Using Reward/Utility Based Impact Scores in Partitioning

Paper 597

ABSTRACT

Reinforcement learning with reward shaping is a natural approach to solving large multiagent domains with agents who must cooperate to achieve some system objective. However, reward shaping can be computationally expensive to compute. Agent partitioning can assist in this computational complexity by treating each partition of agents as an independent problem. In this paper we introduce a novel agent partitioning approach called Reward/Utility-Based Impact (RUBI). RUBI finds an effective partitioning of agents while requiring no prior knowledge about the domain, leads to better performance by discovering a non-trivial agent partitioning, and leads to faster simulations. We test RUBI in the Air Traffic Flow Management Problem (ATFMP), where there are simultaneously tens of thousands of aircraft affecting the system and no intuitively accurate similarity metric between agents. When partitioning with RUBI in the ATFMP, there is a 37% increase in performance, with a 510x speed up per simulation step over non-partitioning approaches. Additionally, RUBI matches the performance of the current ATFMP gold standard using no prior knowledge and performing each learning step 10% faster.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent Agents

General Terms

Algorithms, Performance, Experimentation

Keywords

Multiagent Partitioning, Multiagent Learning

1. INTRODUCTION

Two key elements in a multiagent reinforcement learning system is minimizing computation time and maximizing coordination. Reward shaping is a field in multiagent reinforcement learning that focuses on the design of rewards, and has been shown to assist in multiagent coordination. This reward shaping is typically at a large cost to computation time, and in large, highly coupled domains reward shaping quickly becomes computationally intractable.

Modeling the reward shaping technique [13] in relatively large domains (approx. 400 agents) can be applied to great effect, but requires tens of thousands of randomly generated examples to obtain these approximations. On the other hand, partitioning agents into hierarchies [12] or teams [6] speeds up computation time for extremely large domains (approx. 10000-40000 agents) while still using the pure reward shaping technique. While creating these hierarchies or teams, a fundamental understanding of how agents are coupled must be employed by the algorithm designer. In complex domains where the amount of impact an agent has on another is unknown, or in a situation where the algorithm designer has no domain knowledge, these approaches cannot be used.

In this paper we introduce Reward/Utility Based Impact (RUBI) scores. RUBI learns an effective partitioning of agents, while requiring *no domain knowledge*. It asks the question: What is each agent's local reward with agent a and without agent a? The difference in these two rewards is the impact on one agent to agent a. This impact is computationally inexpensive to calculate and can be used as an accurate similarity metric that can be used in an off the shelf clustering algorithm.

We test RUBI in the Air Traffic Flow Management Problem (ATFMP) and use the approach developed by Curran et al. [6] and Agogino and Rios [3, 15]. In this domain the goal is to minimize both congestion and delay associated with the air traffic in the United States. Because the airspace has many connections from one airport to another, the congestion and associated delay can propagate throughout the system. Delays may be imposed to better coordinate aircraft and mitigate the propagation of congestion and the associated delay, but which aircraft should be delayed? The search space in such a problem is huge, as there are tens of thousands of flights every day within the United States [7].

In the ATFMP, multiagent coordination with reward shaping becomes a computationally intractable task, therefore related work has applied automated agent partitioning to reduce the overhead associated with the hard constraint while computing rewards [3, 6, 15]. In that work, agents were only required to compute the reward relative to other agents within their partition, removing thousands of extra computations per learning step. We employ this approach, but use RUBI-based partitioning rather than the related work domain-based partitioning.

Appears in: *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), Lomuscio, Scerri, Bazzan, Huhns (eds.), May, 5–9, 2014, Paris, France.*

The contributions of this work are:

- Generality: RUBI requires no prior knowledge of the domain, essentially treating the domain as a black box to obtain rewards from.
- Ease-of-use: RUBI removes the need to derive similarity metrics, removing the need for domain experts in situations where a domain expert isn't available.
- Performance: RUBI discovers non-trivial agent partitioning through the concept of using a reward function to partition agents.
- Speed: RUBI leads to a larger number of partitions without losing performance, leading to more independence and therefore faster simulations.

The remainder of this paper is organized as follows. Section 2 describes the related work in agent partitioning, the ATFMP, multiagent coordination and reward shaping. In Section 3 we describe the experimental approach taken using multiagent coordination, the difference reward, the greedy scheduler and agent partitioning. Section 4 contains the key contribution of this work, RUBI, and describes the basic algorithm, defines reward-based impact and when to use it, and the key benefits of RUBI. Experimental results are then provided in Section 5, followed by the conclusion and future work in Section 6.

2. BACKGROUND

To motivate our approach, we introduce previous work performed in the field of agent partitioning, a description and overview of the ATFMP, and describe the reward shaping technique used in this work.

2.1 Agent Partitioning

Previous work in agent partitioning has focused on what specifically to partition. Jordan and Jacobs [10] developed the Hierarchical Mixtures of Experts (HME) method to partition the state space directly, such that different agents can focus on specific regions of the state space. This method works well in non-linear supervised learning tasks. In this work all learning is unsupervised, so this technique cannot be used.

Partitioning actions so that each agent is responsible for a small number of actions is another approach used often. Sun and Pearson [16] divided actions into two types, speed and turn, and each type was handled by a separate agent. This approach uses domain knowledge, and partitioning actions applies well in robotics, but not in domains where all actions need to be explored.

The last partitioning technique we describe is to partitioning system-level goals into smaller tasks. In the work by Dayan and Hinton [8], they accomplished goal partitioning through task allocation, where agents are organized in a hierarchy, where high-level agents assign goals to agents lower in the hierarchy on-line. In the work by Reddy and Tadepalli [14], the approach is more structured. In this work the partitioning of the goal is learned through externally provided examples. Overall, these approaches are under the assumption that the system-level goal can be subdivided, which is not always the case.

In this work, we partition agents, essentially treating each partition of agents as an independent problem. Agents from one partition could potentially affect the environment of agents in another partition, but this work attempts to minimize the partition overlap. In a partitioning with complete reward independence, this work essentially treats the problem as a set of smaller and easier independent problems.

2.2 Air Traffic Flow Management Problem

The ATFMP addresses the congestion in the National Airspace (NAS) by controlling ground delay, en route speed or changing separation between aircraft. The NAS is divided into many sectors, each with a restriction on the number of aircraft that may fly through it at a given time. This number is formulated by the FAA and is calculated from the number of air traffic controllers in the sector, the weather, and the geographic location. These sector capacities are known as en route capacities. Additionally, each airport in the NAS has an arrival and departure capacity that cannot be exceeded. Eliminating the congestion in the system while keeping the amount of delay each aircraft incurs small is the fundamental goal of ATFMP.

In this work we chose to use an aggregate model of the NAS. By choosing to control the ground delay of each aircraft, rather than the enroute speed, and using historical NAS flight data, we only need to count how many aircraft are within a particular sector at a particular time. Actions affect the simulation minimally, as an aircraft with imposed ground delay simply needs to shift its entire flight plan by the amount of ground delay, greatly speeding up simulation time.

The approach we use in the ATFMP follows the same approach by Curran et al. [6] and Agogino and Rios [3, 15]. Rather than treating congestion as a soft constraint, these approaches removed congestion completely from the system algorithmically through the use of a greedy scheduler. A high level view of this approach is as follows: First, they computed partitions of agents using a domain-based similarity metric of sector overlap and hierarchical agglomerative clustering. These partitions are treated as reward independent of each other, and therefore need to only compute rewards relative to the other agents within their partition. We will use the term *reward independent* to denote one partition of agents to have no impact on the reward of other partitions. Essentially, no matter what actions one partition of agents take, it will not affect the the action choice for any agent in another partition. They then performed multiagent reinforcement learning using the difference reward. Lastly, they introduced a greedy scheduler, removing all congestion from the system. They found that combining the multiagent reinforcement learning with reward shaping and the greedy scheduler turns this into a computationally intractable task. They solved this problem with agent partitioning, showing that rewards can be computed many times faster with minimal performance degradation.

2.3 Reward Shaping

Multiagent coordination is an important aspect of many domains, such as data routing [17], air traffic control [2], Robocup soccer [1], rover coordination [11] and power plant operations [5]. A learning or evolutionary algorithm will often convert a once computationally intractable search problem into a feasible guided search.

In learning algorithms, reward design is important in keeping convergence time low while keeping performance high.

In many multiagent coordination domains there is a difference between maximizing the system-level reward and maximizing a single agent's reward. If an agent always takes the locally-optimal action, it does not always maximize the system-level reward; a common example of this is the Tragedy of the Commons [9].

The difference reward [17] is evaluated such that each agent's reward is related to the individual's contribution to team performance, therefore the signal-to-noise ratio is improved considerably. This leads to final better policies at an accelerated convergence rate, as well as overcoming the Tragedy of the Commons. The difference reward is defined as:

$$D_i(z) = G(z) - G(z - z_i + c_i) \ , \tag{1}$$

where z is the system state, z_i is the system state with agent i, and c_i is a counterfactual replacing agent i. This counterfactual offsets the artificial impact of removing an agent from the system. For example, removing an aircraft from the system always artificially decreases the amount of congestion and delay, which would provide a poor reward if a counterfactual is not used.

The difference reward provides a compact encapsulation of the agent's impact on the system. It reduces the noise from other agents in the reward signal and has outperformed both system-level and local rewards in many congestion domains [1, 4, 5, 17]. In many systems it is difficult or impossible to calculate the difference reward without resimulation, which can become prohibitively costly. If resimulation is fast, or the difference reward function is easily approximated, this reward function is a powerful tool for multiagent coordination.

3. ATFMP APPROACH

In this section we give a brief overview of the learning approach used in this paper. This application approach is not a key focus of this work and was developed by Curran et al. [6] and Agogino and Rios [3, 15], and the reader is referred to this related work if they have any additional questions.

3.1 Agent Definition

In this paper, agents are assigned to one of 35,844 aircraft with cooperation enforced by airport terminals. Cooperation needs to be enforced because aircraft are naturally greedy. Aircraft are owned by different companies, and those companies are not interested in making sure aircraft from other companies arrive on time. This enforced cooperation assumption by aircraft terminals (or the government) is essential to remove greedy aircraft from the system.

Aircraft flight plans are from historical flight data from the FAA. Therefore, the only aspect of the environment we can change is the ground delay for each aircraft. Agents may select a certain amount of ground delay from 0 to 10 minutes (11 actions) in the beginning of every simulation. The FAA data has the sector location of each plane for every minute that plane was in service. Therefore, adding ground delay simply shifts a plane's flight plan over by that many minutes. The greedy scheduler then takes all plane flight plans, checks if the flight plans cause any congestion, and then further delays planes to eliminate congestion from the system.

In this formulation, agents do not have the capability to change their action based upon the system once the simulation starts, therefore feedback can only be given once per simulation. Since agents are given no knowledge of the environment, they have no state. This simplifies the learning problem for each agent, as they only have eleven actions to sample from, but complicates the coordination. Agents must choose an action without prior knowledge of other agents choices, and must learn how the environment is changing, and simultaneously what action to take.

Agents learned using Action-Value Learning with a zero initialized value table. This is a stateless approach where agents map actions to values representing the quality of that action.

3.2 Reward Structures

In this section we first develop the system-level reward. This reward represents how well the system as a whole is performing. We want this value to be as high as possible. We then develop the difference reward from the system-level reward. The difference reward represents how much a particular agent contributes to the system-level reward. Agents should be rewarded with the difference reward, and system performance should be measured as the system-level reward.

The system-level reward in the ATFMP developed focuses on the cumulative delay (δ) and congestion (C) throughout the system:

$$G(z) = -(C(z) + \delta(z)) \ , \tag{2}$$

where $C(z)$ is the total congestion penalty, and $\delta(z)$ is the cumulative system delay.

The total congestion penalty is the sum of differences between sector capacity and the current sector congestion. The total delay is the sum of delays over all aircraft. This is a linear combination of the amount of ground delay and the amount of scheduler delay an agent incurred.

Agogino and Rios originally had the idea of adding a greedy scheduler to algorithmically remove congestion from the system, while simultaneously using learning to minimize delay. We follow this approach, and therefore our system-level reward is simply:

$$G(z) = -\delta(z) \tag{3}$$

With so many agents, tens of thousands of actions simultaneously impact the system, causing the reward for a specific agent to become noisy with the actions of other agents. An agent would have difficulty learning an optimal solution using such a noisy reward signal. A difference reward function reduces much of this noise, and is easily derived from the system-level reward:

$$D_i(z) = \delta(z - z_i + c_i) - \delta(z) \ , \tag{4}$$

where $\delta(z - z_i + c_i)$ is the cumulative delay of all agents with agent i replaced with counterfactual c_i.

We used a non-zero counterfactual for two reasons. One, when an aircraft is removed from the system, the reward given to the agents is artificially increased. Furthermore, the aircraft are less likely to cause conflicts due to the easier scheduling problem. Two, we used a counterfactual where the agent does not delay at all, meaning that if not delaying produces a higher reward than delaying, this should be found quickly. We also take advantage of the fact that most airplanes do not need to be delayed. If the counterfactual is equal to the agent removed, the difference reward becomes

zero and does not need to be calculated, thus speeding up reward calculations.

4. RUBI

In this section we will describe in detail the Reward/Utility-Based Impact (RUBI) algorithm. We will first describe a general overview of RUBI, and the implementation. We will then go over the variety of ways RUBI impact scores and simulations can be developed. Lastly we will give an overview of the benefits of using RUBI.

4.1 RUBI Basics

In this work we introduce an autonomous partitioning algorithm requiring no domain knowledge, the Reward/Utility Based Impact algorithm. Domain-based partitioning directly looks at aircraft flight plans and partitions agents together based on how similar their flight plans are. Instead, we develop an initial agent similarity matrix that uses no knowledge about the domain, and partitions agents together based on the impact of one agent to another. This matrix can then be used as an input to a hierarchical agglomerative clustering algorithm. Additionally, by removing all knowledge about the domain and partitioning based on reward, RUBI can be used to discover non-trivial indirect interactions encoded in a reward signal.

On a basic level, this algorithm leverages the same central idea behind difference rewards: If an agent is removed from the system, how does that affect system-level reward? We modify this approach to answer the question: If an agent is removed from the system, how does that impact other agents? If one agent's action heavily impacts another agent's reward (positively or negatively), those agents are coupled enough to be partitioned together. The RUBI algorithm computes a localized reward for each agent with agent i in the system, and then compares that reward to the localized reward for each agent if agent i is not in the system. This partitioning algorithm is based around the central idea:

$$|L_i(z) - L_i(z - z_j)| > |L_k(z) - L_k(z - z_j)| \Rightarrow \quad (5)$$
$$SIM(i,j) > SIM(k,j)$$

where $L_i(z - z_j)$ is the localized reward of agent i if j is not in the system, $L_k(z - z_j)$ is the localized reward of agent k if j is not in the system, and L_i and L_k are the localized rewards of i and k when all agents are in the system. This means that if the localized reward of agent i changes more than the localized reward of agent k when agent j is taken out of the system, agent j has more effect on agent i than agent k. This is the essential idea behind RUBI, and is encoded in this algorithm through the equation:

$$C_{r,a} \leftarrow C_{r,a} + |L_a(z) - L_a(z - z_r)| \,, \quad (6)$$

where $L_a(z)$ is the reward agent a receives when all agents are in the system, $L_a(z - z_r)$ is the reward agent a receives with agent r not in the system, and $C_{r,a}$ is the cumulative impact agent r has on agent a.

4.2 Implementation of RUBI

The RUBI algorithm (Algorithm 1) first initializes an N x N matrix C, where N is the number of agents within the system. It then calculates actions based on the $ACT()$ function, which is typically random action selection. A simulation is then ran with all of the agents in the system and the localized reward is calculated for every agent. We then remove an agent from the system, recalculate the reward for each agent (since this is a localized reward, this is typically a fast operation), and update the impact table based on equation 6. This is a high level understanding of RUBI, and the following sections will explain how the impact data is computed, simulation specifics, the $ACT()$ function, and finally when this partitioning algorithm is practical.

Algorithm 1 Reward/Utility Based Impact Algorithm

```
1: function RUBI(sim)
2:     C ← NxN
3:     for i ← 1 to iterations do
4:         actions ← ACT()
5:         sim.run(actions)
6:         L(z) ← sim.getRewards()
7:         for r ← 1 to N do
8:             sim.removeAgent(r)
9:             L(z − z_r) ← sim.getRewards()
10:            for a ← 1 to N do
11:                C_{r,a} ← C_{r,a} + |L_a(z) − L_a(z − z_r)|
12:            end for
13:            sim.addAgent(r)
14:        end for
15:    end for
16: end function
```

4.3 Reward/Utility based impact

The impact data used to compute the similarity matrix is obtained from a localized reward or utility with respect to an agent. Learning in congestion problems using local rewards typically lead to a terrible solution, as local rewards correspond to greedy agents. Congestion problems usually need a few agents to receive negative rewards in order for the rest of the agents to receive positive rewards. Greedy agents do not take this into account. In RUBI, we do not want to learn, but instead analyze the local impact one agent has on another, therefore local rewards are an ideal choice.

In this work we apply RUBI only to reinforcement learning, therefore all impact scores are based on reward, but the work applies to algorithms that use utilities. A traditional local reward can work, or a localized reward can be specifically built for the partitioning. A high-level reward, such as the system-level reward, will not work as impact data. When computing the difference in the global reward with all agents and the global reward without a specific agent, we compute the difference reward. The difference reward represents how much an agent impacts the system, but the goal here is to find how much one agent impacts another agent.

Equation 6 is simply an accumulation of impact scores. Given enough iterations, this accumulation is informative enough to perform accurate partitioning. In this research we are interested more in the relative impact score from one agent to another, rather than what the explicit impact score is. This iterative approach requires at minimum enough iterations to evenly distribute over all actions an agent can take. Ideally each action should be sampled many times for a accurate impact estimate. Future work in RUBI involves approximating the impact score of each agent by adding a learning rate and subtracting off previous impact scores

per iteration. This causes $C_{r,a}$ to converge to an impact score, and analyzing this impact score may be useful in team formation or domain analysis.

For the ATFMP we developed a partitioning reward by both simplifying the simulation and adding to the reward. At a high level we want to encapsulate how one agent affects another in the reward. We took out the greedy scheduler and used the congestion as added information for the similarity data:

$$R = -(C(z) + \delta(z)) \,, \tag{7}$$

where *C(z)* is the congestion penalty and $\delta(z)$ is the delay as defined in section 3.2. This is a perfect example of how different the simulation can be during partitioning than during learning.

4.4 Simulation

The simulation is also an aspect that can be widely varied when using RUBI. One example already explored is the ATFMP reward-based impact data, where we change the simulation by removing the greedy scheduler, and taking advantage of the congestion information. This concept can be expanded upon by borrowing some of the concepts from transfer learning.

Transfer learning is a traditional approach used in both classification and learning to apply what is classified or learned from a smaller and easier domain to a larger more complicated domain. One subset of transfer learning is transfer clustering. Given a proper mapping, clusters learned in a small simulation can be applied to a larger simulation [18]. We can apply the same concepts developed in transfer clustering to this RUBI simulation. We've shown the simulation used during partitioning doesn't necessarily have to be the same as the simulation used during learning. It's also intuitive that any simulation can be used during partitioning, as long as there is a mapping from the RUBI simulation to the learning simulation. This approach is very beneficial in domains where the simulation is costly and a mapping can be discovered.

The $ACT()$ function returns a list of agent actions to use in the RUBI simulation. This is yet another aspect of this algorithm that can be chosen by the user. In the domains with no failure modes, this function typically returned random actions. The fundamental goal of $ACT()$ is to have as much of the interactive state space explored as possible, but we cannot exhaustively search the entire state space, as that would be both impractical and computationally impossible. For this reason we choose to take random actions. When random actions are taken, agents are not driven by any motivating logic, and impact scores will be biased only toward agents who consistently impact each other.

There are many domains, such as robotics, where random actions may cause certain failure modes, or where agents need to be in a particular area of the state space before interactions can really be analyzed. In these domains the $ACT()$ function can either sub-sample from a set of known non-failure mode actions, or be replaced with an actual learning algorithm to get the agent in a successful area of the state space before sampling.

4.5 Benefits of RUBI

One of the key strengths of RUBI is its sheer simplicity and generality combined with computing highly informative similarity scores, leading to well-performing partitions, as described in the results section. It needs no prior knowledge about the domain to perform partitioning, and simply needs a localized reward from each agent to build the similarity matrix. This makes RUBI highly generic and can be applied to any multiagent domain. It can treat the multiagent system as a black box, giving it random actions and receiving rewards. It can also discover non-trivial agent coupling.

Since RUBI uses a localized reward as partitioning data, any effect one agent has on another agent will be encoded in this reward. For example, if an agent a is removed from the system, and agent b's reward changes, it means that in some way agent a affects agent b in a direct or indirect way. This indirect affect can be captured by RUBI and used as additional information when partitioning, leading to higher quality partitions in domains with complex interactions.

One of the key benefits of RUBI is that partitions built using RUBI are likely to be greater in number without loss of performance. The ATFMP is a perfect example. Domain-based partitioning based on similar sectors encodes how often two aircraft can impact each other. RUBI on the other hand looks more into how the actions of one agent impact another agents reward, in this case congestion. For example, plane a and plane b go through the same sectors, but are never congested. Using domain-based partitioning, two agents that go through the same sector many times would be partitioned together, so plane a and plane b would be partitioned together. In partitioning using RUBI, if over a few thousand trials the congestion of each plane is always 0, those planes actions never impact each others rewards, therefore they would not be partitioned together. The same is true if the congestion of each plane remains the same non-zero value, the actions do not affect the reward, therefore they are not partitioned. This is a very important key feature of RUBI, and leads partitioning using RUBI to find more partitions without loss of performance. It is a perfect example of RUBI finding a non-trivial partitioning.

5. RESULTS

During preliminary analysis, RUBI works as expected in simple toy problems, but what happens when we apply it in a complex domain with tens of thousands of agents, such as the ATFMP? When RUBI is applied during partitioning in the ATFMP, simulation time decreases and the ease of application raises over developing a similarity metric. The removal of domain knowledge allows the same RUBI algorithm to be used in simple problems as well as the ATFMP with no effort and without any need to develop a similarity metric. The approach used here is the same as in Curran et al. [6] and Agogino and Rios [3, 15], except utilizing RUBI.

Partitions developed using RUBI uses similarity metrics that encapsulated the agent coupling. In this section we will first show how RUBI partitioning works well in the ATFMP, and analyze the cost/benefit of varying the number of partitions. We will then compare RUBI-based partitioning to domain-based partitioning and see that RUBI develops better quality partitions as well as partitions that lead to faster simulation times.

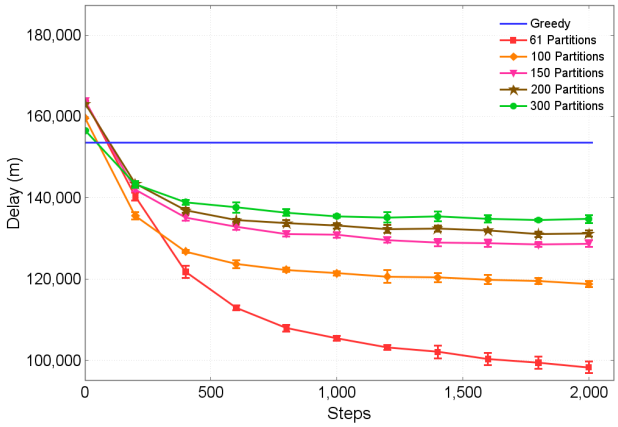

Figure 1: As the number of partitions decreases, performance improves while time complexity increases. Note that a reward independent partitioning using RUBI includes 61 partitions.

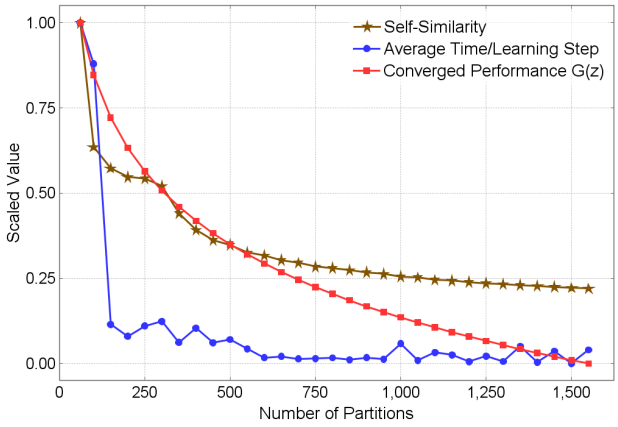

Figure 2: As the self-similarity increases, final performance and time taken per learning step increases. Note that final performance is a 6th degree polynomial trend line with $R^2 = .95$, and all values are scaled between 0 and 1 for comparison purposes.

5.1 RUBI Performance in the ATFMP

When partitioning agents using RUBI in the ATFMP agents took random actions, the greedy scheduler was not used, and the localized reward for each agent involved both congestion and delay (Equation 7). The reward-based impact function included both delay and congestion, since the greedy scheduler was removed, so the difference in delay was constantly 0 when computing reward-based impact. Therefore, agents were partitioned together based on whether their actions cause congestion to other agents.

Partitioning with RUBI and the difference reward outperformed the greedy scheduler. Figure 1 shows a variety of partitions out performing the greedy scheduler. The final performance of the ATFMP using RUBI-based partitioning was similar to domain-based partitioning performance. This is because at a converged partitioning, all agents are considered reward independent. The key benefit of RUBI-based partitioning was that a reward independent partition involved 61 partitions, but in domain-based partitioning the smallest was 3. This leads to faster processing time, at no cost to performance. This is discussed in further detail in the next section.

To understand how much overlap partitions had with each other, we analyzed the similarity each partition had with itself, and the average similarity each partition had with other partitions. Similarity was defined as the similarity metric used in domain-based partitioning, the number of similar sectors between agents.

In this section, we find the number of sectors that are similar between planes in a partition (self-similarity). We then find the the average number of sectors each plane in one partition has in common with each plane in another partition, and average all of these together to obtain a other-similarity metric. When graphing these metrics we take the percentage of self-similarity to other-similarity, and vice-versa. In a purely reward independent partition, the other-similarity is 0, and the self-similarity is 1. As the amount of self similarity increases though, the computation time increases exponentially, and the benefits of partitioning decreases.

The key difficulty in performing partition in the ATFMP, where almost every agent is coupled, is to partition agents in such a way that the similarities between partitions are as small as possible, while still preserving the computational benefits of partitioning. For this reason, we need to analyze both the performance of a variety of partition sizes, as well as the cost associated with the partition sizes.

When partitioning in a multiagent system, unless partitions are reward independent, there is always a cost and a benefit. The benefit is faster simulation time and/or reward calculations at the cost to performance. Although there is always a cost to performance, there is typically a point where the benefit of faster computation offsets the cost to performance. When partitioning with RUBI, with the slowest simulation there is a 37% increase in performance over the greedy scheduler, with a 510x speed up *per learning step* over non-partitioning approaches, and with a larger number of partitions we obtained 5400x speed up *per learning step* with a 20% increase in performance.

In the ATFMP, this partitioning benefit speeds up computation by magnitudes, but how much performance are we losing, and at what point is the partitioning costing too much performance for the benefit of faster computation? Figure 2 compares average computation time, final performance and self-similarity of each partition. The average computation time per learning step for each partition size remains static, and then exponentially increases once the self-similarity becomes large. This is the cost of each partition. There is also has a sharp increase in final performance as the self-similarity increases. This is the benefit of each partition. Both of these performance metrics are correlated greatly with the self-similarity metric.

5.2 Comparison Between RUBI and Domain-Based Partitioning

The comparisons made in this section are made with respect to the similarity metric chosen by Curran et al. [6] and Agogino and Rios [3, 15], the number of overlapping sectors. One needs to keep in mind that if a group of domain experts in air traffic control worked together to develop a similarity metric, partitioning using that metric is likely

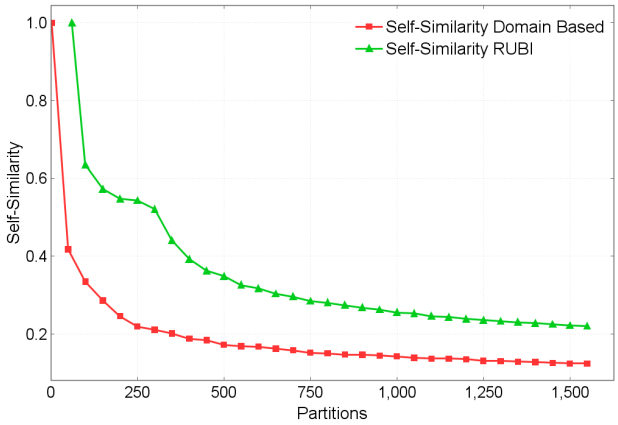

Figure 3: Partitions formed with RUBI had higher self-similarity than using domain-based partitioning. This leads to higher quality learning with respect to each partition.

to be extremely accurate. RUBI-based partitioning benefits from ease-of-use and generality. With RUBI a very well performing partitioning can be computed with little or no effort.

Partitioning with RUBI developed better similarity scores than domain-based partitioning (Figure 3). By partitioning using congestion, the similarity metric was able to represent both similar sectors as well as sector congestion. Consider the case where two aircraft have the same flight plan, excluding their congested arrival and departure location. These aircraft would be partitioned together when using only similar sector domain knowledge. With RUBI, those aircraft would be partitioned together with aircraft that cause them congestion. By partitioning using the impact one agent had on another agents reward, we are able to formulate higher quality partitions, without the overhead of developing similarity metrics or learning domain knowledge.

In addition to higher self-similarity, the partitioning using RUBI converged to a reward independent partitioning that included many more partitions than domain-based partitioning. In a reward independent partitioning, more partitions reduce the computation while incurring no loss of performance. The reward independent domain-based partition included 3 partitions, while partitioning with RUBI included 61 partitions. This is due to using reward-based impact as a similarity metric, as mentioned in Section 4.3. Two agent's actions may greatly affect each other during simulation, but their reward-based impact on each other might still be zero. This is a very important fact to keep in mind during partition analysis. For example, if two agents go through the same sectors, but neither agent causes another agent more or less congestion, then the difference in local reward will be zero, even though those aircraft affect each other. This leads to more partitions, and faster simulations. This is an example of RUBI finding a non-trivial partitioning.

When comparing final performance, a direct comparison of performance is not adequate. This is because partitions are not evenly distributed, which causes extremely large bias when comparing final performance to number of partitions. To show this in detail, we directly compared performance. We saw that with a larger number of partitions domain-based partitioning initially out performs RUBI, but RUBI began out performing domain-based partitioning with a smaller number of partitions.

In looking deeper into the reasoning behind this, we realize that partitions are not evenly distributed. Initially, domain-based partitioning have very large partitions compared to RUBI partitioning. This is due to the accumulation of similar sectors in domain-based partitioning. In domain-based partitioning, all of the planes with high overlap are initially merged together, giving domain-based partitioning initially larger partitions. Since this is a highly-coupled domain, this also leads to one partition becoming much larger than others. This is not seen in the self-similarity due to self-similarity being an average of all self-similarities. Basically, domain-based partitioning in this domain results in one partition having very high self-similarity, while other partitions get ignored. Although performance is initially better in domain-based partitioning, it is at 800+ partitions, meaning that the performance at this level of partitioning is already low for both partitioning approaches, and is therefore less useful.

RUBI, which partitions based on the reward of congestion and delay, initially merged together partitions whose reward highly impacted each other, which creates more evenly distributed partitions, leading to a better overall similarity, but initially worse performance. With a smaller number of partitions, RUBI-based partitions end up becoming larger on average. This in turn led to a bias for RUBI partitioning. Note that when all partitions are reward independent, RUBI partitioning included a much smaller average size, and many more partitions.

This results in us comparing final performance to the average size of the top 5% of partitions. This gives a comparison of how well a partitioning performs with respect to the size of it's partitions, rather than the number. Figure 4 shows that initially in domain-based partitioning, learning performance does not increase with respect to the average size of the partitions. RUBI on the other hand creates a partitioning with constantly increasing performance. One key point of this figure is that RUBI-based partitioning developed a better partitioning with every additional partition merge. This is because RUBI uses rewards to develop an impact score, so when two partitions are merged together it is guaranteed that performance will increase. Domain-based partitioning on the other hand had a steady performance until a certain point, and then performance increased dramatically. Note how many points are sampled for domain-based partitioning in Figure 4 for the smaller sized partitions.

6. CONCLUSION

This paper introduces RUBI, a partitioning algorithm that computes reward-based impacts that can then be used to partition agents together, removing the need for prior knowledge of the system. This method also removes the need develop similarity metrics derived from expert domain knowledge. Additionally, by removing all knowledge about the domain and partitioning based on reward, RUBI can be used to discover non-trivial indirect interactions encoded in a reward signal. Since RUBI uses only a reward signal to compute impacts, it will theoretically work in any domain where partitioning is useful.

In this work, we showed that partitioning with RUBI accurately encapsulated the amount of coupling between agents,

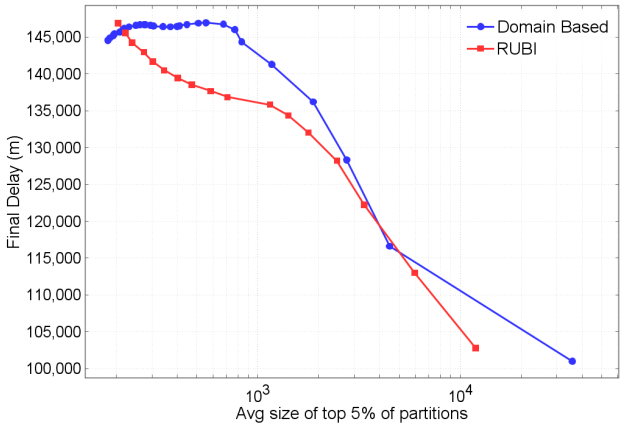

Figure 4: When comparing individual partition size averages to performance, RUBI partitions perform much better than domain specific partitions with respect to average partition size and final performance.

leading to a higher self-similarity metric over the domain-based partitioning, leading to faster simulation computation times. Learning using partitions developed with RUBI also found a 37% increase in performance over the greedy solution with a 510x reduction in time complexity per learning step compared to the 450x speed up using domain-based partitioning. RUBI-based partitioning is able to achieve the same performance 10% faster. This reduction in simulation cost was due to partitioning with RUBI leading to a larger number of smaller sized reward independent partitions.

Future work in RUBI would involve applying it to domains where coupling is very difficult to predict. We expect that RUBI would work fine in such a domain, but more exploration is needed. Performing a formal analysis of the relation between the number of iterations of RUBI and partition performance is important for future work, as we currently do not have a formal stop criteria. Approximating the impact score of each agent, rather than using a accumulation has the potential of being very informative when performing an analysis of a system. Future work in performing a simple approximation of the local reward function for each agent will greatly speed up RUBI computation time. Lastly, performing distributed clustering would be an important, yet simple extension to this work. Agents need only to compute the difference in local rewards if they come into contact with another agent, and then partition using this trimmed down similarity matrix.

7. REFERENCES

[1] Noa Agmon and Peter Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, June 2012.

[2] A. K. Agogino and K. Tumer. A multiagent approach to managing air traffic flow. *Autonomous Agents and MultiAgent Systems*, 24:1–25, 2012.

[3] Adrian Agogino. Evaluating evolution and monte carlo for controlling air traffic flow. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 1957–1962, New York, NY, USA, 2009. ACM.

[4] Adrian Agogino, Chris HolmesParker, and Kagan Tumer. Evolving large scale uav communication system. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, 2012.

[5] Mitchell Colby and Kagan Tumer. Shaping fitness functions for coevolving cooperative multiagent systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[6] William J. Curran, Adrian Agogino, and Kagan Tumer. Addressing hard constraints in the air traffic problem through partitioning and difference rewards. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013.

[7] FAA OPSNET data Jan-Dec 2011. US Department of Transportation website. (http://www.faa.gov/data_statistics/), 2011.

[8] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, pages 271–278. Morgan Kaufmann, 1993.

[9] G. Hardin. The tragedy of the commons. *Science*, 162:1243âĂŞ1248, December 1968.

[10] M.I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. In *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, volume 2, pages 1339–1344 vol.2, 1993.

[11] G.A. Kaminka, D. Erusalimchik, and S. Kraus. Adaptive multi-robot coordination: A game-theoretic perspective. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010.

[12] C. Holmes Parker and K. Tumer. Combining difference rewards and hierarchies for scaling to large multiagent system. In *AAMAS-2012 Workshop on Adaptive and Learning Agents*. Valencia, Spain, June 2012.

[13] Scott Proper and Kagan Tumer. Modeling difference rewards for multiagent learning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, 2012.

[14] Chandra Reddy and Prasad Tadepalli. Learning goal-decomposition rules using exercises. In *In Proceedings of the 14th International Conference on Machine Learning*, pages 278–286. Morgan Kaufmann.

[15] J. Rios and J. Lohn. A comparison of optimization approaches for nationwide traffic flow management. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference, Chicago, Illinois*, August 2009.

[16] Ron Sun and Todd Peterson. Some experiments with a hybrid model for learning sequential decision making. *Information Sciences*, 111:83–107, 1998.

[17] D. H. Wolpert and K. Tumer. Collective intelligence, data routing and Braess' paradox. *Journal of Artificial Intelligence Research*, 16:359–387, 2002.

[18] Litao Yu, Yanzhong Dang, and Guangfei Yang. Transfer clustering via constraints generated from topics. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, 2012.