Agent Partitioning with Reward/Utility-Based Impact

William Curran
Oregon State University
Corvallis, Oregon
curranw@onid.orst.edu

Adrian Agogino
NASA AMES Research Center
Moffet Field, California
adrian.k.agogino@nasa.gov

Kagan Tumer
Oregon State University
Corvallis, Oregon
kagan.tumer@oregonstate.edu

Abstract

Reinforcement learning with reward shaping is a well established but often computationally expensive approach to large multiagent systems. Agent partitioning can reduce this computational complexity by treating each partition of agents as an independent problem. We introduce a novel agent partitioning approach called Reward/Utility-Based Impact (RUBI). RUBI finds an effective partitioning of agents while requiring no prior domain knowledge, improves 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 tens of thousands of aircraft affecting the system and no obvious similarity metric between agents. When partitioning with RUBI in the ATFMP, there is a 37% increase in performance, with a 510x speed increase over non-partitioning approaches. Additionally, RUBI matches the performance of the current domain-dependent ATFMP gold standard using no prior knowledge and with 10% faster performance.

1 Introduction

Two key elements in a multiagent reinforcement learning system are 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 often computationally expensive, and in large, highly coupled domains reward shaping quickly becomes computationally intractable.

Modeling the reward shaping technique (Proper and Tumer 2012) in relatively large domains (approx. 400 agents) works well, but requires tens of thousands of randomly generated examples to obtain these approximations. On the other hand, partitioning agents into hierarchies (Parker and Tumer 2012) or teams (Curran, Agogino, and Tumer 2013) speeds up computation time for much larger domains (approx. 10,000-40,000 agents) while still using the reward shaping technique without approximation error. In order to create these hierarchies or teams, the algorithm designer must also have a fundamental understanding of how agents are coupled. These approaches break down in complex domains where the amount of impact an agent has on another is unknown, and in situations where the algorithm designer has no domain knowledge.

In this paper we introduce Reward/Utility-Based Impact (RUBI) scores. RUBI partitions agents by determining the effect of one agent's action on another agent's reward. Using this metric it develops similarity metrics between all agents in order to usefully partition agents and therefore reduce the complexity of the learning problem. In contrast to many other partitioning approaches, this has the advantage of requiring *no domain knowledge*.

We test RUBI in the El Farol Bar Problem (Arthur 1994) and the Air Traffic Flow Management Problem (ATFMP). In the ATFMP we use the approach developed by Curran et al. (2013), Agogino (2009) and Rios and Lohn (2009). 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 can be used to better coordinate aircraft and mitigate the propagation of congestion and the associated delay, but which aircraft should be delayed? There are tens of thousands of flights every day within the United States (OPSNET 2011), making the search space in this problem huge.

The contributions of this work are:

- Generality: RUBI requires no prior knowledge of the domain, using it only to obtain reward information.
- Ease-of-use: RUBI removes the need to derive similarity metrics from the domain, removing the need for domain experts in situations where a domain expert isn't available.
- Performance: RUBI discovers non-trivial agent partitioning by 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, multiagent coordination and reward shaping. Section 3 contains the key contribution of this work, RUBI, and describes the basic algorithm and defines reward-based impact. In Section 4 we describe the experimental validation approach taken using multiagent coordination in both the Bar Problem and ATFMP. Experimental results are then provided in Section 5, followed by the discussion and conclusion in Sections 6 and 7.

2 Background

To motivate our approach, we outline previous work performed in the field of agent partitioning and describe the reward shaping technique used in this work.

Agent Partitioning

Previous work in agent partitioning has focused mainly on how to divide the problem, by the state space, actions or goals. Jordan and Jacobs (1993) 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. However, many multiagent learning domains, such as those in this paper, are unsupervised.

A common solution is to partition actions so that each agent is responsible for a small number of actions. Sun and Pearson (1998) divided actions into two types, and a separate agent handled each type. This approach leverages direct domain knowledge, which is not always available, and partitioning actions does not apply well in domains where all actions need to be explored.

Another approach is to partition system-level goals into smaller tasks. In the work by Dayan and Hinton (1993), they accomplished goal partitioning through on-line task allocation, where agents are organized in a hierarchy, and high-level agents assign goals to agents lower in the hierarchy. In the work by Reddy and Tadepalli (1997), the approach is more structured; agents learn the partitioning of the goal through externally provided examples. These approaches assume that the system-level goal can be subdivided, which is not always the case.

Overall, current partitioning techniques work well in smaller multiagent system domains. Zhang and Lesser (2010) effectively partition 324 agents in a distributed task allocation problem. They use joint-even-driven interactions and conditional probabilities at every timestep to compute the 'gain of interactions'. This additional on-line computation makes this approach becomes computationally intractable in domains with tens of thousands of agents. However, the work of Zhang and Lesser has a different focus than the work presented here. Zhang and Lesser effectively increase performance in their experiments, where we wish to scale to extremely large MAS while sacrificing as little performance as possible.

Reward Shaping

In learning algorithms reward design is important for keeping convergence time low and 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; this is known as the Tragedy of the Commons (Hardin December 1968).

The difference reward (Wolpert and Tumer 2002) is evaluated such that each agent's reward is related to the individual's contribution to team performance, thereby improving the signal-to-noise ratio. This leads to better policies at an accelerated convergence rate. The difference reward is defined as $D_i(z) = G(z) - G(z - z_i + c_i)$, 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.

3 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 review the variety of ways to develop RUBI impact scores and simulations.

RUBI Overview

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 the domain and partitions agents together based on how similar two agents 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.

RUBI is motivated by a simple concept: 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:

$$|L_i(z) - L_i(z - z_j)| > |L_k(z) - L_k(z - z_j)| \Rightarrow \\ Similarity(i, j) > Similarity(k, j) \quad (1)$$

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 on agent k. This is the essential idea behind RUBI, and is encoded in RUBI (Algorithm 1) on line 11.

The RUBI Algorithm

The RUBI algorithm (Algorithm 1) first initializes the impact table, 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. RUBI then runs a simulation 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, and update the impact table, C. Since this is a localized reward, and the algorithm is highly parallelizable, this is a fast operation. This is a high level understanding of RUBI, and the following sections will explain how the impact data is computed, simulation specifics, and the $ACT()$ function.

Algorithm 1 Reward/Utility-Based Impact Algorithm

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

Implementation

The impact data used to compute the similarity matrix are obtained from a localized reward or utility with respect to an agent. Typically the use of local rewards in congestion problems leads to a suboptimal solution. However, because RUBI is implemented prior to learning, we can use the local reward to identify agent similarity, and in the learning stage use a reward which will promote coordination.

In this work we use reinforcement learning, but since RUBI acts outside the learning process, it can be applied to any system using a learning mechanism. RUBI uses local rewards to construct the partitions, but the developer can specifically build a localized reward for the partitioning. When computing the difference between 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.

The impact table, C, is 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 an accurate impact estimate.

Simulation

The simulation is also an aspect that can be widely varied by the developer when using RUBI. One example borrows 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 (Yu, Dang, and Yang 2012). We can apply the same concepts developed in transfer clustering to this RUBI simulation. 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 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. 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 agents take random actions, they are not driven by any motivating logic, and impact scores will be biased only toward agents who consistently impact each other.

4 Experimental Validation

We validate our approach in the heterogeneous bar problem and the ATFMP. The El Farol Bar Problem (Arthur 1994) is a benchmarking domain typically used in preliminary work as an abstraction of a congestion domain. We use this domain to show preliminary results demonstrating the general effectiveness of RUBI before applying it to the more complex ATFMP. We then provide an overview of the ATFMP and our multiagent approach. In both experiments, agents learned using Action-Value Learning (Stateless Q-Learning) with a zero-initialized value table. This is a stateless approach where agents map actions to values representing the quality of that action.

In this work, we treat each partition of agents as an independent problem. Agents from one partition could potentially affect the environment of agents in another partition, but we attempt to minimize the partition overlap. We will use the term *reward-independent* to denote when one partition of agents will have no impact on the rewards of other partitions.

Heterogeneous Bar Problem

The El Farol Bar Problem (Arthur 1994) is an abstraction of congestion problems. In this problem there is a capacity c which provides the most enjoyment for everyone who attends the bar on that particular night. This is a stateless one shot problem where agents choose the night they attend the bar, and receive a reward based on their enjoyment. The traditional bar problem local reward is a function of the attendance of that night:

$$L_i = e^{\frac{-x_i(z)}{c}} \tag{2}$$

where $x_i(z)$ is the attendance on the night agent i went to the bar. The system-level reward is a simple summation of these local rewards across all agents:

$$G(z) = \sum_{k=0}^{K} x_k(z) e^{\frac{-x_k(z)}{c}} \tag{3}$$

where k is the index of the night, and $x_k(z)$ is the number of people who attended on the kth night.

From the reward, we know that if there are enough agents to be equally spread out across the bars, $n \leq ck$, this becomes a scheduling problem. This problem becomes a congestion problem when there are more than twice as many agents as the capacity for each night allows, $n > 2ck$. In this case a good group policy is for the majority of agents to attend one night, thus making agents attending that night receive a very low reward, and the rest of the agents equally distributing over the rest of the nights such that the number of agents for each other night becomes c, receiving the optimal reward for those nights.

In order to test RUBI's effectiveness at partitioning in congestion problems, we modify the bar problem by introducing heterogeneous agents that can attend the bar only a subset of nights, rather than any night. The problem is still the same, but there are now t types of agents. This modification gives us an intuitive partitioning of agents and allows us to directly compare a direct learning approach that finds the near-optimal solution to the partitioning given by RUBI.

Air Traffic Flow Management Problem

The ATFMP is a large congestion problem. Congestion problems are defined as a problem where agents share the same action space, and system performance is a function of how many agents take each action. Many congestion problems require coordination between agents, such that one agent must take a locally suboptimal action in order to benefit another agent, and raise the system-level reward.

The ATFMP addresses the congestion in the National Airspace (NAS) by controlling ground delay, en route speed or 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, 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 reducing the amount of delay each aircraft incurs is the fundamental goal of ATFMP.

The approach we use in the ATFMP follows the same approach by Curran et al. (2013), Agogino (2009) and Rios and Lohn (2009), who removed congestion completely from the system algorithmically through the use of a greedy scheduler. This greedy scheduler analyzed the schedule after each agent had taken an action, and greedily assigned delays to remove congestion. 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. They treated each partition independent of each other only when computing the reward, and therefore only computed rewards relative to the agents within a partition. They then performed multiagent reinforcement learning using the difference reward and the greedy scheduler. 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 domain-based agent partitioning, showing that rewards can be computed many times faster with minimal performance degradation.

Agent Definition In this paper, agents are assigned to one of 35,844 aircraft with cooperation enforced by airport terminals. 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 by that many minutes. The greedy scheduler then checks if the flight plans cause any congestion, and 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, so feedback can only be given once per simulation. Because agents are given no knowledge of the environment, they have no state. This simplifies the learning problem for each agent, but complicates 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.

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 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 focuses on the cumulative delay (δ) and congestion (C) throughout the system:

$$G(z) = -(C(z) + \delta(z)) \ , \tag{4}$$

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.

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 the delay in the system, $\delta(z)$.

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 cannot learn 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:

$$D_i(z) = \delta(z - z_i + c_i) - \delta(z) \ , \tag{5}$$

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

When using RUBI in the ATFMP we developed a partitioning reward by simplifying the simulation and using congestion information. At a high level we want to encapsulate how one agent affects another in the reward. We removed the greedy scheduler and used the congestion as information for the similarity data:

$$R = -(C(z)) \ , \tag{6}$$

This is a perfect example of how different the reward can be during partitioning than during learning.

Computational Complexity When performing reinforcement learning with the difference reward, you must remove an agent and recompute the system-level reward. This computation occurs once for each agent. In the ATFMP, the system-level reward is the summation of delays, an $O(n)$ routine. Therefore, each learning iteration is on the order of $O(n * O(removeAgent(a)) + n^2)$, where n is the number of agents. In the ATFMP each $removeAgent(a)$ call must call the greedy scheduler, an $O(n)$ algorithm, leaving the time complexity of the learning system to remain $O(n^2 + n^2) = O(n^2)$. When the complexity of removing an agent becomes larger than linear time, the cost of removing an agent dominates the cost of computing the agent reward. In this case the agent partitioning becomes more useful.

When using agent partitioning, this time complexity can be drastically reduced. When computing time complexity, the number of agents, n, can be replaced with the number of agents per partition p_n squared multiplied by the number of partitions, p:

$$O(n^2) = O(p_n^2 * p) \tag{7}$$

With many agents we see a significant speedup. If we have 10,000 agents in the system, and convert that to 100 partitions of 100 agents we get 10^6 iterations as opposed to 10^8, two magnitudes lower. Additionally, the more partitionable the domain, the better the speed up performance. If a domain is highly partitionable, then p becomes larger and p_n becomes smaller, meaning fewer agents per partition. This lowers time complexity by lowering the faster-growing term, and becomes extremely important when $removeAgent(a)$ involves a high time complexity operation. Also, keep in mind this is a growth rate complexity analysis. In actuality, the speed up is much higher. For example, in the ATFMP the greedy scheduler only has to reschedule the number of agents within each partition, rather than all agents.

5 Results

During preliminary analysis, RUBI works as expected in simple congestion problems, such as the El Farol Bar Problem (Arthur 1994). When we apply RUBI during partitioning in the ATFMP, simulation time decreases and the ease of RUBI 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.

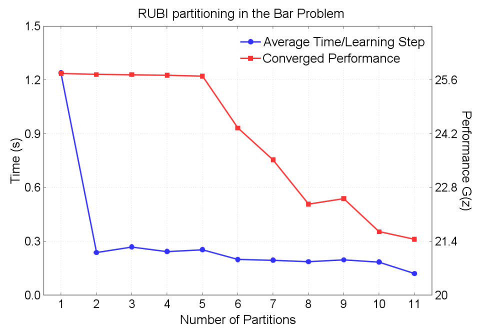

Figure 1: As the number of partitions decrease, the agents receive more information about the environment, leading to better performance at the cost of speed.

RUBI Performance in the Bar Problem

In our formulation of the heterogeneous bar problem each agent is placed on a team of other agents who go on the same subsets of days. We used 10 nights, 1000 agents, and 10 different types of agents during experimental runs. Each type of agent randomly generated 3 nights the agents of that type can go to the bar, and each agent was randomly given a type.

When partitioning agents with RUBI in the heterogeneous bar problem agents took random actions, and the localized reward was simply the local reward used traditionally (Equation 2). In this experimental set up, partitions had some overlap, as it is very unlikely that there is a type of agent completely reward independent from all other types. This causes performance to degrade when using partitioning. Figure 1 shows that this degradation is minimal with fewer number of partitions, and increases as more partitions are added, and additionally shows the computational speed-up involved when having many smaller partitions. This emphasizes the fact that partitioning without reward independent partitions increases speed at the cost of performance.

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 only congestion (Equation 6). Using RUBI, 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 2 shows a variety of partitions outperforming the greedy scheduler. During analysis, we found that the final performance of the ATFMP using RUBI-based partitioning was similar to domain-based partitioning performance. This is because at a converged

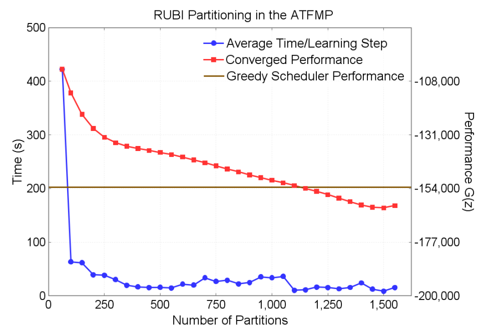

Figure 2: Again, we see a trade-off between performance and speed.

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.

When partitioning in a multiagent system, unless partitions are reward-independent, there is a trade-off between faster simulation time/reward calculation and 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 over non-partitioning approaches, and with a larger number of partitions we obtained a 5400x speed increase with a 20% increase in performance. Since non-partitioning approaches are computationally intractable in the ATFMP, we compare the computational time per learning step, rather than the overall time taken.

In addition, 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 partitioning included 3 partitions, while partitioning with RUBI yielded 61 partitions. This is due to using reward-based impact as a similarity metric. The actions of two agents may greatly affect each other during simulation, but their reward-based impact on each other might still be zero. 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.

6 Discussion

One of the key strengths of RUBI is its sheer simplicity and generality combined with computing highly informative similarity scores. It needs no prior knowledge about the domain to perform partitioning, but instead simply needs a localized reward from each agent to build the similarity matrix. This localized reward can be easily obtained from the system-level reward or utility already developed for the learning approach. 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 agent a affects agent b in a direct or indirect way. This indirect effect can be captured by RUBI and used as additional information when partitioning, leading to higher quality partitions in domains with complex interactions.

Another benefit of RUBI is that partitions built using RUBI are likely to be greater in number without loss of performance. This leads to faster computation time due to fewer agents per partition. Domain-based partitioning based on agent similarity encodes how often two agents impact each other. RUBI on the other hand looks more into how the actions of one agent impact another agent's rewards. For example, suppose in a congestion scenario agent a and agent b go through the same part of the environment, but are never congested. Using domain-based partitioning, two agents that go through the same area many times would be partitioned together, so agent a and agent b would be partitioned together. In partitioning using RUBI, if over a few thousand trials the reward impact of each agent is always 0, those agent's actions never impact each other's rewards, so they would not be partitioned together. The same is true if the congestion of each agent remains the same non-zero value; the actions do not affect the reward, so they are not partitioned together.

Lastly, RUBI was able to find an effective partitioning for 35,000 agents in the ATFMP within a few hours. This is a pre-processing technique and is not included in the learning speed. RUBI was able to reduce the amount of computation in the ATFMP by hundreds of hours, and therefore the additional computation is negligible.

7 Conclusion

This paper introduces RUBI, a partitioning algorithm that computes reward-based impacts to perform agent partitioning, removing the need for prior knowledge of the system. This method also removes the need to develop similarity metrics derived from expert domain knowledge. Additionally, by removing all knowledge about the domain and instead 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.

Future work in RUBI would involve performing a formal analysis of the relation between the number of iterations of RUBI and partition performance. Additionally, approximating the impact score of each agent, rather than using an accumulation, has the potential of being informative when performing an analysis of a system. Lastly, performing distributed clustering would be an important, yet simple extension to this work.

This work was partially supported by the National Science Foundation under Grant No. CNS-0931591.

References

Agogino, A. 2009. 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*.

Arthur, B. W. 1994. Inductive reasoning and bounded rationality. In *American Economic Review (Papers and Proceedings)*, volume 84, 406–411.

Curran, W. J.; Agogino, A.; and Tumer, K. 2013. Addressing hard constraints in the air traffic problem through partitioning and difference rewards. In *Proceedings of the 2013 international conference on Autonomous agents and multiagent systems*.

Dayan, P., and Hinton, G. E. 1993. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, 271–278. Morgan Kaufmann.

Hardin, G. December 1968. The tragedy of the commons. *Science* 162:12431248.

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

OPSNET, F. 2011. US Department of Transportation website. (http://www.faa.gov/data_statistics/).

Parker, C. H., and Tumer, K. 2012. Combining difference rewards and hierarchies for scaling to large multiagent system. In *AAMAS-2012 Workshop on Adaptive and Learning Agents*.

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

Reddy, C., and Tadepalli, P. 1997. Learning goal-decomposition rules using exercises. In *In Proceedings of the 14th International Conference on Machine Learning*, 278–286. Morgan Kaufmann.

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

Sun, R., and Peterson, T. 1998. Some experiments with a hybrid model for learning sequential decision making. *Information Sciences* 111:83–107.

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

Yu, L.; Dang, Y.; and Yang, G. 2012. Transfer clustering via constraints generated from topics. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*.

Zhang, C.; Lesser, V.; and Abdallah, S. 2010. Self-organization for coordinating decentralized reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, 739–746. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.