Agent Partitioning with Reward/Utility-Based Impact

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 simultaneously tens of thousands of aircraft affecting the system and no intuitive similarity metric between agents. When partitioning with RUBI in the ATFMP, there is a 37% increase in performance, with a 510x speed up over non-partitioning approaches. Additionally, RUBI matches the performance of the current domain dependent ATFMP gold standard using no prior knowledge and performing 10% faster.

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 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. 10000-40000 agents) while still using the reward shaping technique without approximation error. 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 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 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 Air Traffic Flow Management Problem (ATFMP) and use the approach developed by Curran et al. (Curran, Agogino, and Tumer 2013) and Agogino and Rios (Agogino 2009; 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 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 (OPSNET 2011).

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 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. Section 3 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. In Section 4 we describe the experimental approach taken using multiagent coordination, the difference reward, the greedy scheduler and agent partitioning. 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.

Agent Partitioning

Previous work in agent partitioning has focused on what specifically to partition, including the actions, state space and goals. Jordan and Jacobs (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 the one in this paper, are unsupervised.

Partitioning actions so that each agent is responsible for a small number of actions is another approach used often. Sun and Pearson (Sun and Peterson 1998) divided actions into two types, speed and turn, and each type was handled by a separate agent. 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.

The last partitioning technique we describe is to partition system-level goals into smaller tasks. In the work by Dayan and Hinton (Dayan and Hinton 1993), 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 (Reddy and Tadepalli), the approach is more structured. In this work the partitioning of the goal is learned through externally provided examples. Overall, these approaches assume that the system-level goal can be subdivided, which is not always the case.

In our 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 we attempt to minimize the partition overlap. In a partitioning with complete reward independence, we essentially treat the problem as a set of smaller and easier independent problems. 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.

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 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 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. (Curran, Agogino, and Tumer 2013) and Agogino and Rios (Agogino 2009; Rios and Lohn 2009). This work found an alternative to performing multiobjective optimization on delay and congestion. Rather than treating congestion as a soft constraint, as is typically the case in reinforcement learning, these approaches 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.

Reward Shaping

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 (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, therefore the signal-to-noise ratio is improved considerably. This leads to final better policies at an accelerated convergence rate, and overcomes the Tragedy of the Commons. The difference reward is defined as:

$$D_i(z) = G(z) - G(z - z_i + c_i) \ , \qquad (1)$$

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.

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 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.

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.

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:

$$\begin{gathered} |L_i(z) - L_i(z - z_j)| > |L_k(z) - L_k(z - z_j)| \Rightarrow \\ SIM(i,j) > SIM(k,j) \end{gathered} \qquad (2)$$

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:

$$C_{r,a} \leftarrow C_{r,a} + |L_a(z) - L_a(z - z_r)| \ , \qquad (3)$$

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.

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. 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 based on equation 3. 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, the $ACT()$ function, and finally when this partitioning algorithm is practical.

Algorithm 1 Reward/Utility-Based Impact Algorithm

```
function RUBI($sim$)
    $C \leftarrow NxN$
    for $i \leftarrow 1$ to $iterations$ do
        $actions \leftarrow ACT()$
        $sim.run(actions)$
        $L(z) \leftarrow sim.getRewards()$
        for $r \leftarrow 1$ to $N$ do
            $sim.removeAgent(r)$
            $L(z - z_r) \leftarrow sim.getRewards()$
            for $a \leftarrow 1$ to $N$ do
                $C_{r,a} \leftarrow C_{r,a} + |L_a(z) - L_a(z - z_r)|$
            end for
            $sim.addAgent(r)$
        end for
    end for
end function
```

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 leads to a suboptimal solution, as local rewards correspond to greedy agents. Congestion problems require a few agents to receive suboptimal rewards in order for the system-level reward to increase. 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 also applies to algorithms that use utilities. A traditional local reward can work, or the developer can specifically build a localized reward 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 3 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.

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. This is yet another aspect of this algorithm that can be chosen by the developer. 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 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.

Benefits of RUBI

One of the key strengths of RUBI is its sheer simplicity and generality combined with computing highly informative similarity scores, 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 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 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. 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 agents reward. For example, 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 agents actions never impact each others rewards, therefore 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, 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.

4 ATFMP Approach

In this section we give a brief overview of the learning approach used in this paper. We provide an overview of one ATFMP approach that was developed by Curran et al. (Curran, Agogino, and Tumer 2013) and Agogino and Rios (Agogino 2009; Rios and Lohn 2009), and the reader is referred to this related work if they have any additional questions.

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 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.

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:

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

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.

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:

$$G(z) = -\delta(z) \tag{5}$$

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:

$$D_i(z) = \delta(z - z_i + c_i) - \delta(z) \ , \tag{6}$$

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 took out the greedy scheduler and used the congestion as information for the similarity data:

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

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

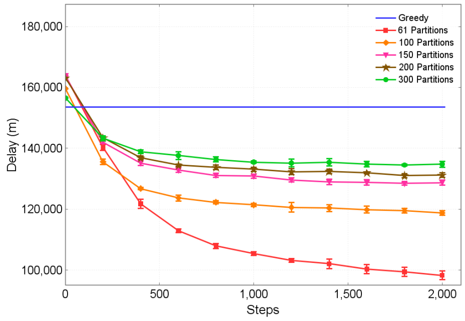

Figure 1: As the number of partitions decreases, performance improves while time complexity increases. Note that a reward-independent partitioning using RUBI includes 61 partitions.

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 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. (Curran, Agogino, and Tumer 2013) and Agogino and Rios (Agogino 2009; Rios and Lohn 2009), except utilizing RUBI.

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 7). 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 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.

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 5400x speed up with a 20% increase in performance.

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. (Curran, Agogino, and Tumer 2013) and Agogino and Rios (Agogino 2009; Rios and Lohn 2009), the number of overlapping sectors. 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.

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, reducing the computational benefits of partitioning.

Partitioning with RUBI developed better similarity scores than domain-based partitioning (Figure 2). 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 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. The actions of two agents may greatly affect each other during simulation, but their reward-based impact on each other might still be zero.

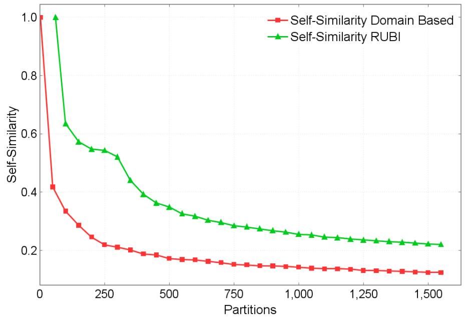

Figure 2: Partitions formed with RUBI had higher self-similarity than using domain-based partitioning. This leads to higher quality learning with respect to each partition.

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, 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 difficult to predict. 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 an accumulation, has the potential of being 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.

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 multi-agent 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. 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*.