were accepted by the nurses and doctors participating in the study.

In this work, we also introduce a new technique called Collaborative Optimization via Apprenticeship Scheduling (COVAS), which incorporates learning from human expert demonstration within an optimization framework to automatically and efficiently produce optimal solutions for challenging real-world scheduling problems. This technique applies apprenticeship scheduling to generate a favorable (if suboptimal) initial solution to a new scheduling problem. To guarantee that the generated schedule is serviceable, we augment the apprenticeship scheduler to solve a constraint satisfaction problem, ensuring that the execution of each scheduling commitment does not directly result in infeasibility for the new problem. COVAS uses this initial solution to provide a tight bound on the value of the optimal solution, substantially improving the efficiency of a branch-and-bound search for an optimal schedule.

We first presented the apprenticeship scheduling technique in a prior work (Gombolay, Jensen, Stigile, Son, & Shah, 2016), and also previously discussed an application of the technique to the hospital scheduling problem (Gombolay et al., 2016). This paper incorporates multiple extensions to these original works: First, we improve the performance of the original technique through the use of hyperparameter tuning. Second, we incorporate the data set acquired from the hospital domain in the previous study (Gombolay et al., 2016) to validate apprenticeship scheduling using a second real-world data set consisting of scheduling decisions generated by hospital nurses. Third, we present COVAS, an algorithmic extension that enables human-machine collaborative optimization. COVAS leverages apprenticeship scheduling to optimally solve scheduling problems, whereas apprenticeship scheduling alone does not provide guarantees for solution quality. We report here that COVAS is able to leverage viable (but imperfect) human demonstrations to quickly produce

globally optimal solutions. Fourth, we show that COVAS can transfer an apprenticeship scheduling policy learned for a small problem to optimally solve problems involving twice as many variables as those observed during any training demonstrations, and also produce an optimal solution an order of magnitude faster than mathematical optimization alone.

2. Background

In this section, we briefly review goal and policy learning, as well as methods for bridging machine learning (ML) and optimization. We also discuss the applicability and limitations of prior works related to learning through scheduling demonstration.

2.1 Goal Learning

Here, we review both IRL-based techniques and methods proposed for recommender and preference-learning systems within the realm of goal learning.

2.1.1 INVERSE REINFORCEMENT LEARNING

Learning from demonstration (LfD) is an active subfield of ML (Abbeel & Ng, 2004; Berry et al., 2011; Ijspeert, Nakanishi, & Schaal, 2002; Konidaris et al., 2011b; Zheng et al., 2015; Odom & Natarajan, 2015; Terrell & Mutlu, 2012; Thomaz & Breazeal, 2006; Vogel et al., 2012; Ziebart et al., 2008). Arguably, the most ubiquitous approach to LfD is inverse reinforcement learning, which is founded on a Markov decision process $M = (S, A, T, \gamma, R)$ where:

- S is a set of states.
- A is a set of actions.
- $T : S \times A \times S \rightarrow [0, 1]$ is a transition function, where $T(s, a, s')$ is the probability of being in state s' after executing action a in state s .
- $R : S \rightarrow \mathbb{R}$ ($S \times A \rightarrow \mathbb{R}$) is a reward function that takes the form of $R(s)$ or $R(s, a)$ depending upon whether the reward is assessed for being in a state or for taking a particular action within a state.
- $\gamma \in [0, 1)$ is the discount factor for future rewards.

In a Markov decision process, the goal is to learn a policy $\pi : S \rightarrow A$ that dictates which action to take in each state in order to maximize the infinite-horizon expected reward starting in state s . This reward is defined by a value function, $V^\pi(s)$, as shown in Equation 1:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^T \gamma^t R(s_t) | s_0 = s \right] \quad (1)$$

The value function satisfies the Bellman equation for all $s \in S$, as shown in Equation 2.

$$V^\pi(s) = R(s) + \gamma \left[\sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \right] \quad (2)$$

A policy π is an optimal policy π^* iff $\forall s \in S$ Equation 3 holds.

$$\pi(s) = \operatorname{argmax}_{a \in A} \left(\sum_{s' \in S} T(s, a, s') (R(s') + \gamma V^\pi(s')) \right) \quad (3)$$

The problem of inverse reinforcement learning (IRL) is to take as input 1) a Markov decision process without a known reward function R and 2) a set of m expert demonstrations $O = \{(s_o, a_o), (s_1, a_1), \dots, (s_m, a_m)\}$, and to then determine a reward function R that produces the expert demonstrations. IRL has previously been successfully applied to autonomous driving (Abbeel & Ng, 2004), aerobatic helicopter flight (Abbeel, Coates, Quigley, & Ng, 2007), urban navigation (Ziebart et al., 2008), spoken dialog systems (Chandramohan, Geist, Lefevre, & Pietquin, 2011), and more. Researchers have also extended the capability of IRL algorithms to enable learning from operators with differing skill levels (Ramachandran & Amir, 2007) and identification of operator subgoals (Michini & How, 2012).

The computational bottleneck of IRL and dynamic programming, in general, is the size of the state space. Algorithms that solve the IRL problem (Lagoudakis & Parr, 2003; Sutton, McAllester, Singh, Mansour, et al., 1999; Tesauro, 1995; Watkins & Dayan, 1992) typically work by iteratively updating the estimate of the future expected reward of each state until convergence. However, for many problems of interest, the number of states is too numerous to hold in the memory of modern computers, and the time required for the expected future reward to converge can be impractical (Wu et al., 2011; Wang & Usher, 2005; Zhang & Dietterich, 1995).

Even if one approximately solves the RL problem (Konidaris et al., 2011b; Sutton et al., 1999), RL is still ill-suited for handling the temporal dependencies among tasks inherent in scheduling problems. Some researchers have attempted to extend the traditional Markov decision process to characterize temporal phenomena, but these techniques do not scale efficiently (Bradtke & Duff, 1994; Das, Gosavi, Mahadevan, & Marchalleck., 1999; Yu, 2010). The inherent challenge is that complex real-world scheduling problems are highly non-Markovian: the next state of the environment is dependent upon the history of actions taken to arrive at the current state and time. The few works that have addressed scheduling problems via RL assume models that are too restrictive: tasks must be periodic, occur with a regular frequency, and be independent, meaning there are no temporal dependencies between the tasks (Zhang & Dietterich, 1995; Wu et al., 2011). Even work (Aydin & Öztemel, 2000) that relaxes the assumption of determinism and allows for tasks comprising pre-defined subtasks linked through precedence (as opposed to tasks representing atomic units of work) still does not consider wait-, deadline-, or resource-based constraints, nor does it consider problems in the **XD** complexity class (Korsah et al., 2013).

2.1.2 RECOMMENDER/PREFERENCE-LEARNING SYSTEMS

While not typically considered LfD, recommender systems are important within the field of goal learning. Recommender systems – those that use collected information to predict a rating or degree of preference a consumer would give for an item (e.g., goods or services) – have become ubiquitous during the Internet age, including services such as Netflix, which predicts which movies a viewer would want to watch (Koren, Bell, & Volinsky, 2009). These systems generally fall into one of two categories: collaborative filtering (CF) or content-based filtering (CB) (Park, Kim, Choi, & Kim, 2012). In essence, collaborative filtering is a technique through which an algorithm learns to predict content for a single user based upon his or her history and that of other users who share his

or her interests. However, CF suffers from problems related to data sparsity and scalability (Park et al., 2012). CB works by comparing content that the user has previously viewed with new content (Claypool, Gokhale, Miranda, Murnikov, Netes, & Sartin, 1999; Herlocker, Konstan, Terveen, & Riedl, 2004; Sarwar, Karypis, Konstan, & Riedl, 2000). The challenge of content-based filtering lies in the difficulty of measuring the similarities between two items; also, these systems can often over-fit, only predicting content that is very similar to that which the user has previously used (Basu, Hirsh, & Cohen, 1998; Schafer, Frankowski, Herlocker, & Sen, 2007). Researchers have previously employed association rules (Cho, Kim, & Kim, 2002), clustering (Lihua, Lu, Jing, & Zongyong, 2005; Linoff & Berry, 2004), decision trees (Kim, Cho, Kim, Kim, & Suh, 2002), k-nearest neighbor algorithms (Kim, Kim, & Ryu, 2009), neural networks (Anders & Korn, 1999; Ibnkahla, 2000), link analysis (Cai, He, Wen, & Ma, 2004), regression (Malhotra, 2010), and general heuristic techniques (Park et al., 2012) to recommend content to users.

Ranking the relevance of Web pages is a key focus within systems that recommend suggested topics to users (Cao, Qin, Liu, Tsai, & Li, 2007; Haveliwala, 2002; Herbrich, Graepel, & Obermayer, 2000; Jin, Valizadegan, & Li, 2008; Page, Brin, Motwani, & Winograd, 1999; Pahikkala, Tsivtsivadze, Airola, Boberg, & Salakoski, 2007; Li, Wu, & Burges, 2007; Valizadegan, Jin, Zhang, & Mao, 2009; Volkovs & Zemel, 2009). The seminal paper on Web page ranking by Page et al. initiated the computational study of page ranking with an algorithm, PageRank, which assesses the relevance of a page by determining the number of other pages that link to the page in question (Page et al., 1999). Since that paper, many have focused on developing better models for recommending Web pages to users; these models can then be trained using various ML algorithms (Haveliwala, 2002; Herbrich et al., 2000; Jin et al., 2008; Pahikkala et al., 2007).

There are three primary approaches to modeling the importance of a Web page: pointwise, pairwise, and listwise ranking. In pointwise ranking, the goal is to determine a score for a Web page via regression analysis, given features describing its contents (Li et al., 2007; Page et al., 1999). Pairwise ranking is typically a classification problem in which the aim is to predict whether one page is more relevant than another, given a user's query (Jin et al., 2008; Pahikkala et al., 2007). More recent efforts have focused on listwise ranking, in which researchers develop loss-functions based on entire lists of ranked Web pages, rather than individual pages or pairwise comparisons between pages (Cao et al., 2007; Valizadegan et al., 2009; Volkovs & Zemel, 2009). Our approach draws inspiration from the Web page pairwise ranking formulation in order to improve the tractability of learning scheduling policies from demonstration. We further discuss the relationship between prior work and our own approach in Section 3.2.

The recommender and preference-learning system most closely related to ours is that of Berry et al. (Berry, Peintner, Conley, Gervasio, Uribe, & Yorke-Smith, 2006; Berry et al., 2011), which focused specifically on scheduling applications. Their goal was to develop an autonomous scheduling assistant that learned the preferences of the user. Berry et al. produced a number of works over the course of a decade, culminating in the development of an automated scheduling assistant, called PTIME. The purpose of PTIME was to help human coworkers schedule meetings. Berry et al. incorporated extensive questionnaires to solicit the preferences of human workers regarding how they preferred to arrange their schedules. PTIME would take these preferences as input and map them to a mathematical objective function. When a new meeting needed to be arranged amongst the workers, PTIME would solve a mixed-integer mathematical program to determine the optimal time for this meeting to occur. However, after approximately a decade of work, the ultimate acceptance

rate of PTIME’s suggestions was only 60%. These authors conducted a retrospective analysis of their work and presented the following guidance for future researchers (Berry et al., 2011):

1. “A personal assistant must build trust.”
2. “An assistive agent must aim to support, rather than replace, the user’s natural process.”

These tenants have served as an inspiration for our own work, and we believe all future works should begin with these key design principles.

Other works have outlined alternate approaches to elicitation and utilization of user preferences. De Grano et al. presented a method for optimizing scheduling shifts among nurses by soliciting nurses’ preferences via an auction process (Grano, Medeiros, & Eitel, 2009). In particular, De Grano et al. used an iterative approach in which nurses first bid on which shifts they would prefer; then, their algorithm matches nurses to shifts based on their collective bids. Next, the nurses view the results and adjust their bids to push the algorithm toward a more preferable result. This process repeats over a number of iterations. The need for this iterative approach is due to the fact that nurses’ preferences were not independent: each nurse’s preferences would change according to the preferences of others. Further, it was not feasible for De Grano et al. to codify a rule set or learn a policy for each nurse (Grano et al., 2009).

Boutilier et al. and others (Boutilier, Brafman, Domshlak, Hoos, & Pool, 2004; Boutilier, Brafman, Hoos, & Poole, 1999; Öztürk, Tsouki, & Vincke, 2005) alternatively focused on modeling preferences as a set of *ceteris paribus* (all other things being equal) preference statements. In these works, researchers solicited preferences from users, typically in the form of binary comparisons. For example, consider the problem of determining which food and drink to serve a guest (Boutilier et al., 2004). In this scenario, one may already know the following:

- The guest prefers to drink red over white wine when eating a steak.
- The guest prefers steak over chicken.
- The guest prefers to drink white wine when eating chicken.

Determining the optimal food/drink pairing can be performed in polynomial-time; however, identifying the relative optimality two pairings is NP-complete (Boutilier et al., 2004).

Other researchers have focused on developing techniques for efficiently incorporating preferences into constraint satisfaction problems (Dubois & Fortemps, 1999; Lin, Xie, Guo, & Wang, 2005; Rossi, Venable, & Walsh, 2009; Rudová & Murray, 2002; Schiex, Fargier, Verfaillie, et al., 1995; Soomer & Franx, 2008). A subset of this work has specifically addressed the unique challenges of solving such formulations for scheduling problems (Benton, Coles, & Coles, 2012; Khatib, Morris, Morris, & Rossi, 2001; Minton, Johnston, Philips, & Laird, 1992; Morris, Morris, Khatib, Ramakrishnan, & Bachmann, 2004; Peintner & Pollack, 2004; Yorke-Smith, Venable, & Rossi, 2003; Rossi, Venable, & Yorke-Smith, 2006).

These methods, which are designed for scheduling problems, still suffer from issues with computational tractability. As mentioned previously, Berry et al. used a preference learning algorithm to codify an objective function, which could then be solved via mathematical optimization (Berry et al., 2006). Similarly, Wilcox et al. used mathematical programming to maximize the incorporation of users’ scheduling preferences into the system (Wilcox & Shah, 2012). However, mathematical programming is not a tractable solution technique for many real-world scheduling problems

(Bertsimas & Weismantel, 2005), including the anti-ship missile defense and hospital resource allocation problems presented in this work. Solving these problems typically requires specification of domain-specific heuristics in order to focus the search space. In this work, we present a system designed to automatically learn a heuristic policy from expert demonstration, and then apply the heuristic in order to intelligently explore the search space, reducing computation time.

2.2 Policy Learning

One alternative approach to goal learning is policy learning, which focuses on learning a mapping from states to actions (Chernova & Veloso, 2007; Huang & Mutlu, 2014; Sammut et al., 1992; Ramanujam & Balakrishnan, 2011). This technique has been applied to learn cognitive decision-making tasks from human experts (Ramanujam & Balakrishnan, 2011; Sammut et al., 1992; Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, et al., 2016; Inamura, Inaba, & Inoue, 1999; Rybski & Voyles, 1999), including an air traffic control task (Ramanujam & Balakrishnan, 2011) and a piloting task (Sammut et al., 1992).

Ramanujam and Balakrishnan investigated learning a discrete-choice model for how air traffic controllers decide which runways to use for arriving and departing aircraft according to weather, arrival and departure demand, and other environmental factors. The authors trained a discrete-choice model on real data from air traffic controllers and showed how the model was able to accurately predict the correct runway configuration for the airport (Ramanujam & Balakrishnan, 2011).

Sammut et al. applied a decision tree model to train an airplane’s autopilot from expert demonstration. Their approach generates a separate decision tree for each of the following control inputs: elevators, ailerons, flaps, and thrust. In their investigation, Sammut et al. noted that each pilot demonstrator could execute a planned flight path differently. These demonstrations could be in disagreement, thus making the learning problem significantly more difficult. To cope with the variance between pilot executions, the system learned a separate model for each pilot (Sammut et al., 1992).

Other systems learn policies through interaction and feedback, as well as demonstration, from the user (Baranes & Oudeyer, 2013; Bullard, Akgun, Chernova, & Thomaz, 2016; Chernova & Veloso, 2008; Grollman & Jenkins, 2008; Inamura et al., 1999; Konidaris, Kuindersma, Grupen, & Barto, 2011a; Zeng & Kuipers, 2016). For example, Chernova and Veloso developed a Gaussian mixture model able to interactively learn from demonstration (Chernova & Veloso, 2007). Their algorithm first learns a reasonable policy for a given task (e.g., driving a car along a highway), then solicits user feedback by constructing scenarios involving a high level of uncertainty. Support vector machines are then applied to learn when an autonomous agent should request additional demonstrations (Chernova & Veloso, 2008).

Policy learning is an important complement to goal- or reward-learning. While goal- and reward-learning approaches are able to capture high-level goals in order to produce quality schedules (Abbeel & Ng, 2004; Berry et al., 2006), these methods are limited by their reliance on computational methods for exploring the search space to identify a high-quality schedule. IRL relies on dynamic programming, which requires state space enumeration, while approaches such as PTIME (Berry et al., 2006) rely upon mathematical programming. Policy learning, on the other hand, is well-suited to guiding exploration of a state space. With a function mapping states to actions, a system can construct a schedule by taking sequential scheduling actions (e.g., assigning a worker to a task at the present time). In this sense, a learned policy can serve as a type of domain-specific

heuristic to intelligently guide a search within a large state space. However, we are unaware of any prior attempts to apply policy learning to the scheduling domain.

2.3 Blending Machine Learning and Optimization

Typically, reward and policy learning are limited by the quality of the relevant demonstrations. However, even if the demonstrations are high-quality, one cannot assume demonstrators nor their demonstrations will be optimal – or even uniformly suboptimal (Aleotti & Caselli, 2006; Sammut et al., 1992). As such, some have sought to directly model the sub-optimality of demonstrations. For example, Zheng et al. cleverly extended the work of Ramachandran and Amir (Ramachandran & Amir, 2007) to model the trustworthiness of the demonstrator within a softmax formulation transition function for reinforcement learning (Zheng et al., 2015), as shown in Equation 4. In this equation, $Q^{\pi^*(R)}(s, a)$ is the expected reward for taking action a in state s , assuming reward function R with the associated optimal policy π^* :

$$Pr((s, a)|\alpha; \mathbf{R}) = \frac{e^{\alpha Q^{\pi^*(R)}(s, a)}}{\sum_{a'} e^{\alpha Q^{\pi^*(R)}(s, a')}} \quad (4)$$

Through such a mechanism, it is possible to learn a policy that outperforms human demonstrators by inferring the intended goal rather than the demonstrated goal. Zheng et al. showed that their approach was better able to capture the ground-truth objective function from imperfect training data than Bayesian IRL (Ramachandran & Amir, 2007), which does not include a trustworthiness parameter for demonstrations. They validated their approach using a synthetic data set in an experiment with the goal of identifying the best route through an urban domain. However, one limiting assumption from their work is that a system is able to accurately measure the trustworthiness of the demonstrations – especially the relative trustworthiness amongst the demonstrations.

AlphaGo is another well-known ML-optimization framework recently developed to play Go, a turn-based strategy game (Silver et al., 2016). At its core, AlphaGo is based on policy learning; it uses a Monte-Carlo Tree Search (MCTS) that is guided by a neural network policy trained on a data set of 30 million examples of demonstrations by human Go experts. A policy π is employed to initially explore the search tree, and two additional components are used to evaluate the quality of each branching point in the tree. The first component is a second policy, π' , which is identical to the first except that the neural network includes fewer nodes. This smaller size enables the second policy to rapidly play the Go game to completion in order to predict a winner (Silver et al., 2016).

The second component of AlphaGo is a value function trained via Q-learning. The developers rewired and duplicated the initial policy π to enable improvement through self-play. These duplicated, rewired policies $\pi_{Self-Play}$ would repeatedly play Go against one another and use a policy gradient approach, developed by Sutton et al., to iteratively improve their policies; the developers then captured a data set of 30 million moves taken by these policies (Sutton et al., 1999). They then used this data set to train a Q-learning algorithm to predict the expected value of taking a given action in a given state. Interestingly, the authors noted that these self-play policies actually performed worse than the original π trained on actual human demonstrations, but did not have a cohesive theory for why this was the case. Nonetheless, AlphaGo serves as a key example for how policy learning, coupled with optimization techniques (e.g., Q-learning and policy gradient methods) can yield performance on strategy games that is superior to that of humans.

The learning-optimization system most related to our work is that developed by Banerjee et al., who considered a scheduling problem for aircraft carrier flight deck operations. The system repeatedly solved a scheduling problem wherein the variables remained the same (i.e., variables describing which workers performed which tasks and when), but the constraints relating the variables (e.g. temporal constraints between tasks) changed (Banerjee, Ono, Roy, & Williams, 2011). Using a mixed-integer linear program (MILP) formulation, they proposed a ML-optimization pipeline in which the system performed a branch-and-bound search over the integer variables, and used the prediction of a regression algorithm trained on examples of previously solved problems to provide a provable lowerbound for the optimality of the current integer variable assignments. This approach relied upon the generation of a large database of solutions to train the regression algorithm; however, this generation requires the costly exercise of repeatedly solving a large set of MILPs, which can be intractable for large-scale scheduling problems.

3. Model for Apprenticeship Learning

In this section, we present a framework for learning, via expert demonstration, a scheduling policy that correctly determines which task to schedule as a function of task state.

3.1 Problem Domain

We intend for our apprenticeship learning model to address a variety of scheduling problem types. Korsah et al. provided a comprehensive taxonomy for classes of scheduling problems, which vary according to formulation of constraints, variables and objective or utility function (Korsah et al., 2013). Within this taxonomy, there are four classes addressing interrelated utilities and constraints: No Dependencies (ND (Liu & Shell, 2013)), In-Schedule Dependencies (ID (Brunet, Choi, & How, 2008; Gombolay & Shah, 2015; Nunes & Gini, 2015)), Cross-Schedule Dependencies (XD (Gombolay, Wilcox, & Shah, 2013)) and Complex Dependencies (CD (Jones, Dias, & Stentz, 2011)).

The Korsah et al. taxonomy also delineates between tasks requiring one agent (‘single-agent tasks’ [SA]); and tasks requiring multiple agents (‘multi-agent tasks’ [MA]). Similarly, agents that perform one task at a time are “single-task agents” (ST), while agents capable of performing multiple tasks simultaneously are “multi-task agents” (MT). Lastly, the taxonomy distinguishes between “instantaneous assignment” (IA), in which all task and schedule commitments are made immediately, and “time-extended assignment” (TA), in which current and future commitments are planned.

In this work, we demonstrate our approach for two of the most difficult classes of scheduling problems defined within this taxonomy: **XD [ST-SA-TA]** and **CD [MT-MA-TA]**. The first problem we consider is the VRPTW-TDR, which is an **XD [ST-SA-TA]**-class problem. We next consider two real-world problems within the more-difficult **CD [MT-MA-TA]** class. The second problem (first real-world domain) is a variant of the weapon-to-target assignment problem (WTA) (Lee et al., 2003), known as anti-ship missile defense (ASMD). The third problem (second real-world problem) we address is one of hospital resource allocation on a labor and delivery unit, wherein one nurse, called the “resource nurse,” is responsible for ensuring that the correct patient is in the correct type of room at the correct time, with the correct types of nurses present to care for those patients. The characteristics of the three problem domains we explore in evaluating the apprenticeship scheduling algorithm are shown in Table 1.

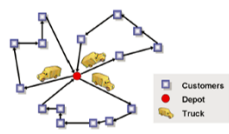


Problem Domain	 VRPTW-TDR	 ASMD	 Hospital Resource Mngmt.
Describing Section	Section 4.1	Section 4.2	Section 4.3
Data Type	Synthetic	Real-world	Real-world
Dependency Type	XD	CD	CD
Agent Type	ST	MT	MT
Task Type	SA	MA	MA
Allocation Type	TA	TA	TA

Table 1: This table summarizes the differing characteristics of the three problem domains used to empirically evaluate the apprenticeship scheduling algorithm.

3.2 Technical Approach

Many approaches to learning via demonstration, e.g., IRL, are based on Markov models (Busoniu, Babuska, & De Schutter, 2008; Barto & Mahadevan, 2003; Konidaris & Barto, 2007; Puterman, 2014). Markov models, however, do not capture the temporal dependencies between states and are computationally intractable for large problem sizes. In order to determine which tasks to schedule at which times, we draw inspiration from the domain of Web page ranking (Page et al., 1999), or predicting the most relevant Web page in response to a search query. One important component of page ranking is capturing how pages relate to one another as a graph with nodes (representing Web pages) and directed arcs (representing links between those pages) (Page et al., 1999). This connectivity is a suitable analogy for the complex temporal dependencies (precedence, wait and deadline constraints) relating tasks within a scheduling problem.

Recent approaches to page ranking have focused on pairwise and listwise models, which each have advantages over pointwise models (Valizadegan et al., 2009). In listwise ranking, the goal is to generate a ranked list of Web pages directly (Cao et al., 2007; Valizadegan et al., 2009; Volkovs & Zemel, 2009), while a pairwise approach determines ranking based on pairwise comparisons between individual pages (Jin et al., 2008; Pahikkala et al., 2007). We chose the pairwise formulation to model the problem of predicting the best task to schedule at time t .

The pairwise model has key advantages over the listwise approach: First, classification algorithms (e.g., support vector machines) can be directly applied (Cao et al., 2007). Second, a pairwise approach is non-parametric, in that the cardinality of the input vector is not dependent upon the number of tasks (or actions) that can be performed at any instance. Third, training examples of pairwise comparisons in the data can be readily solicited. From a given observation during which a task was scheduled, we only know which task was most important, not the relative importance between all tasks. Thus, we create training examples based on pairwise comparisons between scheduled and unscheduled tasks. A pairwise approach is more natural because we lack the necessary context to determine the relative rank between two unscheduled tasks.

We formulate the apprenticeship scheduling problem as one of learning a pairwise preference model, as follows. Consider a set of m observations, $O = \{O_1, O_2, \dots, O_m\}$. Each observation

$O_m = \langle \gamma, \tau_i, t_{\tau_i}, A_{\tau_i}, R_{\tau_i}, \xi_{\tau} \rangle$ is a six-tuple consisting of the following: a set of feature vectors $\gamma = \{\gamma_{\tau_1}, \gamma_{\tau_2}, \dots, \gamma_{\tau_n}\}$, where vector γ_{τ_j} describes the state of each task τ_j ; τ_i , the task to be scheduled by the expert demonstrator at the current time step t_{τ_i} ; $A_{\tau_i} \subseteq \mathbf{A}$, the subset of agents allocated to task τ_i from the set of all agents \mathbf{A} ; $R_{\tau_i} \subseteq \mathbf{R}$, the subset of resources allocated to task τ_i from the set of all resources \mathbf{R} ; and ξ_{τ} , a set of context-specific and “task-independent” features that affect expert decision-making. The state feature vector for each task γ_{τ_j} incorporates features that affect the selection of the task for execution and may represent, for example, the deadline, the earliest time at which the task is available, the duration of the task, which resource r the task requires, etc. The task-independent feature vector, ξ_{τ} , represents global state features, such as the proportion of agents that are currently idle.

An *agent* is defined as an entity that processes tasks and possesses the following set of attributes: time-varying physical location, travel speed, and task-specific proficiency (i.e., two agents may require different amounts of time to execute the same task). A *resource* is defined as an object required to process a task and possesses the following attributes: time-invariant physical location, a finite number of agents that can utilize the resource at any one time, and a task-specific proficiency (i.e., one resource may allow a task to be completed at a faster rate than another). In the event that no task is scheduled at time t , elements τ_i , A_{τ_i} , and R_{τ_i} in O_m are null.

The goal is to learn a scheduling policy that selects a task τ_i to schedule at a selected time t_{τ_i} to be processed by agent a_{τ_i} as a function of the task and problem state encoded by γ_{τ_i} and ξ_{τ} . Our formulation assumes at least one agent is required to process one task, with the assignment and scheduling of agents to tasks determined by the scheduler. The assignment of a resource to a task is assumed to be either pre-allocated based on the problem specification or assigned by the scheduler.

We assume that the cross product of the task-independent feature vectors and the task-dependent feature vector ($\xi_{\tau} \times \gamma_{\tau_1} \times \gamma_{\tau_2} \times \dots \times \gamma_{\tau_n}$) encodes sufficient information to make high quality scheduling decisions. Modeling choices may affect the dimensionalities of these feature vectors. For example, in one formulation the state of task τ_i may include a list of upper- and lowerbound temporal constraints between task τ_i and all other tasks τ_j ; alternatively, depending on the problem, a lower-dimensional representation of the same relevant information may simply include the latest possible time (i.e., the deadline) by which each task must start to satisfy the problem temporal constraints.

We note that our approach relies upon the ability of domain experts to articulate an appropriate set of features for the given problem. We believe this to be a reasonable limitation. Results from prior work have indicated that domain experts are adept at describing the high-level, contextual, and task-specific features used in their decision making; however, it is more difficult for experts to describe how they reason about these features (Cheng, Wei, & Tseng, 2006; Raghavan, Madani, & Jones, 2006). In future work, we aim to extend our approach to include feature learning rather than relying upon experts to enumerate the important features they reason about in order to construct schedules.

Our learning approach de-constructs the problem into two steps: 1) For each agent, determine the candidate next task to schedule; and 2) For each candidate task, determine whether to schedule said task.

3.2.1 LEARNING TASK PRIORITIES

In order to learn to correctly assign the next task to schedule, we transform each observation O_m into a new set of observations by performing pairwise comparisons between the scheduled task τ_i and the set of unscheduled tasks (Equations 5-6). Equation 5 creates a positive example for each observation in which a task τ_i was scheduled. This example consists of the input feature vector, $\phi_{\langle\tau_i, \tau_x\rangle}^m$, and a positive label, $y_{\langle\tau_i, \tau_x\rangle}^m = 1$. Each element of input feature vector $\phi_{\langle\tau_i, \tau_x\rangle}^m$ is computed as the difference between the corresponding values in the feature vectors γ_{τ_i} and γ_{τ_x} , describing scheduled task τ_i and unscheduled task τ_x concatenated with the high-level contextual feature vector ξ_τ . Equation 6 creates a set of negative examples with $y_{\langle\tau_x, \tau_i\rangle}^m = 0$. For the input vector, we take the difference of the feature values between unscheduled task τ_x and scheduled task τ_i concatenated with the high-level contextual feature vector ξ_τ .

We note that it is necessary to separate the task-independent features as point-wise terms so as to preserve their information. Consider the example task-independent feature, ξ_τ^k , representing the proportion of agents currently idle. If this feature would be encoded in each task-specific feature vector as $\gamma_{\tau_i}^k$, the result would be $\gamma_{\tau_i}^k - \gamma_{\tau_j}^k = 0$ for all tasks τ_i and τ_k . Thus, for their information to be preserved for the learning algorithm, one must concatenate a separate vector of contextual features to the pairwise differences.

$$rank \theta_{\langle\tau_i, \tau_j\rangle}^m := [\xi_\tau, \gamma_{\tau_i} - \gamma_{\tau_j}], y_{\langle\tau_i, \tau_j\rangle}^m = 1, \forall \tau_j \in \tau \setminus \tau_i, \forall O_m \in \mathcal{O} | \tau_i \text{ scheduled in } O_m \quad (5)$$

$$rank \theta_{\langle\tau_j, \tau_i\rangle}^m := [\xi_\tau, \gamma_{\tau_j} - \gamma_{\tau_i}], y_{\langle\tau_j, \tau_i\rangle}^m = 0, \forall \tau_j \in \tau \setminus \tau_i, \forall O_m \in \mathcal{O} | \tau_i \text{ scheduled in } O_m \quad (6)$$

$$\hat{\tau}_i^* = \operatorname{argmax}_{\tau_i \in \tau} \sum_{\tau_j \in \tau} f_{priority}(\tau_i, \tau_j) \quad (7)$$

Figure 1 is a graphical depiction of the process for automatically generating positive and negative training examples for each $O_m \in \mathcal{O}$. For illustrative purposes, the graphic depicts the process considering two task-specific features, $\gamma_{\tau_i}^k$ and $\gamma_{\tau_i}^{k'}$, corresponding to the x- and y-axes, respectively.

In the left graphic, the node “ s_t ” represents the state of the scheduling domain at time t , mapped to the feature space $(\gamma_{\tau_i}^k, \gamma_{\tau_i}^{k'})$. At this time t , the apprentice scheduler observes the demonstrator scheduling task τ_i (denoted by the solid arrow vector $\gamma_{\tau_i|t}$). The apprentice scheduler observes that the demonstrator chose to not schedule the two other available tasks τ_1 or τ_n at t (denoted by the dashed vectors $\gamma_{\tau_1|t}$ and $\gamma_{\tau_n|t}$, respectively). After the scheduling and execution of τ_i , the scheduling domain is observed to be in the state represented by node “ s_{t+1} ”. The figure shows the process repeating at time $t = 1$.

The right graphic depicts the generation of training examples. For each time step, the apprenticeship scheduler constructs positive and negative training examples through vector subtraction of task-dependent feature vectors. The red dotted lines depict the vector difference of the scheduled task’s feature vector and each unscheduled task’s feature vector; the resulting vectors are applied to construct negative training examples. The blue dotted lines depict the *negative* vector difference of the scheduled task’s feature vector and each unscheduled task’s feature vector; the resulting vectors are applied to construct positive training examples. Recall that a contextual “task-independent” feature vector, ξ_τ , is appended to each pairwise term in the formation of each training example

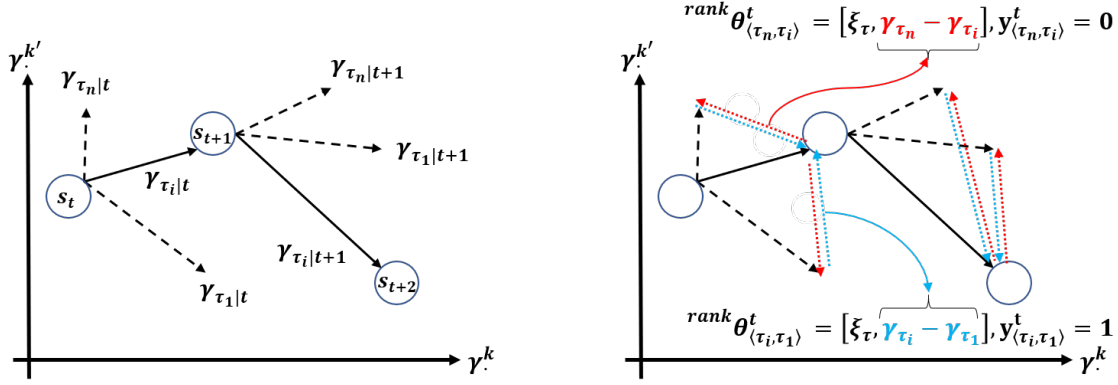


Figure 1: This figure depicts the process for automatically generating positive and negative training examples for each $O_m \in \mathcal{O}$. The left graphic shows the apprentice scheduler’s observations of the expert, and the right graphic depicts the construction of training examples through pairwise comparisons.

$\text{rank } \theta_{\langle \cdot, \cdot \rangle}^m$. This procedure then repeats for each observation (i.e., each time step for each demonstrated schedule) and task. The value of this approach is that the learner does not need to explicitly solicit pairwise comparisons from the demonstrator; instead, the pairwise comparisons are derived automatically through observation of the expert demonstrator.

3.2.2 LEARNING TO SCHEDULE OR IDLE

Given these observations O_m and their associated features, we can train a classifier, $f_{\text{priority}}(\tau_i, \tau_x) \in \{0, 1\}$, to predict whether it is better to schedule task τ_i as the next task rather than τ_x . With this pairwise classifier, we can determine which single task τ_i is the highest-priority task τ_i^* according to Equation 7 by determining which task has the highest cumulative priority in comparison to the other tasks in τ . In this work, we train a single classifier, $f_{\text{priority}}(\tau_i, \tau_j)$, to model the behavior of the set of all agents rather than train one $f_{\text{priority}}(\tau_i, \tau_j)$ for each agent. $f_{\text{priority}}(\tau_i, \tau_j)$ is a function of all features associated with the agents; as such, agents need not be interchangeable, and different sets of features may be associated with each agent.

Next, we must learn to predict whether τ_i^* should be scheduled or the agent should remain idle. To do so, we train a second classifier, $f_{\text{act}}(\tau_i) \in \{0, 1\}$, that predicts whether or not τ_i should be scheduled. The observations set, \mathcal{O} , consists either of examples in which a task was scheduled or those in which no task was scheduled. To train this classifier, we construct a new set of examples according to Equation 8, which assigns positive labels to examples from O_m in which a task was scheduled and negative labels to examples in which no task was scheduled.

$$\text{act } \phi_{\tau_i}^m := [\xi_\tau, \gamma_{\tau_i}], y_{\tau_i}^m = \begin{cases} 1 : \tau_i \text{ scheduled in } O_m \wedge \tau_i \text{ scheduled in } O_{m+1} \\ 0 : \tau_i \text{ not scheduled in } O_m \end{cases} \quad (8)$$

Finally, we construct a scheduling algorithm to act as an apprentice scheduler (Algorithm 1). This algorithm takes as input the set of tasks, τ ; agents, \mathcal{A} ; temporal constraints (i.e., upper- and lowerbound temporal constraints) relating tasks in the problem, TC ; and the set of task pairs that require the same resources and can therefore not be executed at the same time, τ_R . Lines 1- 2 iterate over each agent at each time step. (In the event that resource-to-task assignments are not

predefined, the algorithm would also iterate over each resource $r \in \mathbf{R}$ that could be assigned.) In Line 3, the highest-priority task, τ_i^* , is determined for a particular agent. In Lines 4-5, τ_i^* is scheduled if $f_{act}(\tau_i^*)$ predicts that τ_i^* should be scheduled at the current time.

Algorithm 1 Pseudocode for an Apprentice Scheduler

ApprenticeScheduler($\tau, \mathbf{A}, TC, \tau_R$)

```

1: for  $t = 0$  to  $T$  do
2:   for all agents  $a \in \mathbf{A}$  do
3:      $\tau_i^* \leftarrow \operatorname{argmax}_{\tau_i \in \tau} \sum_{\tau_j \in \tau} f_{priority}(\tau_i, \tau_j)$ 
4:     if  $f_{act}(\tau_i^*) == 1$  then
5:       Schedule  $\tau_i^*$ 
6:     end if
7:   end for
8: end for

```

Note that iteration over agents (Line 2) can be performed according to a specific ordering, or the system can alternatively learn a more general priority function to select and schedule the best agent-task-resource tuple using $f_{priority}(\langle \tau_i, a, r \rangle, \langle \tau_j, a', r' \rangle)$, $f_{act}(\langle \tau_i, a, r \rangle^*)$. In the latter case, the features γ_{τ_i} are mapped to agent-task-resource tuples rather than tasks τ_i , which represent the atomic (i.e., lowest-level) job. For the synthetic evaluation, we use the original formulation, $f_{priority}(\tau_i, \tau_j)$. For the ASMD application, we use $f_{priority}(\langle \tau_i^t, a, r \rangle, \langle \tau_j^t, a', r' \rangle)$, where τ_i^t represents the objective of mitigating missile i during time step t , a is the decoy to be deployed, and r is the physical location for that deployment. For the hospital domain evaluation, we use $f_{priority}(\langle \tau_i^j, a, r \rangle, \langle \tau_p^q, a', r' \rangle)$, where τ_i^j represents the j^{th} stage of labor for patient i , a is the assigned nurse, and r is the room to which the patient is assigned. For convenience in notation, we refer to this tuple as a “scheduling action.” Finally, note that multiple agent-resource pairs can be assigned to a single task, τ_i . The apprentice scheduler would first pick the best agent (or agent-resource pair) to assign to a task according to the $f_{priority}$ metric. During the same time step (or a subsequent time step), another agent (or agent-resource pair) can be added. The algorithm will continue to add assignments to the task until the null assignment (i.e., no further changes to the current set of assignments) is the best option according to f_{act} .

Our model is a hybrid point- and pairwise formulation, which has several key benefits for learning to schedule from expert demonstration. First, we can directly apply standard classification techniques, such as a decision tree, support vector machine, logistic regression, or neural networks. Second, because this technique only considers two scheduling actions at a time, the model is non-parametric in the number of possible actions. Thus, the system can train on $f_{priority}(\tau_i, \tau_j)$ schedules with a agents and n tasks, yet apply $f_{priority}(\tau_i, \tau_j)$ to construct a schedule for a problem with a' agents and n' tasks where $a \neq a'$, $n \neq n'$, and $a * n \neq a' * n'$. Furthermore, it can even train $f_{priority}(\tau_i, \tau_j)$ on demonstrations of a heterogeneous data set of scheduling observations with differing numbers of agents and tasks. Third, the pairwise portion of the formulation provides structure for the learning problem. A formulation that simply concatenated the features of two or more scheduling actions would need to solve the more complex problem of learning the relationships between features and then how to use those relationships to predict the highest-priority scheduling action. Such a concatenation approach would suffer from the curse of dimensionality and require a

very large training data set (Indyk & Motwani, 1998). Note, however, that this method requires the designer to appropriately partition the features into pairwise and pointwise components such that the pairwise portion does not lose information by considering the differences between actions' features. Fourth, the transformation of the observations into a pairwise model results in some features that are advantageous for learning from small data sets: the number of positive and negative training examples is balanced given that the algorithm simultaneously creates one negative label for every positive label, and the observations are bootstrapped to create $2 * |\tau|$ examples for each time step, rather than only $|\tau|$ for a pointwise model, where $n = |\tau|$.

4. Data Sets

Here, we validate that schedules produced by the learned policies are of comparable quality to those generated by human or synthetic experts. To do so, we considered a synthetic data set from the **XD [ST-SA-TA]** class of problems and two real-world data sets from the **CD [MT-MA-TA]** class of problems, as defined by Korsah et al. (Korsah et al., 2013). We present each problem domain and describe the manner in which the data set of expert demonstrations for the domain was acquired.

4.1 Synthetic Data Set

For our first investigation, we generated a synthetic data set of scheduling problems in which agents were assigned a set of tasks. The tasks were related through precedence or wait constraints, as well as deadline constraints, which could be absolute (relative to the start of the schedule) or relative to another task's initiation or completion time. Agents were required to access a set of shared resources to execute each task. Agents and tasks had defined starting locations, and task locations were static. Agents were only able to perform tasks when present at the corresponding task location, and each agent traveled at a constant speed between task locations. Task completion times were potentially non-uniform and agent-specific, as would be the case for heterogeneous agents. An agent that was incapable of performing a given task was assumed to have an infinite completion time for that task. The objective was to minimize the makespan or other time-based performance measures.

This problem definition spans a range of scheduling problems, including the traveling salesman, job-shop scheduling, multi-vehicle routing and multi-robot task allocation problems, among others. We describe this range as a vehicle routing problem with time windows, temporal dependencies, and resource constraints (VRPTW-TDR), which falls within the **XD [ST-SA-TA]** class in the taxonomy by Korsah et al. (2013): agents perform tasks sequentially (ST), each task requires one agent (SA), and commitments are made over time (TA).

To generate our synthetic data set, we developed a mock scheduling expert that applies one of a set of context-dependent rules based on the composition of the given scheduling problem. This behavior was based upon rules presented in prior work addressing these types of problems (Gombolay et al., 2013; Gombolay & Shah, 2015; Solomon, 1987; Tan, Lee, Zhu, & Ou, 2001). Our objective was to show that our apprenticeship scheduling algorithm learns both context-dependent rules and how to identify the associated context for their correct application.

The mock scheduling expert functions as follows: First, the algorithm collects all alive and enabled tasks $\tau_i \in \tau_{AE}$ as defined by (Muscettola, Morris, & Tsamardinos, 1998). Consider a pair of tasks, τ_i and τ_j , with start and finish times s_i, f_i and s_j, f_j , respectively, such that there is a wait constraint requiring τ_i to start at least $W_{\langle \tau_j, \tau_i \rangle}$ units of time after τ_j . A task τ_i is alive and enabled if $t \geq f_j + W_{\tau_j, \tau_i}$ for all such τ_j and $W_{\langle \tau_j, \tau_i \rangle}$ in τ .

After task collection, the heuristic iterates over each agent to identify the highest-priority task, τ_i^* , to schedule for that agent. The algorithm determines which scheduling rule is most appropriate to apply for each agent. If agent speed is sufficiently slow (≤ 1 m/s), travel time will become the major bottleneck. If agents move quickly but utilize one or more resources R heavily ($\sum_{\tau_i} \sum_{\tau_j} 1_{R_{\tau_i}=R_{\tau_j}} \geq c$ for some constant c), use of these resources can become the bottleneck. Otherwise, task durations and associated wait constraints are generally most important.

If the algorithm identifies travel distance as the primary bottleneck, it chooses the next task by applying a priority rule well-suited for vehicle routing that minimizes a weighted, linear combination of features (Gambardella, Éric Taillard, & Agazzi, 1999; Solomon, 1987) comprised of the distance and angle relative to the origin between agent a and τ_j . This rule is depicted in Equation 9, where \vec{l}_x is the location of τ_j , \vec{l}_a is the location of agent a , θ_{xa} is the relative angle between the vector from origin to the agent location and the origin to the location of τ_j , and α_1 and α_2 are weighting constants:

$$\tau_i^* \leftarrow \underset{\tau_j \in \tau_{AE}}{\operatorname{argmin}} \left(\|\vec{l}_x - \vec{l}_a\| + \alpha_1 \theta_{xa} + \alpha_2 \|\vec{l}_x - \vec{l}_a\| \theta_{xa} \right) \quad (9)$$

If the algorithm identifies resource contention as the most important bottleneck, it employs a rule to mitigate resource contention in multi-robot, multi-resource problems based on prior work in scheduling for multi-robot teams (Gombolay et al., 2013). Specifically, the algorithm uses Equation 10 to select the high-priority task to schedule next, where d_{τ_j} is the deadline of τ_j and α_3 is a weighting constant:

$$\tau_i^* \leftarrow \underset{\tau_j \in \tau_{AE}}{\operatorname{argmax}} \left(\left(\sum_{\tau_i} \sum_{\tau_j} 1_{R_{\tau_i}=R_{\tau_j}} \right) - \alpha_3 d_{\tau_j} \right) \quad (10)$$

If the algorithm decides that temporal requirements are the major bottleneck, it employs an Earliest Deadline First rule (Equation 11), which performs well across many scheduling domains (Chen & Askin, 2009; Gombolay et al., 2013; Gombolay & Shah, 2015):

$$\tau_i^* \leftarrow \underset{\tau_j \in \tau_{AE}}{\operatorname{argmin}} d_{\tau_j} \quad (11)$$

After selecting the most important task, τ_i^* , the algorithm determines whether the resource required for τ_i^* , $R_{\tau_i^*}$, is idle and whether the agent is able to travel to the task location by time t . If these constraints are satisfied, the heuristic schedules task τ_i^* at time t . (An agent is able to reach task τ_i^* if $t \geq f_j + k(l_i - l_j) / \|l_i - l_j\|$ for all $\tau_j \in \tau$ that the agent has already completed, where k is the agent's speed.)

We constructed the synthetic data set for two homogeneous agents and 20 partially ordered tasks located within a 20 x 20 grid.

4.2 Real-World Data Set: Anti-Ship Missile Defense

In ASMD, the goal is to protect one's naval vessel against attacks by anti-ship missiles using "soft-kill weapons" (i.e., decoys) that mimic the qualities of a target in order to direct the missile away from its intended destination.

Developing tactics for soft-kill weapon coordination is highly difficult due to the relationship between missile behavior and soft-kill weapon characteristics. The control laws governing anti-ship missiles vary, and the captain must select the correct decoy types in order to counteract the

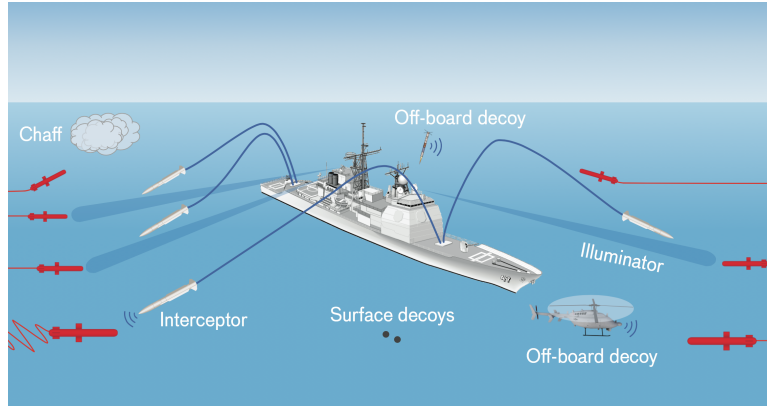


Figure 2: The tactical action officer aboard a naval vessel must coordinate a heterogeneous set of soft- and hard-kill weapons to defeat various anti-ship missiles.

associated anti-ship missiles. For example, a ship's captain may deploy a decoy that emits a large amount of heat in order to cause an enemy heat-seeking missile to fly toward the decoy rather than the ship. Also, an enemy missile may consider the spatial layout of all targets in order to select the nearest or furthest targets; in doing so, the missile may consider the magnitude of the radar reflectivity, radar emissions, and heat emissions, either separately or in various combinations.

Further, decoys have different financial costs and timing characteristics: Some decoys, such as unmanned aerial vehicles (UAVs), are able to function throughout the entirety of an engagement, while others, such as an infrared (IR) flares, disappear after a certain time. As a result, a captain may be required to use multiple decoys in tandem in order to divert a single anti-ship missile, but may also be able to use a single decoy to defeat multiple missiles. There is a complex interplay between the types and locations of decoys relative to the control laws governing anti-ship missiles. For example, deployment of a particular decoy, while effective against one airborne enemy missile, may actually cause a second enemy missile that was previously homing in on a second decoy to now impact the ship.

The ASMD problem is characterized as the most complex class of scheduling problem according to the Korsah et al. (2013). taxonomy : **CD [MT-MA-TA]**. The problem considers multi-task agents (MA) in the form of decoys, each of which can work to divert multiple missiles at the same time. The problem also incorporates multi-agent tasks (MT); a feasible solution may require the simultaneous use of multiple agents in order to complete an individual task. Further, time-extended agent allocation (TA) must be considered, given the potential future consequences of scheduling actions taken at the current moment. Finally, the ASMD problem falls within the CD class, because each task can be decomposed in a variety of ways – each with their own cost – in order to accomplish the same goal, with each decomposition affecting the value and feasibility of the decompositions of other tasks. The full specification of the mixed-integer linear program formulation for the ASMD problem is provided in Appendix A.

4.2.1 DATA COLLECTION

A real-world data set was collected, consisting of human demonstrators of various skill levels solving the anti-ship missile defense (ASMD) weapon-to-target assignment problem. Data was collected

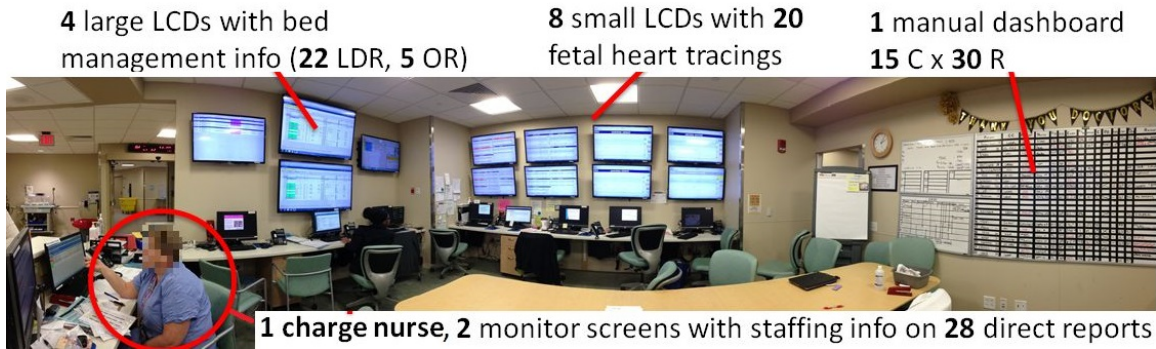


Figure 3: A resource nurse must assimilate a large variety and volume of information to effectively reason about resource management for patient care.

from domain experts playing a serious game, called Strike Group Defender¹ (SGD), for ASMD training. Game scenarios involved five types of decoys and 10 types of threats. Threats were randomly generated for each played scenario, promoting the development of strategies that were robust to a varied distribution of scenarios. Each decoy had a specified effectiveness against each threat type.

Players attempted to deploy a set of decoys by using the correct types at the correct locations and times in order to distract incoming missiles. Threats were launched over time; an effective deployment at time t could become counterproductive in the future as new enemy missiles were launched.

Games were scored as follows: 10,000 points were received each time a threat was neutralized and 2 points were received for each second a threat spent homing in on a decoy. Players lost 5,000 points for each threat impact and 1 point was deducted for each second a threat spent homing in on the player's ship. At each decoy deployment, players lost 25-1,000 points depending upon decoy type.

The collected data set consisted of 311 games played by 35 humans across 45 threat configurations, or "levels." From this set, we also separately analyzed 16 threat configurations such that each configuration included at least one human demonstration in which the ship was successfully protected from all enemy missiles. For these 16 configurations, there were 162 total games played by 27 unique human demonstrators. The player cohort consisted of technical fellows and associates, as well as contractors at a federally funded research and development center (FFDRC), with expertise varying from "generally knowledgeable about the ASMD problem" to "domain experts" with professional experience or training in ASMD.

4.3 Real-World Data Set: Labor and Delivery

To further evaluate our approach, we applied our method to a second data set collected from a labor and delivery floor at a Boston hospital. In this domain, a "resource nurse" must solve a problem of task allocation and schedule optimization with stochasticity in the number and types of patients and the duration of tasks. Specifically, the resource nurse is responsible for ensuring that the correct patient is in the correct type of room at the correct time, with the correct types of nurses present

1. SGD was developed by Pipeworks Studio in Eugene, Oregon, USA.

to care for those patients. The functions of a resource nurse are to assign nurses to take care of labor patients; assign patients to labor beds, recovery room beds, operating rooms, antepartum ward beds or postpartum ward beds; assign scrub technicians to assist with surgeries in operating rooms; call in additional nurses if necessary; accelerate, delay or cancel scheduled inductions or cesarean sections; expedite active management of a patient in labor; and reassign roles among nurses.

Using our apprenticeship scheduling method in for the Labor and Delivery problem domain, a task τ_i represents the set of steps (subtasks) required to care for patient i , and each τ_i^j is a given stage of labor for that patient. Stages of labor are related by stochastic lowerbound constraints $W_{\langle \tau_i^j, \tau_x^y \rangle}$, requiring the stages to progress sequentially. There are stochastic time constraints, $D_{\tau_i^j}^{abs}$ and $D_{\langle \tau_i^j, \tau_x^y \rangle}^{rel}$, relating the stages of labor to account for the inability of resource nurses to perfectly control when a patient will move from one stage to the next. Arrivals of τ_i (i.e. patients) are drawn from stochastic distributions. The model considers three types of patients: scheduled cesarean patients, scheduled induction patients and unscheduled patients. The set of $W_{\langle \tau_i^j, \tau_x^y \rangle}$, $D_{\tau_i^j}^{abs}$ and $D_{\langle \tau_i, \tau_j \rangle}^{rel}$ are dependent upon patient type.

Labor nurses are modeled as agents with a finite capacity to process tasks in parallel, where each subtask requires a variable amount of this capacity. For example, a labor nurse may generally care for a maximum of two patients simultaneously. If the nurse is caring for a patient who is “full and pushing” (i.e., the cervix is fully dilated and the patient is actively trying to push out the baby) or in the operating room, he or she may only care for that patient.

Rooms on the labor floor (e.g., a labor room, an operating room, etc.) are modeled as resources, which process subtasks in series. Agent and resource assignments to subtasks are pre-emptable, meaning that the agent and resource assigned to care for any patient during any step in the care process may be changed over the course of executing that subtask.

In this formulation, ${}^tA_{\tau_i^j}^a \in \{0, 1\}$ is a binary decision variable for assigning agent a to subtask τ_i^j for time epoch $[t, t + 1)$. ${}^tG_{\tau_i^j}^a$ is an integer decision variable for assigning a certain portion of the effort of agent a to subtask τ_i^j for time epoch $[t, t + 1)$. ${}^tR_{\tau_i^j}^r \in \{0, 1\}$ is a binary decision variable for whether subtask τ_i^j is assigned resource r for time epoch $[t, t + 1)$. $H_{\tau_i} \in \{0, 1\}$ is a binary decision variable for whether task τ_i and its corresponding subtasks are to be completed. $U_{\tau_i^j}$ specifies the effort required from any agent to work on τ_i^j . $s_{\tau_i^j}, f_{\tau_i^j} \in [0, \infty)$ are the start and finish times of τ_i^j .

$$\min fn \left(\{ {}^tA_{\tau_i^j}^a \}, \{ {}^tG_{\tau_i^j}^a \}, \{ {}^tR_{\tau_i^j}^r \}, \{ H_{\tau_i} \}, \{ s_{\tau_i^j}, f_{\tau_i^j} \} \right) \quad (12)$$

$$\sum_{a \in A} {}^t A_{\tau_i^j}^a \geq 1 - M(1 - H_{\tau_i}), \forall \tau_i^j \in \tau, \forall t \quad (13)$$

$$M \left(2 - {}^t A_{\tau_i^j}^a - H_{\tau_i} \right) \geq -U_{\tau_i^j} + {}^t G_{\tau_i^j}^a \geq M \left({}^t A_{\tau_i^j}^a + H_{\tau_i} - 2 \right), \forall \tau_i^j \in \tau, \forall t \quad (14)$$

$$\sum_{\tau_i^j \in \tau} {}^t G_{\tau_i^j}^a \leq C_a, \forall a \in A, \forall t \quad (15)$$

$$\sum_{r \in R} {}^t R_{\tau_i^j}^r \geq 1 - M(1 - H_{\tau_i}), \forall \tau_i^j \in \tau, \forall t \quad (16)$$

$$\sum_{\tau_i^j \in \tau} {}^t R_{\tau_i^j}^r \leq 1, \forall r \in R, \forall t \quad (17)$$

$$ub_{\tau_i^j} \geq f_{\tau_i^j} - s_{\tau_i^j} \geq lb_{\tau_i^j}, \forall \tau_i^j \in \tau \quad (18)$$

$$s_{\tau_x^y} - f_{\tau_i^j} \geq W_{\langle \tau_i, \tau_j \rangle}, \forall \tau_i, \tau_j \in \tau, \forall W_{\langle \tau_i, \tau_j \rangle} \in \mathbf{TC} \quad (19)$$

$$f_{\tau_x^y} - s_{\tau_i^j} \leq D_{\langle \tau_i, \tau_j \rangle}^{rel}, \forall \tau_i, \tau_j \in \tau | \exists D_{\langle \tau_i, \tau_j \rangle}^{rel} \in \mathbf{TC} \quad (20)$$

$$f_{\tau_i^j} \leq D_{\tau_i}^{abs}, \forall \tau_i \in \tau | \exists D_{\tau_i}^{abs} \in \mathbf{TC} \quad (21)$$

Equation 13 enforces that each subtask τ_i^j during each time epoch $[t, t + 1)$ is assigned a single agent. Equation 14 ensures that each subtask τ_i^j receives a sufficient portion of the effort of its assigned agent a during epoch $[t, t + 1)$. Equation 15 ensures that agent a is not oversubscribed. Equation 16 ensures that each subtask τ_i^j of each task τ_i that is to be completed (i.e., $H_{\tau_i} = 1$) is assigned one resource r . Equation 17 ensures that each resource r is assigned to only one subtask during each epoch $[t, t + 1)$. Equation 18 requires the duration of subtask τ_i^j to be less than or equal to $ub_{\tau_i^j}$ and at least $lb_{\tau_i^j}$ units of time. Equation 19 requires that τ_x^y occurs at least $W_{\langle \tau_i^j, \tau_x^y \rangle}$ units of time after τ_i^j . Equation 20 requires that the duration between the start of τ_i^j and the finish of τ_x^y be less than $D_{\langle \tau_i^j, \tau_x^y \rangle}^{rel}$. Equation 21 requires that τ_i^j finishes before $D_{\tau_i^j}^{abs}$ units of time have expired since the start of the schedule.

The functions of a resource nurse are to assign nurses to take care of labor patients and to assign patients to labor beds, recovery room beds, operating rooms, antepartum ward beds or postpartum ward beds. The resource nurse has substantial flexibility when assigning beds, and his or her decisions will depend upon the type of patient and the current status of the unit in question. He or she must also assign scrub technicians to assist with surgeries in operating rooms, and call in additional nurses if required. The corresponding decision variables for staff assignments and room/ward assignments in the above formulation are ${}^t A_{\tau_i^j}^a$ and ${}^t R_{\tau_i^j}^r$, respectively.

The resource nurse may accelerate, delay or cancel scheduled inductions or cesarean sections in the event that the floor is too busy. Resource nurses may also request expedited active management of a patient in labor. The decision variables for the timing of transitions between the various steps in the care process are described by $s_{\tau_i^j}$ and $f_{\tau_i^j}$. The commitments to a patient (or that patient's procedures) are represented by H_{τ_i} .

The resource nurse may also reassign roles among nurses: For example, a resource nurse may pull a nurse from triage, or even care for patients herself if the floor is too busy. Or, if a patient's

condition is particularly acute (e.g., the patient has severe preeclampsia), the resource nurse may assign one-to-one nursing. The level of attentional resources a patient requires and the level a nurse has available correspond to variables $U_{\tau_i^j}$ and ${}^tG_{\tau_i^j}^a$, respectively. The resource nurse makes his or her decisions while considering current patient status $\Lambda_{\tau_i^j}$, which is manually transcribed on a whiteboard, as shown in Figure 3.

The stochasticity of the problem arises from the uncertainty in the upper- and lowerbound of the durations ($ub_{\tau_i^j}$ and $lb_{\tau_i^j}$) of each of the steps in caring for a patient; the number and types of patients, τ ; and the temporal constraints, TC , relating the start and finish of each step. These variables are a function of the resource and staff allocation variables, ${}^tR_{\tau_i^j}^a$ and ${}^tA_{\tau_i^j}^a$, as well as patient task state $\Lambda_{\tau_i^j}$, which includes information on patient type (i.e., presentation with scheduled induction, scheduled cesarean section, or acute unplanned anomaly), gestational age, gravida, parity, membrane status, anesthesia status, cervix status, time of last exam and the presence of any comorbidities. Formally, $(\{ub_{\tau_i^j}, lb_{\tau_i^j} | \tau_i^j \in \tau\}, \tau, TC) \sim P(\{{}^tR_{\tau_i^j}^a, {}^tA_{\tau_i^j}^a, \Lambda_{\tau_i^j}, \forall t \in [0, 1, \dots, T]\})$.

The computational complexity of completely searching for a solution that satisfies the constraints in Equations 13-21 is given by $O(2^{|A||R|T^2} C_a^{|A|T})$, where $|A|$ is the number of agents, with each agent possessing an integer processing capacity of C_a . There are n tasks τ_i , each with m_i sub-tasks, $|R|$ resources, and an integer-valued planning horizon of T units of time. In practice, there are ~ 10 nurses (agents) who can care for up to two patients at a time (i.e., $C_a = 2, \forall a \in A$), 20 different rooms (resources) of varying types, 20 patients (tasks) at any one time, and a planning horizon of 12 hours or 720 minutes, yielding a worst-case complexity of $\sim 2^{10 \times 20 \times 720^2} 2^{10 \times 720} \geq 2^{10^6}$, which is computationally intractable for exact methods without the assistance of informative search heuristics.

4.3.1 DATA COLLECTION

To collect data from resource nurses about their decisions, a high-fidelity simulation of a labor and delivery floor was developed, as depicted in Figure 4. We developed this simulation in collaboration with Beth Israel Medical Deaconess Hospital in Boston. The effort was part of a quality-improvement project at the hospital to develop training tools and involved a rigorous, year-long design and iteration process that included workshops with nurses, physicians, and medical students to ensure the tool accurately captured the role of a resource nurse. Parameters within the simulation (e.g., patient arrivals, timelines for labor progression) were drawn from medical textbooks and papers and modified through alpha and beta testing to ensure that the simulation closely mirrored the patient population and nurse experience at our partner hospital.

We invited expert resource nurses to play this simulation in order to collect a data set for training our apprenticeship scheduling algorithm. This data set was generated by seven resource nurses working with the simulation for a total of $2^{1/2}$ hours, simulating 60 hours of elapsed time on a real labor floor and yielding a set of more than 3,013 individual decisions.

5. Empirical Evaluation of Apprenticeship Scheduling

In this section, we evaluate our prototype for apprenticeship scheduling using synthetic and real-world data sets.



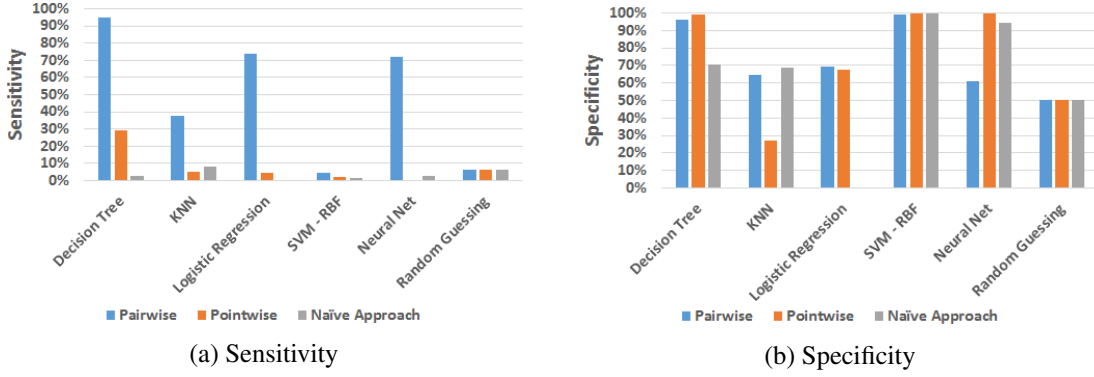


Figure 5: Figures 5a-5b depict the sensitivity and specificity of ML techniques using the pairwise, pointwise and naïve approaches.

5.1 Synthetic Data Set

We trained our model using a decision tree, KNN classifier, logistic regression (logit) model, a support vector machine with a radial basis function kernel (SVM-RBF), and a neural network to learn $f_{priority}(\cdot, \cdot)$ and $f_{act}(\cdot)$. We randomly sampled 85% of the data for training and 15% for testing.

We defined the input features as follows: The high-level feature vector of the task set, ξ_τ , was comprised of the agents' speed and the degree of resource contention, $\sum_{\tau_i} \sum_{\tau_j} 1_{R_{\tau_i}=R_{\tau_j}}$. The task-specific feature vector, γ_{τ_i} , was comprised of the task's deadline, a binary indicator for whether or not the task's precedence constraints had been satisfied, the number of other tasks sharing the given task's resource, a binary indicator for whether or not the given task's resource was available, the travel time remaining to reach the task location, the distance agent a would travel to reach τ_i , and the angular difference between the vector describing the location of agent a and the vector describing the position of τ_i relative to agent a .

We compared the performance of our pairwise approach with pointwise and naïve approaches. In the pointwise approach, training examples for selecting the highest-priority task were of the form $^{rank}\phi_{\tau_i}^m := [\xi_\tau, \gamma_{\tau_i}]$. The label $\gamma_{\tau_i}^m$ was equal to 1 if task τ_i was scheduled in observation m , and was 0 otherwise. In the naïve approach, examples were comprised of an input vector that concatenated the high-level features of the task set and the task-specific features of the form $^{rank}\phi^m := [\xi_\tau, \gamma_{\tau_1}, \gamma_{\tau_2}, \dots, \gamma_{\tau_n}]$; labels y^m were given by the index of the task τ_i scheduled in observation m .

Figures 5a-5b depict the sensitivity (true positive rate) and specificity (true negative rate), respectively, of the model. We found that a pairwise model outperformed the pointwise and naïve approaches. Within the pairwise model, a decision tree yielded the best performance: The trained decision tree was able to identify the correct task and when to schedule that task 95% of the time, and was able to accurately predict when no task should be scheduled 96% of the time.

To more fully understand the performance of a decision tree trained with a pairwise model as a function of the number and quality of training examples, we trained decision trees with the pairwise model using 15, 150, and 1,500 demonstrations. The sensitivity and specificity depicted in Figures 6a and 6b for 15 and 150 demonstrations represent the mean sensitivity and specificity of 10 models trained via random sub-sampling without replacement.

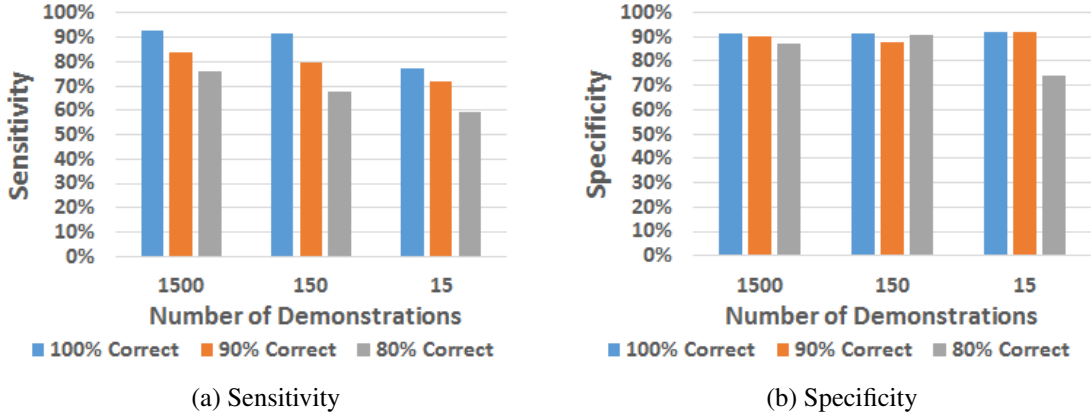


Figure 6: Figures 6a-6b depict the sensitivity and specificity of a pairwise decision tree, varying the number and proportion of correct demonstrations.

We also varied the quality of the training examples, assuming the demonstrator was operating under an ϵ -greedy approach with a $(1 - \epsilon)$ probability of selecting the correct task to schedule, and selecting another task from a uniform distribution otherwise. Our goal in this evaluation was to empirically investigate the impact of noisy demonstrations (i.e., those in which the demonstrator does not always select the “best” tasks) on the quality of the learned policy. There are a number of possible models for introducing such noise, including an epsilon-greedy approach or a softmax model. An epsilon-greedy approach is expected to produce lower-quality demonstrations compared with a noisy human demonstrator, since a human would be more likely to select the second- or third-best task when making an error than to select a task at random, thus making the LfD problem more difficult. While no model will perfectly imitate an imperfect human demonstrator, we selected an epsilon-greedy approach as a reasonably conservative method of introducing more noise than might be generated by an imperfect human demonstrator.

Training a model from pairwise comparisons of between the scheduled and each unscheduled tasks produced a comparable policy to that of the synthetic expert. The decision tree model performed well due to the modal nature of the multifaceted scheduling heuristic. Note that this data set consisted of scheduling strategies with mixed discrete-continuous functional components; performance could potentially be improved upon in future work by combining decision trees with logistic regression. This hybrid learning approach has been successful in prior ML classification tasks (Landwehr, Hall, & Frank, 2005) and can be readily applied to this apprenticeship scheduling framework. There is also an opportunity to improve performance through hyperparameter tuning (e.g., to select the minimum number of examples in each leaf of the decision tree). We leave comprehensive investigation of the relative benefits for a range of learning techniques for future work.

Note that the results presented in Figures 5a-6b were achieved without any hyperparameter tuning. For example, with the decision tree, we did not perform an inner cross-validation loop to estimate the minimum number of examples in each leaf to achieve the best performance. The purpose of this analysis was to show that, with our pairwise approach, the system can accurately learn expert heuristics from example. In the following section, we investigate how apprenticeship scheduling using a decision tree classifier can be improved upon via an inner cross-validation loop to tune the model’s hyperparameters.

5.1.1 PERFORMANCE OF DECISION TREE WITH HYPERPARAMETER TUNING

We performed our initial analysis, detailed above, to identify which techniques have inherent advantages that can be realized without extensive hyperparameter tuning. Our results indicate that the pairwise formulation for apprenticeship scheduling, in conjunction with a decision tree classifier, has advantages over alternative formulations for learning a high-quality scheduling policy. Given evidence of this advantage, we further evaluated the potential of the pairwise formulation with hyperparameter tuning.

To improve the performance of the model, we manipulated the “leafiness” of the decision tree to find the best setting to increase the accuracy of the apprenticeship scheduler. Specifically, we varied the minimum number of training examples required in each leaf of the tree. As the minimum number required for each leaf decreases, the chance of over-fitting to the data increases. Conversely, as the minimum number increases, the chance of not learning a helpful policy (under-fitting) increases. To identify the best number of leaves for generalization, we tested values for the minimum number of examples required for each leaf of the decision tree in the set $\{1, 5, 10, 25, 50, 100, 250, 500, 1000\}$. If the minimum number of examples in each leaf exceeded the total number of examples, the setting was trivially set to the total number of examples available for training.

We performed 5-fold cross-validation for each value of examples as follows: We trained an apprentice scheduler on four-fifths of the training data and tested on one-fifth of the data, and recorded the average testing accuracy across each of the five folds. Then, we used the setting of the minimum number of examples required for each leaf that yielded the best accuracy during cross-validation to train a full apprenticeship scheduling model on all of the training data (85% of the total data). Finally, we tested the full apprenticeship scheduling model on the 15% of the total data reserved for testing. Thus, none of the data used to test the full model was used to estimate the best setting for the leafiness of the tree. We repeated this procedure 10 times, randomly sub-sampling the data and taking the average performance across the 10 trials.

The sensitivity and specificity of the fully trained apprenticeship scheduling algorithm are depicted in Figures 7a and 7b for 1, 5, 15, and 150 scheduling demonstrations with homogeneous agents, and in Figures 8a and 8b for demonstrations with heterogeneous agents. As before, we also varied the quality of the training examples, assuming the demonstrator was operating under an ϵ -greedy approach with a $(1 - \epsilon)$ probability of selecting the correct task to schedule and selecting another task from a uniform distribution otherwise.

For both the homogeneous and heterogeneous cases, we found that the apprenticeship scheduling algorithm was able to average $\geq 90\%$ sensitivity and specificity either with five perfect schedules or 15 schedules generated by an operator making mistakes 20% of the time. Hyperparameter tuning substantially increased the sensitivity of the model from 59% to 82% for five scheduling examples generated by an operator making mistakes 20% of the time. (Recall that a schedule consists of allocating 20 tasks to two workers and sequencing those tasks in time.)

Through our synthetic evaluation, we have shown that our apprentice scheduling algorithm is able to learn to make sequential decisions that accurately emulate the decision making process of a mock expert. The apprenticeship scheduler model shows a robust ability to learn from sparse, noisy data. In the following sections, we investigate the ability of the apprentice scheduler to learn from scheduling demonstrations produced by experts performing real-world scheduling tasks.

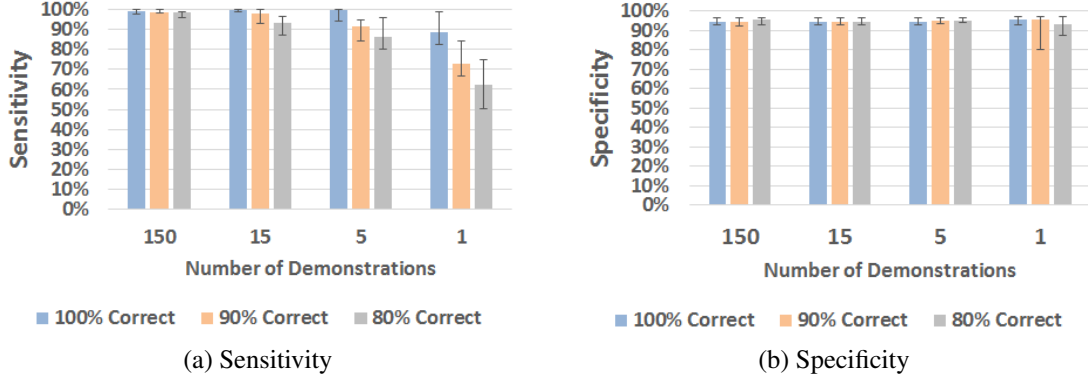


Figure 7: Figures 7a-7b depict the sensitivity and specificity for a pairwise decision tree tuned for leafiness, varying the number and proportion of correct demonstrations. The corresponding data set comprised schedules with homogeneous agents.

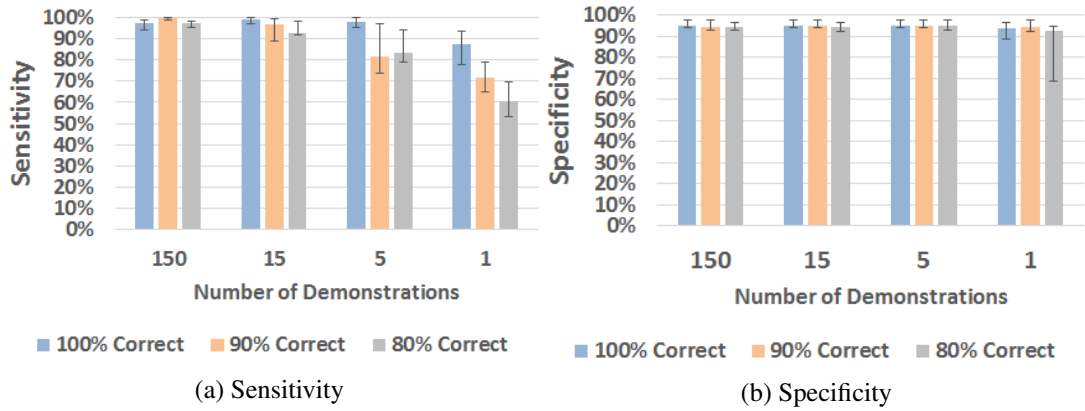


Figure 8: Figures 8a-8b depict the sensitivity and specificity for a pairwise decision tree tuned for leafiness, varying the number and proportion of correct demonstrations. The corresponding data set comprised schedules with heterogeneous agents.

5.2 Real-World Data Set: ASMD

We trained a decision tree with our pairwise scheduling model and tested its performance via leave-one-out cross-validation involving 16 real demonstrations in which a player successfully protected the ship from all enemy missiles. Each demonstration originated from a unique threat scenario. Features for each decoy/missile pair (or null decoy deployment due to inaction) included indicators for whether a decoy had been placed such that a missile was successfully distracted by that decoy, whether a missile would be lured into hitting the ship due to decoy placement, or whether a missile would be unaffected by decoy placement.

Across all 16 scenarios, the mean player score was $74,728 \pm 26,824$. With only 15 examples of expert human demonstrations, our apprenticeship scheduling model achieved a mean score of 87,540, with a standard deviation of 16,842. We hypothesized that scores produced by the learned policy would be statistically significantly better than the scores achieved by the human demonstrators. The null hypothesis stated that the number of scenarios in which the apprenticeship scheduling model achieved superior performance would be less than or equal to the number of scenarios in which the mean score of the human demonstrators was superior to that of the apprenticeship scheduler. We set the significance level at $\alpha = 0.05$, which means that the risk of identifying a difference between the mean scores earned by the apprenticeship scheduler and the set of human performers when no such difference exists is less than 5%.

Results from a binomial test rejected the null hypothesis, indicating that the learned scheduling policy performed better than the human demonstrators in significantly more scenarios (12 versus 4 scenarios; $p < 0.011$). In other words, we can say with 95% certainty that the apprenticeship scheduler outperformed the average human player for the majority of the presented missile defense scenarios. This promising result was achieved using a relatively small training set, and suggests that learned policy can form the basis for a training tool to improve the average human player's score.

5.3 Real-World Data Set: Labor and Delivery

Currently, nurse resource managers commonly operate without technological decision-making aids. As such, it is imprudent to introduce a fully autonomous solution for resource management, as doing so could have life-threatening consequences for practitioners unfamiliar with such automation. Rather, research has shown that a semi-autonomous system is preferable when integrating machines into human cognitive workflows (Kaber & Endsley, 1997; Wickens, Li, Santamaria, Sebok, & Sarter, 2010). Such a system would provide recommendations that a human supervisor could then accept or modify, and would be placed within the “4-6” range on Sheridan’s 10-point scale for levels of automation (Parasuraman, Sheridan, & Wickens, 2000).

We found it prudent to test our apprenticeship scheduling technique with the algorithm offering recommendations to labor nurses who would evaluate how acceptable they found the quality of each recommendation. Specifically, we wanted to test whether the algorithm was able to learn to differentiate between high- and low-quality resource management decisions. If nurses accepted what the apprenticeship scheduler had learned to be high-quality advice while rejecting what the scheduler had learned to be low-quality advice, we could be reasonably confident that the apprentice scheduler had captured the desired resource management policy.

The first step, then, was to train a decision tree using the pairwise scheduling model based on the data set described in Section 4.3.1 of resource nurses’ scheduling decisions. Recall that this data set

consisted of the results of expert resource nurses playing the simulation for $2^{1/2}$ hours, simulating 60 hours of elapsed time on a real labor floor, and yielding a data set of more than 3,013 decisions.

Second, we invited 15 labor nurses, none of whom were among those involved in training the algorithm, to play the same simulation used to collect the data (Figure 4). However, instead of purely soliciting decisions from the player, the simulation used the apprenticeship scheduling policy to offer recommendations about how to manage patients. Specifically, whenever a new patient arrived in the simulated waiting room, the apprenticeship scheduler would offer advice recommending 1) which of six wards to admit that patient to, 2) which bed within that ward to place that patient, and 3) which nurse should care for that patient. Nurses would then either accept the advice, automatically implementing the decision, or reject the advice and implement their own decisions.

In order to generate high-quality advice, the apprenticeship scheduler simply applied Equation 7. To generate low-quality advice, the apprenticeship scheduler applied Equation 22, which changes the maximization to a minimization, as follows:

$$\tau_i^* = \underset{\tau_i \in \tau}{\operatorname{argmin}} \sum_{\tau_x \in \tau} f_{\text{priority}}(\tau_i, \tau_x) \quad (22)$$

However, such a minimization could create a straw-man counterpoint to the high-quality advice, demonstrating only that the apprenticeship scheduler learned at least hard constraints (e.g., “do not assign a patient to an occupied bed”) rather than a gradation over feasible actions (e.g., “assign a less-busy nurse to a new patient rather than a busier nurse”). As such, we also used the apprenticeship scheduler to generate low-quality but feasible advice by only considering $\tau_i \in \tau$ such that τ_i was feasible, as determined through a manually-encoded schedulability test.

For each of the 15 nurse players, we conducted two trials with the simulation offering advice. In one trial, the advice was high-quality; in the other, the simulation offered low-quality advice randomly chosen to be low-quality but feasible or low-quality and infeasible. We hypothesized that nurses would accept advice during the high-quality trials and reject advice during the low-quality trials (regardless of feasibility). Each simulation trial was randomly generated, with each player experiencing different scenarios with differing advice. On average, a nurse would receive 8.5 recommendations per trial, resulting in a total of 256 recommendations across all nurses and trials.

The nurses accepted high-quality advice 88.4% of the time (114 of 129 high-quality recommendations), while rejecting low-quality advice 88.2% of the time (112 of 127 low-quality recommendations), indicating that the apprenticeship scheduling technique is able to learn a high-quality model for resource management decision making in the context of labor and delivery. In other words, the apprenticeship scheduler was able to learn context-specific strategies for hospital resource allocation and apply them to make reasonable suggestions about which tasks to perform and when.

Anecdotally, some of the advice was not accepted for reasons that could be easily remedied: For example, upon initiation of the test, we were unaware that one room on the labor and delivery floor was unique because it uniquely contained cardiac monitoring equipment. As such, the algorithm did not know to reason about that feature and sometimes offered a recommendation that was feasible but less preferable for patients with cardiac-related comorbidities. It was not until later that we learned from the nurses about this particular feature. Such findings motivate the need for active learning for improved feature solicitation in future work. We also note that inter-operator agreement among nurse demonstrators is unlikely to be 100%. For these reasons, we believe learning a policy that can generate advice validated to be correct nearly 90% of the time is a favorable result.

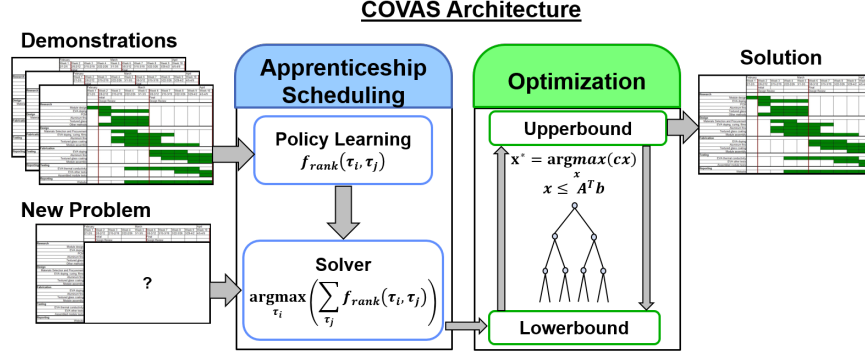


Figure 9: The COVAS architecture.

6. Model for Collaborative Optimization via Apprenticeship Scheduling

Apprenticeship scheduling is designed to simply emulate human expert scheduling decisions; in this work, we also use the apprenticeship scheduler in conjunction with optimization to automatically and efficiently produce optimal solutions to challenging real-world scheduling problems. Our approach, called Collaborative Optimization via Apprenticeship Scheduling (COVAS), involves applying apprenticeship scheduling to generate a favorable (if suboptimal) initial solution to a new scheduling problem. To guarantee that the generated schedule is serviceable, we augment the apprenticeship scheduler to solve a constraint satisfaction problem, ensuring that the execution of each scheduling commitment does not directly result in infeasibility for the new problem. COVAS uses this initial solution to provide a tight bound on the value of the optimal solution, substantially improving the efficiency of a branch-and-bound search for an optimal schedule.

We show that COVAS is able to leverage good (but imperfect) human demonstrations to quickly produce globally optimal solutions. We also report that COVAS can transfer an apprenticeship scheduling policy learned for a small problem to optimally solve problems with twice as many variables as any shown during training, and produce an optimal solution an order of magnitude faster than mathematical optimization alone. Here, we provide an overview of the COVAS architecture and present its two components: the policy learning and optimization routines.

6.1 COVAS Architecture

The system (Figure 9) takes as input a set of domain expert scheduling demonstrations (e.g., Gantt charts) that contains information describing which agents complete which tasks, when and where. These demonstrations are passed to an apprenticeship scheduling algorithm that learns a classifier, $f_{\text{priority}}(\tau_i, \tau_j)$, to predict whether the demonstrator(s) would have chosen scheduling action τ_i over action $\tau_j \in \tau$. Next, COVAS uses $f_{\text{priority}}(\tau_i, \tau_j)$ to construct a schedule for a new problem. The system creates an event-based simulation of this new problem and runs this simulation in time until all tasks have been completed. In order to complete tasks, COVAS uses $f_{\text{priority}}(\tau_i, \tau_j)$ at each moment in time to select the best scheduling action to take. We describe this process in detail in the next section. COVAS then provides this output as an initial seed solution to an optimization subroutine (i.e., a MILP solver). The initial solution produced by the apprenticeship scheduler improves the

efficiency of a search by providing a bound on the objective function value of the optimal schedule. This bound informs a branch-and-bound search over the integer variables (Bertsimas & Weismantel, 2005), enabling the search algorithm to prune areas of the search tree and focus its search on areas that can yield the optimal solution. After the algorithm has identified an upper- and lowerbound within some threshold, COVAS returns the solutions that have proven optimal within that threshold. Thus, an operator can use COVAS as an anytime algorithm and terminate the optimization upon finding a solution that is acceptable within a provable bound.

6.2 Apprenticeship Scheduling Subroutine

In Section 3, we presented our apprenticeship scheduling algorithm, which is centered around learning a classifier, $f_{priority}(\tau_i, \tau_j)$, to predict whether an expert would take scheduling action τ_i over τ_j . With this function, we can then predict which single action τ_i^* amongst a set of actions τ the expert would take by applying Equation 7. In this section, we build upon this formulation and integrate it into our collaborative-optimization via apprenticeship scheduling framework.

As a subroutine within COVAS, $f_{priority}(\tau_i, \tau_j)$ is applied to obtain the initial solution to a new scheduling problem as follows: First, the user must instantiate a simulation of the scheduling domain; then, at each time step in the simulation, take the scheduling action predicted by Equation 7 to be the action that the human demonstrators would take. This equation identifies the task τ_i with the highest importance marginalized over all other tasks $\tau_j \in \tau$. Unlike our original formulation in Section 3, each selected action is validated using a schedulability test (i.e., solving a constraint satisfaction problem) to ensure that direct application of that action does not violate the constraints of the new problem. For example, in anti-ship missile defense, one would check to ensure that the given action does not result in a suicidal deployment (i.e., the decoy directly causes a missile to impact the ship). This test must be fast, so as to make the benefit to feasibility and optimality in the resulting schedule worth the additional complexity. If, at a given time step, τ_i^* does not pass the schedulability test, COVAS uses Equation 7 for all $\tau_i \in \tau \setminus \tau_i^*$ to consider the second-best action. If no action passes the schedulability test, no action is taken during that time step.

While the schedulability test forces the apprenticeship scheduling algorithm to follow a subset of the full constraints in the MILP formulation, it is possible that the algorithm may not successfully complete all tasks. Here, we model tasks as optional and use the objective function to maximize the total number of tasks completed. In turn, constraints for a task that the apprenticeship scheduling algorithm did not satisfactorily complete can be turned off, with a corresponding penalty in the objective function score. Thus, an initial seed solution that has not completed all tasks (i.e., satisfied all constraints to complete the task) can still be helpful for seeding the MILP.

6.3 Optimization Subroutine

For optimization, we employ mathematical programming techniques to solve mixed-integer linear programs via branch-and-bound search. COVAS incorporates the solution produced by the apprenticeship scheduler to seed a mathematical programming solver with an initial solution, which is a built-in capability provided by many off-the-shelf, state-of-the-art MILP solvers, including CPLEX² and Gurobi³. This seed provides a tight bound on the objective function value of the optimal solution, which serves cut the search space; these cuts allow COVAS to more quickly hone in on the

2. IBM ILOG CPLEX Optimization Studio <http://www-03.ibm.com/software/products/en/ibmilogcpleoptstud>

3. Gurobi Optimization, Inc. <http://www.gurobi.com>

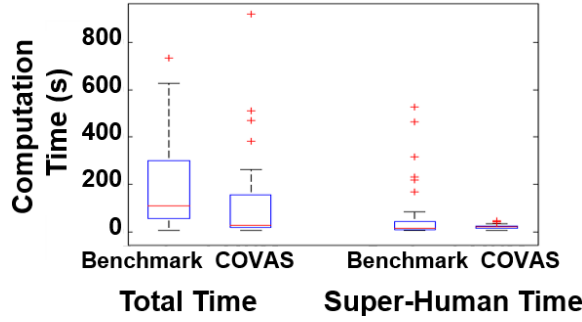


Figure 10: The total computation time for COVAS, as well as the time COVAS required to identify a solution superior to that resulting from a human expert’s demonstration. Results for the benchmark and COVAS are depicted offset to the left and right of each position along the x-axis, respectively.

optimal solution. Furthermore, this approach allows COVAS to quickly achieve a bound on the optimality of the solution provided by the apprenticeship scheduling subroutine. In such a manner, an operator can determine whether the apprenticeship scheduling solution is acceptable or whether waiting for successive solutions from COVAS is warranted.

7. Results and Discussion

In this section, we empirically validate that COVAS is able to generate optimal solutions more efficiently than state-of-the-art optimization techniques. We also analyze the sensitivity of the computational time COVAS required to find an optimal solution as a function of the quality of the scheduling policy learned by the apprenticeship scheduling algorithm.

7.1 Validation Against Expert Benchmark

In this section, we empirically validate that COVAS is able to generate optimal solutions more efficiently than state-of-the-art optimization techniques. As a baseline benchmark, we solve a pure MILP formulation (Appendix A Equations 23-44) using Gurobi, which applies state-of-the-art techniques for heuristic upperbounds, cutting planes and LP relaxation lowerbounds. We set the optimality threshold at 10^{-3} . For the apprenticeship scheduling subroutine’s schedulability test, we apply Equations 36-37 as a constraint satisfaction check when testing the feasibility of action τ_i^* , given by applying Equation 7. With regard to tasks within the apprenticeship scheduler’s seed solution that are not satisfactorily completed, the MILP can leave those tasks incomplete to start by initially setting $V_m \leftarrow 0$.

We trained COVAS’ apprenticeship scheduling algorithm on demonstrations of experts’ solutions to unique ASMD scenarios (save for one “hold-out” scenario) from the ASMD data set described in Section 4.2. We then tested COVAS on the hold-out scenario. We also applied a pure MILP benchmark on this scenario and compared the performance of COVAS to the benchmark. We generated one data point for each unique demonstrated scenario (i.e., leave-one-out cross-validation) to validate the benefit of COVAS.

Figure 10 consists of two performance indicators: The total computation time required for the MILP benchmark and COVAS to solve for the optimal solution is depicted on the left; to the right is the computation time required for the benchmark and COVAS to identify a solution better than that

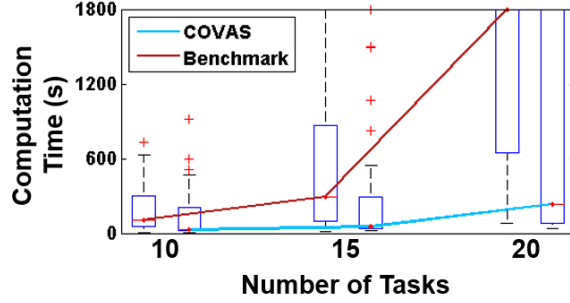


Figure 11: The total computation time needed for COVAS and the MILP benchmark to identify the optimal solution for the tested scenarios. Results for the benchmark and COVAS are depicted offset to the left and right of each position along the x-axis, respectively.

provided by a human expert. This figure indicates that COVAS was not only able to improve overall optimization time, but that it also substantially improved computation time for solutions superior to those produced by human experts. The average improvements in computation time with COVAS were 6.7x the overall optimization time and 3.1x the expert-generated solutions.

Next, we evaluated COVAS’ ability to transfer prior learning to more-challenging task sets. We trained on a level in the ASMD game in which a total of 10 missiles of varying types came from specific bearings at given times. We randomly generated a set of scenarios involving 15 and 20 missiles, with bearings and times randomly sampled with replication from the set of bearings used in the 10-missile scenario.

Figure 11 depicts the computation time required by COVAS and the MILP benchmark to identify the optimal solution for scenarios involving 10, 15 and 20 missiles. The average improvement to computation time with COVAS was 4.6x, 7.9x, and 9.5x, respectively, demonstrating that COVAS is able to efficiently leverage the solutions of human domain experts to quickly solve problems twice as large as those the demonstrator provided for training.

7.2 Sensitivity Analysis of COVAS to Apprenticeship Scheduler’s Learned Policy

Here, we assess the sensitivity of the computational time COVAS required to find an optimal solution as a function of the quality of the scheduling policy learned by the apprenticeship scheduling algorithm.

7.2.1 SENSITIVITY ANALYSIS DESIGN

We sought to understand how incorrect predictions generated by the apprenticeship scheduling algorithm’s classifier, $f_{priority}(\cdot, \cdot)$, would affect COVAS’ computational efficiency. We considered three classes of mistakes that the apprenticeship scheduler could make when creating an initial schedule: two types of mistakes related to agent allocation (swapping tasks among agents and the misallocation of agents to a particular task), as well as task sequencing errors. We generated a synthetic dataset involving these three error classes as follows:

- *Allocation: Swapping*: Select two tasks with uniform probability, τ_i and τ_j , such that the agent a assigned to τ_i is different from the agent a' assigned to τ_j , and subsequently swap their assignment such that agent a' now performs τ_i and vice-versa.

- *Allocation: Stealing*: Select one task, τ_i , with uniform probability, where τ_i is assigned to agent a , and reassign it to a different agent, a' .
- *Sequencing*: Select two tasks, τ_i and τ_j , with uniform probability, such that τ_i precedes τ_j in the schedule, and reverse their order such that τ_j now precedes τ_i .

Table 2 depicts the experiment design for our sensitivity analysis. We incorporated the synthetic dataset because the scheduling problem has a well-defined objective function and set of constraints for use in the optimization component of COVAS, and also because the data set encompasses three different types of scheduling problems. The problem domain and the mock demonstrator’s heuristics for each problem were defined in Section 5.1. We generated 15 problems for each problem type, misclassification type, and number of misclassifications. Five replicates were generated for each problem, with the replicates varying according to misclassification type (e.g., switching the ordering of τ_i and τ_j versus switching τ_p and τ_q). In total, the analysis involved $3 \times 3 \times 3 = 27$ different experimental settings and $27 \times 15 \times 5 = 2,025$ total data points.

7.2.2 STATISTICAL MODEL FOR THE ANALYSIS

We performed a mixed-effects multiple linear regression to quantify the sensitivity of COVAS with respect to the quality of the apprenticeship scheduling policy. The dependent variable was the *computational time* required by COVAS to identify the optimal solution. The independent variables were the *problem type* (vehicle routing, resource contention, and temporal requirements), the *misclassification type* (allocation-based swapping and stealing and sequencing-based errors), the *number of errors* (one, two, or three), and the *objective function value* of the schedule produced by the apprenticeship scheduling algorithm (normalized to the objective function value of the optimal solution). The aforementioned independent variables were modeled as fixed effects. We also included random effects for the individual problem and for the optimality of the apprenticeship scheduler’s solution as a function of the fixed-effects independent variables. We applied a Box-Cox transformation (Box & Cox, 1964) to normalize the data for regression, which returned $\lambda = 0.223$ as the optimal transformation factor. We established statistical significance for the regression parameters at the $\alpha = 0.05$ level.

7.2.3 RESULTS AND DISCUSSION

Table 3 reports the statistical results of our sensitivity analysis.⁴ There are three key takeaways from these results: First, the primary driver of COVAS’ computational time was the objective function

4. The Akaike information criterion (Akaike, 1974) and Bayesian information criterion (Schwarz et al., 1978) are 2,968.1 and 3,016, respectively. The log likelihood of the model is -1,474.1, and the deviance is 2,948.1.

Problem Type (Heuristic Applied)	Travel Distance (Equation 9)									Resource Contention (Eq. 10)	Temporal Requirements (Eq. 11)
Misclassification Type	Swapping			Stealing			Sequencing		
# Misclassifications	1	2	3	1	2	3	1	2	3

Table 2: This table depicts the experimental design for COVAS’ sensitivity analysis.

Parameter		Estimate	Confidence Interval	p-value
Intercept		-13.095	(-15.047, -11.143)	< 0.001
Obj. Fn. Val. of AS's schedule (Normalized)		5.256	(3.937, 6.575)	< 0.001
Number of Classification Errors		1.357	(0.630, 2.083)	< 0.001
Allocation vs. Sequencing Misclassification	Allocation	6.471	(4.942, 7.999)	< 0.001
	Sequencing	-	-	-
Allocation Misclassification Type	Steal	1.018	(-0.448, 2.484)	0.173
	Swap	-	-	-
Problem Type:	Temporal Req.	3.074	(1.300, 4.848)	< 0.001
	Resource Contention	0.271	(-1.893, 2.435)	0.806
	Travel Distance	-	-	-

Table 3: This table depicts the results of the regression analysis. Entries with dashes indicate that the associated parameter setting was the baseline. Statistically significant values are in bold.

value of the schedule produced by the apprentice scheduler's learned policy – not the number of misclassification errors made when constructing the schedule. As shown in the third and fourth rows of Table 3, the impact of the objective function value on COVAS' computational efficiency was $5.256/1.357 \approx 4$ times more than for the individual classification errors made by $f_{priority}(\cdot, \cdot)$.

Second, there was a statistically significant effect for allocation- versus sequencing-based perturbations. The regression analysis shows that COVAS's improvement in computation time lessens when considering allocation-based perturbations ($p < 0.001$). However, there was no statistically significant effect present between allocation-based swapping and stealing errors ($p = 0.173$).

Third, we observed a sensitivity to the problem type / heuristic applied, with the problem type emphasizing temporal requirements (for which the heuristic in Equation 11 is applied) representing the most challenging problem. There was not a significant difference between the resource contention and VRP-style problem types ($p = 0.806$).

Finally, note that COVAS showed an improvement in computation time relative to a commercial, state-of-the-art solver regardless of problem and misclassification type and number. We performed separate regression analyses for each problem type and as a function of whether the classification errors by the apprenticeship scheduler were allocation- or sequencing-based. Table 4 depicts the objective function value of the apprenticeship scheduler's schedule for which COVAS no longer demonstrates an advantage in computation time (relative to a state-of-the-art solver) when marginalizing over the number of errors. For example, if COVAS is scheduling a problem for which travel distance is key (and the apprentice scheduler was trained on a mock demonstrator applying Equation 9) and the apprentice scheduler makes a number of (1, 2, or 3) allocation-based swapping-type classification errors while constructing the schedule, COVAS is faster than a state-of-the-art benchmark, so long as the objective function value of the schedule produced by the apprentice scheduler is no worse than 1.88 times that of the optimal solution. The results show that COVAS demonstrates an advantage for all problem and misclassification types.

	Travel Distance	Resource Contention	Temporal Requirements
Allocation-Based Swapping Errors	1.68	1.38	1.13
Allocation-Based Stealing Errors	1.88	1.43	1.14
Sequencing-Based Errors	1.34	1.27	1.12

Table 4: This table depicts the maximum objective function value of the apprenticeship scheduler’s initial solution (normalized to that of the optimal solution) to provide COVAS’ optimization subroutine with an improvement in computation time.

7.2.4 CONCLUSION

The results from our sensitivity analysis support the hypothesis that COVAS is robust to misclassification errors by the apprenticeship scheduler’s learned policy. The data indicate that the apprenticeship scheduler’s solution quality is the dominating factor, rather than the number of individual mistakes made when generating that solution. While further investigation of other scheduling problems is warranted, the variants within this data set inform our understanding of the sensitivity of COVAS to imperfections in the learned policy across a range of problem types.

These results also provide insight into ways that we can potentially improve COVAS’ apprenticeship scheduling subroutine. For example, COVAS is more sensitive to the objective function value of the schedule produced by the apprenticeship scheduler’s policy, while being somewhat robust to the number of errors made by the apprenticeship scheduler when constructing the schedule. As such, imitation learning (Ross, Gordon, & Bagnell, 2011; Cheng & Boots, 2018) approaches, which attempt to bootstrap off of an initial, learned policy, may be able to improve the quality of the solutions produced by the apprenticeship scheduler. Further, a Bayesian IRL approach (Michini & How, 2012; Ramachandran & Amir, 2007), which seeks to infer a policy mimicking an “ideal demonstrator,” may also be able to leverage demonstrations to better guide COVAS’ optimization subroutine. In future work, we will seek to determine how to combine such approaches with our pairwise training procedure for COVAS’ apprenticeship scheduling subroutine.

8. Limitations and Future Work

The core of the apprenticeship scheduling algorithm is learning a classifier, $f_{\text{priority}}(\tau_i, \tau_j)$, to predict whether a human expert would take action τ_i over τ_j . The output of $f_{\text{priority}}(\tau_i, \tau_j)$ is a probability in $[0, 1]$. This pairwise approach has a number of key advantages: For example, it is nonparametric with regard to the number of tasks, meaning one can learn from problems involving n actions and apply that knowledge to problems with $n' \neq n$ actions. However, there are two interesting anomalies inherent in this approach: First, one could hypothetically evaluate $f_{\text{priority}}(\tau_i, \tau_j)$ and find that it predicts that the expert has a higher probability of taking action τ_i than τ_j ; however, evaluating $\operatorname{argmax}_{\tau_i \in \tau} \sum_{\tau_j \in \tau} f_{\text{priority}}(\tau_i, \tau_j)$ could predict that τ_j is the action most likely to be taken by the expert. The second anomaly entails the lack of a guarantee that the transitive property will hold for arbitrary $f_{\text{priority}}(\tau_i, \tau_j)$. For example, it could be that $f_{\text{priority}}(\tau_i, \tau_j) > 0.5$, $f_{\text{priority}}(\tau_j, \tau_k) > 0.5$, but also $f_{\text{priority}}(\tau_k, \tau_i) > 0.5$ for some τ_i, τ_j , and τ_k . Through our evaluation, we have shown that the formulation for apprenticeship scheduling can learn high-quality

policies from human domain experts’ demonstrations. However, an interesting aim for future work would be to study these anomalies, quantify their effects – if any – and develop a formulation to alter these effects. Appendix B provides an example formulation for how to mitigate such anomalies.

COVAS also has some interesting aspects that merit future investigation. COVAS is able to leverage expert scheduling demonstrations to speed up the computation of provable, globally optimal scheduling solutions. However, the approach is still limited by the quality of the demonstrations provided by experts, as well as the ability of the apprenticeship scheduling algorithm to generalize the information within those demonstrations. The MILP’s computation time is expedited by tight upperbounds (i.e., an initial seed) provided by the apprenticeship scheduling algorithm. If the apprenticeship scheduling algorithm is unable to provide a tight upperbound, the MILP’s computation time may not be significantly improved. In future work, we will explore potential extensions to the apprenticeship scheduling algorithm to improve its ability to learn from noisy demonstrations. One approach could be to incorporate a trustworthiness metric à la Zhang and Burns (2009) directly into the training of the classifier to uncover a latent action ranking. For example, instead of binary labels, we could reformulate the problem to be one of regression, where positive and negative labels are proportional and inversely proportional, respectively, to the fidelity of the demonstrator.

Finally, apprenticeship scheduling with heterogeneous demonstrators is an important area for future work. In this paper, we demonstrated the ability to learn from 1) homogeneous demonstrators with varying quality and quantity of training data in a synthetic domain and 2) heterogeneous demonstrators in multiple real-world domains (i.e., healthcare and ship defense). Future lines of research include learning clusters of operator archetypes through unsupervised or semi-supervised learning so that the apprenticeship scheduler can better account for individual differences between operators. If each demonstrator applies different strategies, it will be more difficult to generalize across operators (Sammut et al., 1992). However, if there are a small number of demonstrator types relative to the number of demonstrators, it may be possible to leverage commonality within or across types to bootstrap the learning process.

9. Conclusions

In this paper, we proposed a technique for apprenticeship scheduling that relies upon a pairwise comparison of scheduled and unscheduled tasks to learn a model for task prioritization. We validated that our apprenticeship scheduling algorithm is able to learn high-quality scheduling policies from demonstration across both synthetic data and real-world data sets. Specifically, apprenticeship scheduling can learn from nurse resource managers to make scheduling decisions that are accepted by resource nurses 90% of the time, and can learn from military experts to solve a variant of the weapon-to-target assignment problems with better performance than the average human expert. Next, we embedded this apprenticeship scheduling algorithm within a ML-optimization framework. This algorithm, COVAS, leverages the ability of apprenticeship scheduling to capture the knowledge of human domain experts in order to produce optimal solutions for complex real-world scheduling problems. We validated our technique using a data set collected from human experts solving an anti-ship missile defense problem, and showed that our approach can substantially improve upon solutions produced by experts, at a rate up to 9.5 times faster than an optimization approach that does not incorporate human expert demonstration.

References

- Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 1.
- Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proc. ICML*.
- Akaike, H. (1974). A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6), 716–723.
- Aleotti, J., & Caselli, S. (2006). Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems*, 54(5), 409–413.
- Anders, U., & Korn, O. (1999). Model selection in neural networks. *Neural Networks*, 12(2), 309 – 323.
- Aydin, M. E., & Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2), 169–178.
- Banerjee, A. G., Ono, M., Roy, N., & Williams, B. (2011). Regression-based LP solver for chance-constrained finite horizon optimal control with nonconvex constraints. In *Proc. ACC*, pp. 131–138. IEEE.
- Baranes, A., & Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1), 49–73.
- Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2), 41–77.
- Basu, C., Hirsh, H., & Cohen, W. (1998). Recommendation as classification: Using social and content-based information in recommendation. In *Proc. Fifteenth National Conference on Artificial Intelligence*, pp. 714–720. AAAI Press.
- Benton, J., Coles, A. J., & Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs.. In *Proc. ICAPS*, Vol. 77, p. 78.
- Berry, P., Peintner, B., Conley, K., Gervasio, M., Uribe, T., & Yorke-Smith, N. (2006). Deploying a personalized time management agent. In *Proc. AAMAS*, pp. 1564–1571.
- Berry, P. M., Gervasio, M., Peintner, B., & Yorke-Smith, N. (2011). Ptime: Personalized assistance for calendaring. *ACM Trans. Intell. Syst. Technol.*, 2(4), 40:1–40:22.
- Bertsimas, D., & Weismantel, R. (2005). *Optimization over Integers*. Dynamic Ideas, Belmont.
- Boese, K. D., Kahng, A. B., & Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2), 101–113.
- Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., & Pool, D. (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21, 135–191.
- Boutilier, C., Brafman, R. I., Hoos, H. H., & Poole, D. (1999). Reasoning with conditional ceteris paribus preference statements. In *Proc. UAI, UAI'99*, pp. 71–80.
- Box, G. E., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 211–252.

- Bradtke, S. J., & Duff, M. O. (1994). Reinforcement learning methods for continuous-time markov decision problems. In *Proc. NIPS*, pp. 393–400. MIT Press.
- Brunet, L., Choi, H.-L., & How, J. P. (2008). Consensus-based auction approaches for decentralized task assignment. In *Proc. AIAA Guidance, Navigation, and Control Conference (GNC)*, Honolulu, HI.
- Bullard, K., Akgun, B., Chernova, S., & Thomaz, A. L. (2016). Grounding action parameters from demonstration. In *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*, pp. 253–260. IEEE.
- Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. SMC Part C*, 38(2), 156–172.
- Cai, D., He, X., Wen, J.-R., & Ma, W.-Y. (2004). Block-level link analysis. In *Proc. 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '04*, pp. 440–447. ACM.
- Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proc. ICML*, pp. 129–136. ACM.
- Chandramohan, S., Geist, M., Lefevre, F., & Pietquin, O. (2011). User simulation in dialogue systems using inverse reinforcement learning. In *Interspeech*, pp. 1025–1028.
- Chen, J., & Askin, R. G. (2009). Project selection, scheduling and resource allocation with time dependent returns. *European Journal of Operational Research*, 193, 23–34.
- Cheng, C.-A., & Boots, B. (2018). Convergence of value aggregation for imitation learning. *arXiv preprint arXiv:1801.07292*.
- Cheng, T.-H., Wei, C.-P., & Tseng, V. S. (2006). Feature selection for medical data mining: Comparisons of expert judgment and automatic approaches. In *Proc. CBMS*, pp. 165–170.
- Chernova, S., & Veloso, M. (2007). Confidence-based policy learning from demonstration using gaussian mixture models. In *Proc. AAMAS*, pp. 233:1–233:8. ACM.
- Chernova, S., & Veloso, M. (2008). Multi-thresholded approach to demonstration selection for interactive robot learning. In *Proc. IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE.
- Cho, Y. H., Kim, J. K., & Kim, S. H. (2002). A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, 23(3), 329 – 342.
- Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., & Sartin, M. (1999). Combining content-based and collaborative filters in an online newspaper. In *Proc. ACM SIGIR Workshop on Recommender Systems*.
- Das, T., Gosavi, A., Mahadevan, S., & Marchallick, N. (1999). Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45, 560–574.
- Do, M. B., & Kambhampati, S. (2003). Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res.(JAIR)*, 20, 155–194.
- Dubois, D., & Fortemps, P. (1999). Computing improved optimal solutions to max–min flexible constraint satisfaction problems. *European Journal of Operational Research*, 118(1), 95–126.

- Gambardella, L. M., Éric Taillard, & Agazzi, G. (1999). MACS-VRPTW: A multiple colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pp. 63–76. McGraw-Hill.
- Gombolay, M., Jensen, R., Stigile, J., Son, S.-H., & Shah, J. (2016). Decision-making authority, team efficiency and human worker satisfaction in mixed human-robot teams. In *Proc. IJCAI*, New York City, NY, U.S.A.
- Gombolay, M., & Shah, J. (2015). Schedulability analysis of task sets with upper- and lower-bound temporal constraints. *Journal of Aerospace Information Systems*, 11(12), 821–841.
- Gombolay, M., Wilcox, R., & Shah, J. (2013). Fast scheduling of multi-robot teams with temporospatial constraints. In *Proc. RSS*, Berlin, Germany.
- Gombolay, M., Yang, X. J., Hayes, B., Seo, N., Liu, Z., Wadhwania, S., Yu, T., Shah, N., Golen, T., & Shah, J. (2016). Robotic assistance in coordination of patient care. In *Proceedings RSS*, Ann Arbor, MI, U.S.A.
- Grano, M. L. D., Medeiros, D. J., & Eitel, D. (2009). Accommodating individual preferences in nurse scheduling via auctions and optimization. *Healthcare Management Science*, 12, 228–242.
- Grollman, D. H., & Jenkins, O. C. (2008). Sparse incremental learning for interactive robot control policy estimation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3315–3320. IEEE.
- Haveliwala, T. H. (2002). Topic-sensitive PageRank. In *Proc. WWW*, pp. 517–526. ACM.
- Herbrich, R., Graepel, T., & Obermayer, K. (2000). *Large Margin Rank Boundaries for Ordinal Regression*, chap. 7, pp. 115–132. MIT Press.
- Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1), 5–53.
- Huang, C.-M., & Mutlu, B. (2014). Learning-based modeling of multimodal behaviors for human-like robots. In *Proc. HRI*, pp. 57–64.
- Ibnkahla, M. (2000). Applications of neural networks to digital communications - a survey. *Signal Processing*, 80(7), 1185 – 1215.
- Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. ICRA*, Vol. 2, pp. 1398–1403.
- Inamura, T., Inaba, M., & Inoue, H. (1999). Acquisition of probabilistic behavior decision model based on the interactive teaching method. In *Proc. International Conference on Advanced Robotics*.
- Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. Symposium on Theory of computing*, pp. 604–613.
- Jin, R., Valizadegan, H., & Li, H. (2008). Ranking refinement and its application to information retrieval. In *Proc. Conference on WWW*, pp. 397–406.
- Jones, E., Dias, M., & Stentz, A. (2011). Time-extended multi-robot coordination for domains with intra-path constraints. *AuRo*, 30(1), 41–56.

- Kaber, D. B., & Endsley, M. R. (1997). Out-of-the-loop performance problems and the use of intermediate levels of automation for improved control system functioning and safety. *Process Safety Progress*, 16(3), 126–131.
- Kehle, S. M., Greer, N., Rutks, I., & Wilt, T. (2011). Interventions to improve veterans' access to care: A systematic review of the literature. *Journal of General Internal Medicine*, 26(2), 689–696.
- Khatib, L., Morris, P., Morris, R., & Rossi, F. (2001). Temporal constraint reasoning with preferences. Tech. rep., Moffet Field, CA, USA.
- Kim, H. K., Kim, J. K., & Ryu, Y. (2009). Personalized recommendation over a customer network for ubiquitous shopping. *IEEE Transactions on Services Computing*, 2.
- Kim, J. K., Cho, Y. H., Kim, W. J., Kim, J. R., & Suh, J. H. (2002). A personalized recommendation procedure for internet shopping support. *Electronic Commerce Research and Applications*, 1(3-4), 301 – 313.
- Konidaris, G., & Barto, A. (2007). Building portable options: Skill transfer in reinforcement learning. In *Proc. IJCAI*, pp. 895–900.
- Konidaris, G., Kuindersma, S., Grupen, R., & Barto, A. (2011a). Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 0278364911428653.
- Konidaris, G., Osentoski, S., & Thomas, P. (2011b). Value function approximation in reinforcement learning using the fourier basis. In *Proc. AAAI*, pp. 380–385.
- Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8), 30–37.
- Korsah, G. A., Stentz, A., & Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *IJRR*, 32(12), 1495–1512.
- Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4(Dec), 1107–1149.
- Landwehr, N., Hall, M., & Frank, E. (2005). Logistic model trees. *Machine Learning*, 59(1-2), 161–205.
- Lee, Z.-J., Su, S.-F., & Lee, C.-Y. (2003). Efficiently solving general weapon-target assignment problem by genetic algorithms with greedy eugenics. *IEEE Trans. SMC Part B*, 33(1), 113–121.
- Li, P., Wu, Q., & Burges, C. J. (2007). Mcrank: Learning to rank using multiple classification and gradient boosting. In Platt, J., Koller, D., Singer, Y., & Roweis, S. (Eds.), *Proc. NIPS*, pp. 897–904, Cambridge, MA: MIT Press.
- Lihua, W., Lu, L., Jing, L., & Zongyong, L. (2005). Modeling user multiple interests by an improved GCS approach. *Expert Systems with Applications*, 29(4), 757 – 767.
- Lin, M., Xie, J., Guo, H., & Wang, H. (2005). Solving qos-driven web service dynamic composition as fuzzy constraint satisfaction. In *Proc. IEEE International Conference on e-Technology, e-Commerce and e-Service*, pp. 9–14. IEEE.

- Linoff, G. S., & Berry, M. J. (2004). *Data Mining Techniques: For Marketing, Sales and Customer Relationship Management*. Wiley, Hoboken, New Jersey.
- Liu, L., & Shell, D. A. (2013). Optiml market-based multi-robot task allocation via strategic pricing. In *Proc. RSS*, Berlin, Germany.
- Lombrozo, T. (2006). The structure and function of explanations. *Trends in cognitive sciences*, 10(10), 464–470.
- Malhotra, N. K. (2010). *Marketing Research: an Applied Orientation*. Prentice Hall, Upper Saddle River, New Jersey.
- Michini, B., & How, J. P. (2012). Bayesian nonparametric inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, Vol. 7524 of *Lecture Notes in Computer Science*, pp. 148–163. Springer Berlin Heidelberg.
- Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3), 161–205.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Morris, P., Morris, R., Khatib, L., Ramakrishnan, S., & Bachmann, A. (2004). Strategies for global optimization of temporal preferences. In *Proc. International Conference on Principles and Practice of Constraint Programming*, pp. 408–422. Springer.
- Muscettola, N., Morris, P., & Tsamardinou, I. (1998). Reformulating temporal plans for efficient execution. In *Proc. KR&R*, Trento, Italy.
- Nunes, E., & Gini, M. (2015). Multi-robot auctions for allocation of tasks with temporal constraints. In *Proc. AAAI*, pp. 2110–2116.
- Odom, P., & Natarajan, S. (2015). Active advice seeking for inverse reinforcement learning. In *Proc. AAAI*, pp. 4186–4187.
- Öztürk, M., Tsouki, A., & Vincke, P. (2005). Preference modelling. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, Vol. 78 of *Proc. International Series in Operations Research & Management Science*, pp. 27–59. Springer New York.
- Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web.. Technical report 1999-66, Stanford InfoLab. SIDL-WP-1999-0120.
- Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., & Salakoski, T. (2007). Learning to rank with pairwise regularized least-squares. In *SIGIR Workshop on Learning to Rank for Information Retrieval*, pp. 27–33.
- Parasuraman, R., Sheridan, T., & Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *Trans. SMC-A*, 30(3), 286–297.
- Park, D. H., Kim, H. K., Choi, I. Y., & Kim, J. K. (2012). A literature review and classification of recommender systems research. *Expert Systems with Applications*, 39(11), 10059 – 10072.
- Peintner, B., & Pollack, M. E. (2004). Low-cost addition of preferences to dtps and tcsp. In *AAAI*, pp. 723–728.

- Pizer, S. D., & Prentice, J. C. (2011). What are the consequences of waiting for health care in the veteran population?. *Journal of General Internal Medicine*, 26(2), 676–682.
- Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
- Raghavan, H., Madani, O., & Jones, R. (2006). Active learning with feedback on features and instances. *Journal of Machine Learning Research*, 7, 1655–1686.
- Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. In *Proc. IJCAI*, pp. 2586–2591.
- Ramanujam, V., & Balakrishnan, H. (2011). Estimation of maximum-likelihood discrete-choice models of the runway configuration selection process. In *Proc. ACC*, pp. 2160–2167.
- Ross, S., Gordon, G., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635.
- Rossi, F., Venable, K. B., & Walsh, T. (2009). Preferences in constraint satisfaction and optimization. *AI magazine*, 29(4), 58.
- Rossi, F., Venable, K. B., & Yorke-Smith, N. (2006). Uncertainty in soft temporal constraint problems: A general framework and controllability algorithms for the fuzzy case. *Journal of Artificial Intelligence Research*, 27, 617–674.
- Rudová, H., & Murray, K. (2002). University course timetabling with soft constraints. In *International Conference on the Practice and Theory of Automated Timetabling*, pp. 310–328. Springer.
- Rybski, P. E., & Voyles, R. M. (1999). Interactive task training of a mobile robot through human gesture recognition. In *IEEE International Conference on Robotics and Automation*, pp. 664–669.
- Saaty, T. L. (2008). Relative measurement and its generalization in decision making why pairwise comparisons are central in mathematics for the measurement of intangible factors the analytic hierarchy/network process. *RACSAM-Revista de la Real Academia de Ciencias Exactas, Fisicas y Naturales. Serie A. Matematicas*, 102(2), 251–318.
- Sammur, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. In *Proc. ICML*, pp. 385–393.
- Sarwar, B. M., Karypis, G., Konstan, J. A., & Riedl, J. T. (2000). Application of dimensionality reduction in recommender system - a case study. In *Proc. ACM WEBKDD workshop*.
- Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In *The Adaptive Web*, pp. 291–324. Springer-Verlag, Berlin, Heidelberg.
- Schiex, T., Fargier, H., Verfaillie, G., et al. (1995). Valued constraint satisfaction problems: Hard and easy problems. *IJCAI (1)*, 95, 631–639.
- Schwarz, G., et al. (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2), 461–464.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.

- Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2), 254–265.
- Soomer, M., & Franx, G. (2008). Scheduling aircraft landing using airlines’ preferences. *European Journal of Operational Research*, 190, 277–291.
- Streeter, M. J., & Smith, S. F. (2006). How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research (JAIR)*, 26, 247–287.
- Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation.. In *Proc. NIPS*, pp. 1057–1063.
- Tan, K., Lee, L., Zhu, Q., & Ou, K. (2001). Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15(3), 281 – 295.
- Terrell, A., & Mutlu, B. (2012). A regression-based approach to modeling addressee backchannels. In *Proc. Special Interest Group on Discourse and Dialogue*, pp. 280–289.
- Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3), 58–68.
- Thomaz, A. L., & Breazeal, C. (2006). Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proc. AAAI*, pp. 1000–1005.
- Valizadegan, H., Jin, R., Zhang, R., & Mao, J. (2009). Learning to rank by optimizing NDCG measure. In *NIPS*, pp. 1883–1891.
- Vogel, A., Ramach, D., Gupta, R., & Raux, A. (2012). Improving hybrid vehicle fuel efficiency using inverse reinforcement learning. In *Proc. AAAI*, pp. 384–390.
- Volkovs, M. N., & Zemel, R. S. (2009). Boltzrank: Learning to maximize expected ranking gain. In *Proc. ICML*, pp. 1089–1096.
- Wang, Y.-C., & Usher, J. M. (2005). Application of reinforcement learning for agent-based production scheduling. *Eng. Appl. Artif. Intell.*, 18(1), 73–82.
- Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4), 279–292.
- Wickens, C. D., Li, H., Santamaria, A., Sebok, A., & Sarter, N. B. (2010). Stages and levels of automation: An integrated meta-analysis. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 54, pp. 389–393. SAGE Publications.
- Wilcox, R. J., & Shah, J. A. (2012). Optimization of multi-agent workflow for human-robot collaboration in assembly manufacturing. In *Proc. AIAA Infotech@Aerospace*.
- Wu, J., Xu, X., Zhang, P., & Liu, C. (2011). A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems*, 27(5), 430 – 439.
- Yorke-Smith, N., Venable, K. B., & Rossi, F. (2003). Temporal reasoning with preferences and uncertainty. In *IJCAI*, Vol. 3, pp. 1385–1386.
- Yu, S.-Z. (2010). Hidden semi-markov models. *Artificial Intelligence*, 174(2), 215 – 243. Special Review Issue.

- Zeng, Z., & Kuipers, B. (2016). Learning tabletop object manipulation by imitation. *arXiv preprint arXiv:1603.00964*.
- Zhang, F., & Burns, A. (2009). Schedulability analysis for real-time systems with edf scheduling. *IEEE Transactions on Computers*, 58(9), 1250–1258.
- Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proc. IJCAI*, pp. 1114–1120.
- Zheng, J., Liu, S., & Ni, L. (2015). Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proc. AAAI*, pp. 2198–2205.
- Ziebart, B. D., Maas, A., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proc. AAAI*, pp. 1433–1438.

Appendix A. ASMD: Mathematical Program Formulation

We readily formulate the ASMD as a mixed-integer linear program in Equations 23-44. This formulation incorporates a set of binary decision variables: $A_{d,m,t} \in \{0, 1\}$ is set to 1 to indicate that decoy d is assigned to missile m at time t , and is 0 otherwise. $A_{d,t} \in \{0, 1\}$ is set to 1 to indicate that decoy d is assigned to some missile at time t , and is 0 otherwise. $U_{d,m} \in \{0, 1\}$ is set to 1 to indicate that decoy d is used against missile m , and is 0 otherwise. $U_d \in \{0, 1\}$ is set to 1 to indicate that decoy d is used in the solution, and is 0 otherwise. $X_{d,l} \in \{0, 1\}$ is set to 1 to indicate that decoy d is deployed at location l , and is 0 otherwise. $V_m \in \{0, 1\}$ is set to 1 to indicate that missile m has been effectively diverted, and is 0 otherwise. $G_{g,m,t} \in \{0, 1\}$ is set to 1 to indicate that missile m is tracking the ship at time t . A single missile might have multiple, separate epochs during which it tracks the ship (e.g., it first tracks the ship, then a decoy, then the ship again after the decoy disappears); thus, the program can choose which index g to represent the various epochs in $G_{g,m,t}$. $J_{d,m} \in \{0, 1\}$ is set to 1 to indicate that decoy d is deployed after missile m 's flight (i.e., after it either hits the ship or is guided astray by a decoy).

The program contains the following continuous variables: $S_{d,m}^{decoy}$ represents the start time of the assignment of decoy d to missile m , and S_d^{decoy} is the time at which decoy d is deployed from the ship. Likewise, $F_{d,m}^{decoy}$ represents the finish time of the assignment of decoy d to missile m , and F_d^{decoy} is either the time at which the decoy disappears or the end of the engagement. $S_{g,m}^{ship}$ indicates the start time of missile m tracking the ship during epoch g , and $F_{g,m}^{ship}$ indicates the finish time of missile m tracking the ship during epoch g . We include the constant, M , which is a large, positive number allowing one to formulate linear, conditional constraints.

The program also includes the following set of constants: $dt_m^{re-target}$ is the length of time for which a missile will track a single target (i.e., a decoy or ship) before reassessing which target is best to track. Thus, if the missile begins tracking the ship at time t , no decoy can break its lock during the interval $[t, t + dt_m^{re-target})$. ETA_m is the time at which missile m will reach the ship's immediate vicinity. t_m^{appear} is the time at which missile m first becomes close enough to track the ship. c_d represents the financial cost of deploying decoy d . α , α' , and α'' are predefined weighting terms for the objective function. The computational complexity of completely searching for the optimal solution via this formulation is dominated by the integer variables, which yields $O(2^{dmt+dm+dt+dl+d+gmt+m})$.

Equation 23 is a multi-criteria objective function that minimizes a weighted linear combination of the cost of all decoy deployments, less the total time during which missiles are tracking decoys and the number of missiles successfully guided away from the ship. Equations 24-31 ensure internal consistency between the variables. Equation 32 ensures that a decoy, if deployed, is active for dt_d^{evap} units of time given its timing characteristics. Equation 33 ensures that a decoy is deployed to no more than one location. Equation 34 ensures that, if a decoy is deployed against a missile, its deployment location will be a more attractive target than the ship for that missile. Equation 35 requires that each missile track either a ship or decoy while within range. Equations 36-37 force a decoy, if deployed to a location that would cause missile m to impact the ship, to either be deployed after the missile has already been diverted or reached the ship (Equation 36) or to be deployed and disappear before the missile enters targeting range (Equation 37).

Equation 38 ensures that a missile must be tracking a decoy in the final seconds before it reaches the vicinity of the ship, or else the missile will impact the ship. The duration of this critical period is dependent upon missile dynamics and the target selection process. Equation 39 ensures that a

missile will select the most attractive decoy according to that missile's selection logic. Equation 40 restricts decoy deployments such that the missile heading does not "sweep" across the ship in the final seconds of the missile's flight. If a missile does not have enough time to change its direction toward a newly deployed decoy, that missile will fly into the ship.

Equations 41-44 ensure that the duration of epoch g of missile m while tracking the ship lasts exactly as long as the retargeting time for the missile. Equations 41-42 are akin to Equations 28-30 and relate the start and finish times of ship-tracking epoch g to the decision variable $G_{g,m,t}$. Equation 43 is akin to Equation 31 and relates the start and finish times of ship-tracking epoch g to the decision variable $G_{g,m,t}$. Equation 44 ensures that the tracking time is $dt_m^{re-target}$ if the missile is airborne for at least $dt_m^{re-target}$ seconds. Otherwise, the tracking time is equal to the time before impacting the ship (i.e., $ETA^m - t - 1$). Finally, a term (i.e., $-MG_{g,m,t-1}$) disables the constraint for all t except for the exact moment when t begins tracking the ship.

$$\min z, z = \alpha \sum_d c_d U_d - \alpha' \sum_{d,m,t} A_{d,m,t} - \alpha'' \sum_m V_m \quad (23)$$

$$A_{d,m,t} \leq A_{d,t}, \forall d, m, t \quad (24)$$

$$A_{d,m,t} \leq U_{d,m}, \forall d, m, t \quad (25)$$

$$X_{d,l} \leq U_d, \forall d, l \quad (26)$$

$$S_d^{decoy} \leq S_{d,m}^{decoy}, \forall d, m \quad (27)$$

$$S_{d,m}^{decoy} \leq t + M(1 - A_{d,m,t}), \forall d, m, t \quad (28)$$

$$F_{d,m}^{decoy} \leq F_d^{decoy}, \forall d, m \quad (29)$$

$$tA_{d,m,t} \leq F_{d,m}^{decoy}, \forall d, m, t \quad (30)$$

$$M(U_{d,m} - 1) \leq S_{d,m}^{decoy} - F_{d,m}^{decoy} - 1 + \sum_t A_{d,m,t} \leq M(1 - U_{d,m}) \quad (31)$$

$$M(U_d - 1) \leq F_d^{decoy} - S_d^{decoy} - dt_d^{evap} \leq M(1 - U_d) \quad (32)$$

$$\sum_l X_{d,l} \leq 1, \forall d \quad (33)$$

$$U_{d,m} \leq \sum_{l|m \text{ seduced by decoy } d \text{ in location } l} X_{d,l}, \forall d, m \quad (34)$$

$$1 = \sum_d A_{d,m,t} + \sum_g G_{g,m,t}, \forall m, t \quad (35)$$

$$t_m^{appear} - F_d^{decoy} \geq M(X_{d,l} + V_m - J_{d,m} - 2), \quad \forall d, l, m \text{ s.t. decoy } d \text{ in location } l \text{ would cause missile } m \text{ to impact the ship.} \quad (36)$$

$$S_d^{decoy} - ETA_m \geq M(X_{d,l} + V_m + J_{d,m} - 3), \forall d, l, m \text{ s.t. decoy } d \text{ in location } l \text{ would cause missile } m \text{ to impact the ship.} \quad (37)$$

$$V_m \leq \sum_d A_{d,m,t}, \forall m, t | t \text{ in critical region for missile } m. \quad (38)$$

$$2 \geq A_{d,m,t} + X_{d,l} + X_{d',l'}, \forall d, d', l, l', m, t \text{ s.t. missile } m \text{ is more attracted to decoy } d' \text{ at location } l' \text{ than decoy } d \text{ at location } l \text{ at time } t. \quad (39)$$

$$1 \geq A_{d,m,t} + A_{d',m,t}, \forall d, d', m, t \text{ s.t. } d \neq d' \text{ and } t \text{ is in a critical region before impact.} \quad (40)$$

$$S_{g,m}^{ship} \leq t + M(1 - G_{g,m,t}), \forall g, m, t \quad (41)$$

$$t * G_{g,m,t} \leq F_{g,m}^{ship}, \forall g, m, t \quad (42)$$

$$M(U_{g,m} - 1) \leq S_{g,m}^{ship} - F_{g,m}^{ship} - 1 + \sum_t G_{g,m,t} \leq M(1 - U_{g,m}) \quad (43)$$

$$\begin{aligned} F_{g,m}^{ship} - S_{g,m}^{ship} &\geq M(G_{g,m,t} - 1) \\ &+ \begin{cases} dt_m^{re-target} - 1 & \text{if } t < ETA_m - dt_m^{re-target}, \\ ETA_m - t - 1 & \text{otherwise.} \end{cases} \\ &+ \begin{cases} -MG_{g,m,t-1} & \text{if } t > t_m^{appear}, \\ 0 & \text{otherwise.} \end{cases} \\ &\forall g, m, t | t_m^{appear} \leq t < ETA_m \end{aligned} \quad (44)$$

As ASMD is a time-extended problem, the formulation must discretize time. However, note that the granularity with which the task of protecting the ship is decomposed as a function of time is a modeling choice with ramifications for the quality and computation time of a solution. Consider a missile that will hit the ship if it tracks the ship in some time interval $[t, t']$ for a duration $dt = t - t'$. The captain might, at time t , deploy a decoy d , such as a hovering UAV, that is able to last the entire duration dt . However, it may be preferable to deploy one or more decoys, d' , each of which remains active for a portion of the specified time interval. Furthermore, in a situation wherein another missile, m' , is launched before m , it may be best to have a decoy deployed before t that can divert both m and m' during part or all of those missiles' flights.

As we do not know a priori the best time to deploy a decoy that can be used for varying portions (i.e., subtasks) of the task of mitigating each missile, we must decompose the task into sufficiently small time steps. Discretizing time exponentially increases the search space, and thus the time to compute the solution; therefore, there is a balance between optimality (and feasibility) and computation time. In order to generate an exact solution, we chose the least-common multiple of the time constants, which is trivially 1, as the unit of time in the simulation.

Appendix B. Mitigating Anomalies

To mitigate anomalies inherent in a pairwise comparison approach, one could consider the following formulation in Equations 45 through 50 when learning a decision tree model, T^* , for apprenticeship scheduling:

$$T^* = \operatorname{argmin}_T \mathbb{E}_{\theta, y} [L(y_{\langle \tau_i, \tau_j \rangle}^m, T(\operatorname{rank} \theta_{\langle \tau_i, \tau_j \rangle}^m))] \quad (45)$$

subject to

$$T(\text{rank}_{\langle \tau_i, \tau_j \rangle} \theta_{\langle \tau_i, \tau_j \rangle}^m) > 0.5 + M(1 - Z_{i,j}), \forall \tau_i, \tau_j \quad (46)$$

$$T(\text{rank}_{\langle \tau_i, \tau_j \rangle} \theta_{\langle \tau_i, \tau_j \rangle}^m) < 0.5 + M(Z_{i,j}), \forall \tau_i, \tau_j \quad (47)$$

$$\sum_{\tau_k \in \mathcal{T}} T(\text{rank}_{\langle \tau_i, \tau_k \rangle} \theta_{\langle \tau_i, \tau_k \rangle}^m) - \sum_{\tau_k \in \mathcal{T}} T(\text{rank}_{\langle \tau_j, \tau_k \rangle} \theta_{\langle \tau_j, \tau_k \rangle}^m) > M(1 - Z_{i,j}), \forall \tau_i, \tau_j \quad (48)$$

$$\sum_{\tau_k \in \mathcal{T}} T(\text{rank}_{\langle \tau_i, \tau_k \rangle} \theta_{\langle \tau_i, \tau_k \rangle}^m) - \sum_{\tau_k \in \mathcal{T}} T(\text{rank}_{\langle \tau_j, \tau_k \rangle} \theta_{\langle \tau_j, \tau_k \rangle}^m) < M(Z_{i,j}), \forall \tau_i, \tau_j \quad (49)$$

$$Z_{i,j} + Z_{j,k} - 1 \geq Z_{i,k}, \forall \tau_i, \tau_j, \tau_k \quad (50)$$

Equation 45 states that we want to find the decision tree, T^* , among all possible trees, T , that minimizes an expected loss function, L . Recall from Section 3.2 that $y_{\langle \tau_i, \tau_j \rangle}^m$ is the binary label given to an observation to indicate whether the human demonstrator took action τ_i or τ_j . Further, $\text{rank}_{\langle \tau_i, \tau_j \rangle} \theta_{\langle \tau_i, \tau_j \rangle}^m$ is the corresponding feature vector from that observation. Equations 46 through 49 force the pairwise comparisons to agree with the cumulative ranking. $Z_{i,j}$ is a binary decision variable that is equal to 1 when τ_i is expected to be chosen over τ_j and 0 when τ_j is expected to be chosen over τ_i . Recall that M is a large positive number that allows one to formulate linear, conditional constraints. Finally, Equation 50 requires that the transitive property holds for T . Specifically, if τ_i is predicted to be more likely than τ_j (i.e., $Z_{i,j} = 1$), and τ_j (i.e., $Z_{j,k} = 1$) is more likely than τ_k , then τ_i should also be predicted to be more likely than τ_k (i.e., $Z_{i,k} = 1$).