Transfer Learning based on Optimal Reward

Monica Eboli, Weihong Guo, Nan Jiang, Sean Newman

12/16/2011

1. Motivation and Background

1.1.Optimal Rewards

In reinforcement learning, agent designers build agents that seek to maximize their long term expected return. This reward incorporates two aspects that are sometimes assumed as one; namely the reward is used to evaluate how the agent is performing relative to the designer's goals, and the reward also guides the agent's behavior. Separating these functions into two different reward functions, and using the *intrinsic* (which guides the agent's behavior) rather than the external (also known as the *objective reward function*, which the designer uses to evaluate the agent's performance) has yielded results where an agent's performance, under limited, or *bounded*, conditions, produces greater long term return. Sorg, Singh and Lewis[1] showed that by using a carefully chosen intrinsic reward, the limitations on the agent's performance may be mitigated. An intrinsic reward function here is linear combination of *reward features* which are scalars that can be chosen based on its potential to overcome the bounds on the agent. The objective reward is usually one of the features. An example of other features is *inverse recency* which would provide a reward inversely proportional to how recently a state was last visited. A larger weight on this feature would encourage an agent to explore states that had not been visited recently.

Formally, an agent, G, that receives an observation $o \in O$ from its environment M, takes an action,$a \in A$, and repeats this process for a certain time horizon. The agent designer's goals are represented by the objective reward function R_O which produces a cumulative *return*which is accumulated according to a measure such as the summed rewards, or the averaged rewards over a finite time horizon. The agent's goals, which should be to maximize the designer's utility within the confines of the limitations of that agent, are represented by an intrinsic reward function G(R). Intrinsic reward functions are parameterized by a vector of parameters which define a space of reward functions. The optimal reward function is the reward function that is given by the definition,

$$R^* = \arg\max_{R \in \mathcal{R}} \mathbb{E}[U_{R_O(h)} | h \sim M\langle G(R)\rangle] \quad (1)$$

The optimal reward function is the reward function that optimizes the designer's expected objective cumulative reward for an agent *G*, acting in an environment *M*, with a reward function space $\mathcal{R}$.

1.2.PGRD: Policy Gradient for Reward Design

Sorg, Singh and Lewis[2] introduced Policy Gradient for Reward Design (PGRD) as a method for approximating the optimal reward function. PGRD was developed out of the observation that an agent's reward function can be viewed as a parameterization of its policy since the agent's behavior in following a planning algorithm, is determined by its reward function. PGRD is a stochastic online gradient ascent algorithm that updates the reward parameters by estimating the gradient of the cumulative objective reward with respect to each reward parameter. PGRD has been shown to improve reward functions during an agent's lifetime [2] especially with agents that have a limitation as described above in section 1.1.

1.3.Transfer Learning in Reinforcement Learning

In Reinforcement Learning problems transfer learning has been used in many contexts, among them the transfer of policies and of value functions between tasks [3]. Often the transferred knowledge, such as the policy or value function, is used to initialize learning with the new task or domain in order to improve learning in the new task. Sorg, Singh and Lewis haveempirically showed that the optimal reward may be transferred from one domain to another while mitigating the gap in the bounded agent's performance in the target domain[1].

1.4.Motivation for Transfer Learning based on Optimal Reward

In previous approaches of transfer learning, a mapping between the state spaces and sets of actions in two domains should be either provided manually or learned by the algorithm, which turns out to be a difficult problem[4]. In optimal reward, however, the knowledge learned in one domain is presented as the parameters, or the coefficients of the reward features. Some of the commonly used features are domain independent (e.g. inverse recency), while most domain dependent features can naturally generalize to the same type of domains with different complexities. Hence, transferring optimal reward functions has an inherited advantage: it does not need to learn the mapping between domains. For this reason, we believe it is worth trying to transfer the knowledge in terms of optimal reward.

2. Problem Statement

Consider two domains, M_1 and M_2 with some extent of similarities and an RL agent architecture G with bounds. Denote the instances of G applied in $\mathrm{M}_1, \mathrm{M}_2$ as $\mathrm{G}_{\mathrm{M}_1}(\theta_1)$ and $\mathrm{G}_{\mathrm{M}_2}(\theta_2)$, where θ_{i} is the parameters of the internal reward functions used in domain M_{i}.

Let $F_{M_2}(\cdot)$ denote the expected accumulative objective reward gained by an agent in M_2.

Suppose we have already learned the optimal reward in M_1 with parameter θ_1^*. The problem of transferring optimal reward is to find an algorithm $\mathrm{T}, \tilde{\theta}_2 = \mathrm{T}(\mathrm{G}, \mathrm{M}_1, \mathrm{M}_2, \theta_1^*)$, such that $\mathrm{F}_{\mathrm{M}_2}\left(\mathrm{G}_{\mathrm{M}_2}(\theta_2^*)\right) - \mathrm{F}_{\mathrm{M}_2}\left(\mathrm{G}_{\mathrm{M}_2}(\tilde{\theta}_2)\right)$ is minimized while computational resources spent by T is under control.

3. Methodology

3.1.Intuition

Consider the following simple example. The agent behaves greedily according to the immediate internal reward in a continuous task. There are two reward features ϕ_1, ϕ_2. (β_1, β_2) are the optimal parameters. Suppose the agent is at state s with history h, and two actions, a_1 and a_2, are available. The internal rewards for the actions are respectively

$$\beta_1\phi_1(\mathrm{s}, \mathrm{a}_1, \mathrm{h}) + \beta_2\phi_2(\mathrm{s}, \mathrm{a}_1, \mathrm{h}) \text{ and } \beta_1\phi_1(\mathrm{s}, \mathrm{a}_2, \mathrm{h}) + \beta_2\phi_2(\mathrm{s}, \mathrm{a}_2, \mathrm{h}) \quad (2)$$

Assume without the loss of generality that (β_1, β_2) are positive numbers (otherwise we can always flip the sign of the reward features). Also assume that

$$\phi_1(s,a_1,h) > \phi_1(s,a_2,h), \phi_2(s,a_1,h) < \phi_2(s,a_2,h) \quad (3)$$

which makes it an interesting case, since no action is dominant in both features. The agent prefers a_1 if $\beta_1\phi_1(s,a_1,h) + \beta_2\phi_2(s,a_1,h) > \beta_1\phi_1(s,a_2,h) + \beta_2\phi_2(s,a_2,h)$, namely

$$\frac{\phi_1(s,a_1,h)-\phi_1(s,a_2,h)}{\phi_2(s,a_2,h)-\phi_2(s,a_1,h)}\frac{\beta_1}{\beta_2} > 1 \quad (4)$$

This shows that (β_1, β_2) reflect how the agent makes tradeoff between the reward features. Now consider using feature ϕ_1' instead of ϕ_1, and $\phi_1' = c\phi_1 + d, c > 0$. It is obvious that we can achieve the same optimal policy by parameters (β_1', β_2), where $\beta_1' = \beta_1/c$. This is because

$$\frac{\phi_1'(s,a_1,h)-\phi_1'(s,a_2,h)}{\phi_2(s,a_2,h)-\phi_2(s,a_1,h)}\frac{\beta_1'}{\beta_2} = \frac{\phi_1(s,a_1,h)-\phi_1(s,a_2,h)}{\phi_2(s,a_2,h)-\phi_2(s,a_1,h)}\frac{\beta_1}{\beta_2} \quad (5)$$

which implies that with the same history and current state, policies specified by $\beta_1\phi_1 + \beta_2\phi_2$ and $\beta_1'\phi_1' + \beta_2\phi_2$ share exactly the same preferences over actions.

The result also applies to the cases where return or truncated return, rather than one-step reward, is considered, since return is the linear combination of reward at each time step. Heuristically, we generalize it to form our transfer method. To transfer the optimal reward parameters from one domain to another, we **calculate the magnitude of oscillation of each reward featureϕ_i, and use it to normalize the corresponding parameter β_i.**

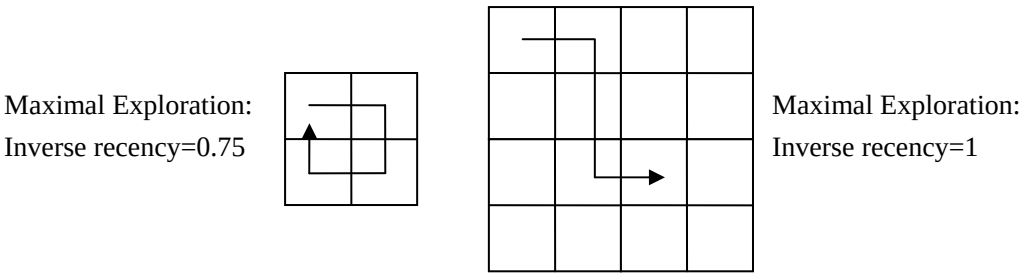


Fig.1. An example for the transfer intuition

Take two grid domains as an example, shown in Fig.1. For both domains, there are two reward features, the objective reward and the inverse recency (for convenience, action is not counted into the inverse recency here), which encourages exploitation and exploration respectively. In the left domain, the world is very small. All nearby squares may be visited only a few steps ago, whose inverse recency is low. On the contrary, in the right domain, the agent may be surrounded by many long-time-no-see squares, whose inverse recency is high. The most long-time-no-see squares in two domains should be treated equivalently as they both realize maximal exploration. If we want to balance exploitation and exploration exactly in the same way for the two domains, the parameter for inverse recency in the right domain should be less than that in the left domain in compensation for the difference in the values of the features.

3.2.Normalization

We transfer the optimal reward by normalizing the parameters in the following way:

$$\beta_i' = \frac{S\{\phi_i\}}{S\{\phi_i'\}}\beta_i \quad (6)$$

where $S\{\cdot\}$ is some statistical property of a reward feature that reflects the magnitude of oscillation, and we will estimate that from the empirical experience about the domain.

We do not have a solid idea about how to select $S\{\cdot\}$. Instead, some reasonable options are proposed heuristically. The first one is

$$S\{\phi_i\} = \sqrt{E\{\sum_{\substack{a,a' \in A \\ a \neq a'}} [\phi_i(s,a,h) - \phi_i(s,a',h)]^2\}} \quad (7)$$

which shows how the value of the reward feature varies as different actions are taken. However, this does not apply to the cases where the space of actions is very large or even continuous. A second choice may be

$$S\{\phi_i\} = \sqrt{E\{[\phi_i(s,a^*,h) - \phi_i(s,a,h)]^2\}} \quad (8)$$

wherea^* is the action specified by the optimal policy, and a is a randomly chosen one. A more simplified version is

$$S\{\phi_i\} = \sqrt{Var\{\phi_i(s,a,h)\}} \quad (9)$$

where the variance is taken in the direction of time. This one applies to some cases but is problematic in others, which we will discuss later in the experiment section.

3.3.Distribution Problem

We expect to estimate the statistical properties from the empirical experience about the domains. Since we are talking about expected values and variances, there is a natural problem to ask: how is the history distributed? Or, equivalently, when we get the empirical data from the agent, what policy doesit follow?

Our answer to this question is that the agents in both domains should follow the optimal policies. Although following random policies also seems reasonable, we use the next example to show why this is not ideal.

Again we have two grid domains. The arrow shows the optimal policy, which is a path. The shadow area in the left domain is some “trap” that gives the agent extremely negative rewards, and the agent learns to avoid it. In the right domain, the trap no longer exists, but the optimal path is still the same as in the left one. These two domains are similar in the regions passed by the optimal path, and the rest part turns to be irrelevant, however they are different in that area. Hence, we should to be able to transfer the knowledge from one to the other. However, if we run random agents in both domains, estimation of the statistical properties may be largely affected by the experiences in the “trap” area, and impair the effectiveness of transfer.

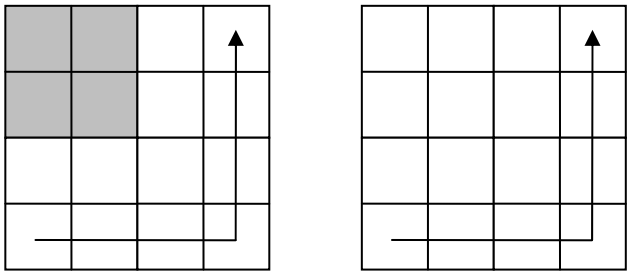


Fig.2. Two grid domains with difference outside the optimal paths

3.4.Iterative Method

Note that we want the agents to follow optimal policies in both domains to estimate the statistical properties of the reward features. However, since our ultimate goal is to transfer the knowledge from one domain to another, how can we know the optimal policy in the new domain before we transfer?

Fortunately, if we assume that the transferred parameters lead to an approximately optimal policy,the statistical properties of the features and the optimal policy will determine each other, as shown in Fig. 3. Analogous to policy iteration and value iteration in conventional reinforcement learning, we develop an iterative method to transfer the optimal reward.

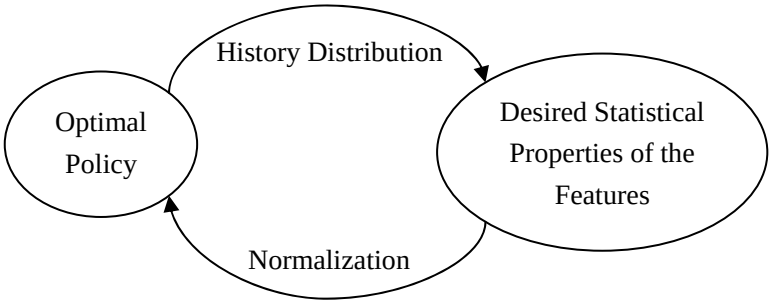


Fig. 3. Iterative Method

Starting from a policy, or, a set of reward parameters, we run the agent in the domain and compute the statistical properties of the features (similar to “policy evaluation”). According to the normalization equation, we improve the parameters to get a better policy (similar to “policy improvement”). If the iteration converges, we will get the normalized parameters as the result of transfer.

The process discussed above is analogues to policy iteration. When the reward features are not sensitive to the history in the distant past (which is usually the case), we can even do the iteration online, analogues to value iteration. The algorithm is described as pseudo-code. Here “target” is the sampling of the estimated quantity at current time step. For $S\{\phi_i\} = \sqrt{\mathrm{Var}\{\phi_i(s,a,h)\}}$, we can separately estimate mean value and the second moment, and calculate the standard deviation from them.

1. Find the optimal reward parameters β_i;
2. Estimate $\hat{S}\{\phi_i\}$ in the original domain;
3. Initialization: $\beta_i' = \beta_i$;
4. While terminal condition not satisfied
5. One-step simulation of the agent under β_i' in the new domain;
6. For each feature i
7. $\hat{S}\{\phi_i\} \leftarrow \hat{S}\{\phi_i\} + \alpha(\mathrm{target} - \hat{S}\{\phi_i\})$;
8. $\beta_i' \leftarrow \beta_i \hat{S}\{\phi_i\}/\hat{S}\{\phi_i'\}$;
9. End for
10. End while

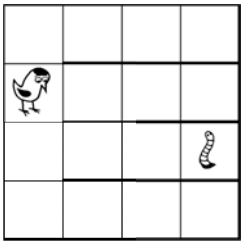
3.5.Discussion on Episodic Tasks

We have been focusing on continuous tasks. The reason that the method does not naturally work for episodic tasks is that the reward in episodic tasks have a quite different nature compared to that in continuous tasks. For example, a constant reward in continuous tasks is meaningless. However, for an episodic task, a negative constant reward encourages the agent to complete the task as soon as possible, while a positive constant reward encourages the agent to survive as long as possible. We have also come up with the statistical property that applies to normalization in transferring between continuous tasks, such as $S\{\phi_i\} = \sqrt{E\{[\phi_i(s,a,h)]^2\}}$. But due to the limit of time and the selection of domains for the experiments, we decide to concentrate on continuous tasks and leave episodic ones for future discussion.

4. Experiments and Evaluation

To validate the proposed transfer methodology and evaluate its performance, we carried out experiments in 2 domains: the worm world and the elevator domain.

4.1.Worm World



The environment. Consider a *n*-Corridor environment that consists of a *n***n* grid world with *n* dead-end corridors (rows) separated by impassable walls. The (bird) agent has four available actions: moving one step in each of the four cardinal directions, North, South, East, and West. If the agent takes an action that would move it off the grid or make it hit the wall, the agent just stays in place. There is a (worm) food source randomly located in one of the *n* right-most locations at the end of each corridor. The agent has an *eat* action, which eats the worm when the agent and the worm are at the same location. After the agent eats the worm, the agent becomes *satiated* for 1 time step, and the worm disappears. Immediately, a new worm appears randomly in one the other (*n*-1) potential worm locations. At all other time steps, the agent is *hungry*. The agent observes only its location and whether or not it is hungry.

Our goal is to maximize the number of worms eaten. Thus, the objective reward function provides a reward of 1 when the agent eats a worm (i.e. is satiated) in the current observation, and a reward of 0 otherwise.

The agent. The agent, G_d (short for $G(R_I, d)$), is a depth-*d* planning model-based learning agent which acts greedily with respect to the *d*-step action-value function $Q_d(s,a)$. The depth *d* is a parameter controlling the degree of boundedness.

Given that the agent cannot observe the location of food, it cannot plan to go to it in the shortest path. Also, given that the agent's observations do not tell it what locations it has visited since the last time it ate food, it cannot plan to avoid exploring locations that should be known not to have food based on the agent's history.

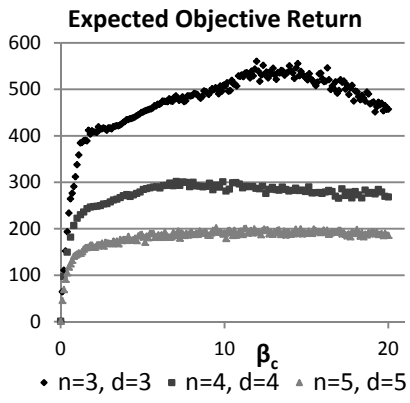
The reward space. $R_I(s,a,s') = \beta^T \phi(s,a,s',h,G)$ is the general form of the rewards, where β is a parameter vector, and ϕ is a vector of features that may depend on statess and s', the action *a*, features of history *h*, and in some cases on internal variables specific to the agent *G*.

In our experiment, we consider the reward space $R_I(s,a) = \beta_\varepsilon \phi_\varepsilon(s) + \beta_c \phi_c(s,a,h)$, where β_ε and β_c are the two parameters, feature $\phi_\varepsilon(s)$ is 1 when the agent is satiated in state *s* and 0 otherwise, and feature $\phi_c(s,a,h) = 1 - \frac{1}{c(s,a,h)}$, where $c(s,a,h)$ is the number of time steps since the agent previously executed action *a* in state *s* within history *h*. Feature ϕ_c captures recency; the feature's value is high when the agent has not taken the indicated state-action pair recently. When β_c is positive, the agent is rewarded for taking actions that it has not taken recently from the current state.

Results. We separately optimized the interval reward function as measured by the mean objective return obtained during a 10000 step horizon for each combination of world size $n \in \{3,4,5,6\}$ and depth $d \in \{1,2,3,4\}$, averaged over 20 trials, and one more experiment at {*n*=5, *d*=5}, averaged over 5 trials due to time limit. We obtained the optimal β_c^* for each experiment setting to achieve its optimal reward. (In experiments, we just fix β_ε to be 1, so β_c actually represents the relative ratio between the 2 parameters.) Then we estimated the statistical quantity $S\{\phi_i(s,a,h)\}$ for feature ϕ_ε and ϕ_c at $\beta_\varepsilon = 1$ and the optimal β_c^* using the transferring methodology in section 3.

Looking through all the results, we found that {*n*=3, *d*=3}, {*n*=4, *d*=4}, and {*n*=5, *d*=5} give very similar $\frac{S\{\phi_c(s,a,h)\}\cdot\beta_c^*}{S\{\phi_\varepsilon(s,a,h)\}\cdot\beta_\varepsilon}$ ratios (results for other experiment scenarios are not shown here). We then conclude that the proposed methodology for transfer learning can be best applied across experiment settings that represent similar characteristics.

Experiment Setting	**n=3, d=3**	**n=4, d=4**	**n=5, d=5**
$\boldsymbol{\beta_\varepsilon}$	1	1	1
Optimal $\boldsymbol{\beta_c^*}$	12.0	7.1	9.5
$\mathbf{S\{\phi_\varepsilon(s,a,h)\}}$	0.294458	0.206415	0.209901
$\mathbf{S\{\phi_c(s,a,h)\}}$	0.339977	0.351301	0.387398
$\frac{\mathbf{S\{\phi_c(s,a,h)\}\cdot\beta_c^*}}{\mathbf{S\{\phi_\varepsilon(s,a,h)\}\cdot\beta_\varepsilon}}$	13.80861	12.03375	17.52409①



① Although the third ratio seems different from the first two, we should notice that the $\beta_c - \text{Objective Return}$ curve is quite flat for {*n*=5, *d*=5}, and a wide range of β_c's yield near-optimal performance. Among them is one that generates the ratio that is very close to cases of {*n*=3, *d*=3} and {*n*=4, *d*=4}.

We run the iterative method (offline) to transfer from {*n*=3, *d*=3} to {*n*=4, *d*=4}. Following is the result of 10 iterations. The last two iterations yield similar results so the iteration has

converged. Rescale the parameters to let $\beta_\varepsilon = 1$, we get $\beta_c = 8.9$, which has near-optimal performance.

Iteration	1	2	3	4	5	6	7	8	9	10
Transferred β_ε	2.42	1.18	1.35	1.33	1.66	1.36	1.84	2.10	1.56	1.42
Transferred β_c	11.21	18.18	11.35	9.64	15.26	14.29	9.67	10.66	12.62	12.66

We have also tried the online iterative method but it does not converge. We analyze the data and find the reason. When the parameters are adjusted to the optimal ones, it takes too long to correct the estimates of the statistical properties, which makes the algorithm think that the current parameters are not desired and keeps changing it. When it runs into the area of poor performances, the algorithm is fooled that the objective reward is naturally sparse in the new domain and tunes up its coefficient. This further deteriorates the performance and the algorithm never converges. A really small step-size factor may solve the problem, but it will also make the computation time unacceptable.

4.2.Elevator

The domain

In order to conduct our experiment in a more complex domain, we decided to use a multiple elevator domain. We found a classical one from James Lewis at UMASS[5], which was a continuous time implementation of the elevator domain. Since we were going to apply the method to discretized environments as a start due to ease of feature definition and implementation we created our own elevator discrete domain.

Our domain is determined by the number of floors *f* and number of elevators *e*. To fully define the domain we determined the states, possible actions, dynamics and rewards.

State – is determined by elevator position (e_i) and how many passengers n_{jk} in each position *j* (inside elevator *e* or in floor *f*) and their desired destination *k* (floor *f*).

Actions– is determined by the action a_{ei} to be taken by each elevator, they may be *up*, *down*, *open door*, *stay*. On *up* the elevator goes up one floor, on *down* the elevator goes down one floor, on *open* the elevator door is opened and people who want to go to the specific floor descend and some people on the floor go in. In order to determined who should go in we created the rule: if currently there are more people going down on the elevator, people who wish to go down should enter, otherwise, people who want to go up.

Reward – is -1 for each passenger waiting at any position in each time step. After the passenger it delivered, it stops being counted.

Dynamics – There are three types of people that arrive: arriving to the building on the first floor, leaving the building (all whose destination is the first floor) and people moving inter-floors. We consider the module of a Gaussian with mean zero and variance one to define the movement. We maintain arrival multiplied by a factor during the first third of the simulation, then after we increase inter-floor movements and at the last third we increase the people leaving the building.

Features

As explained in the background section, we need to define features in order to obtain the optimum rewards based in the available features. Running the method that actually determines how relevant or not is the feature for the model. For this domain we defined three features:

Sparsity – Well distributed are the elevators throughout the floors. To measure sparsity we measured how many floors away was each floor to the closest elevator and we added up this number for all floors.

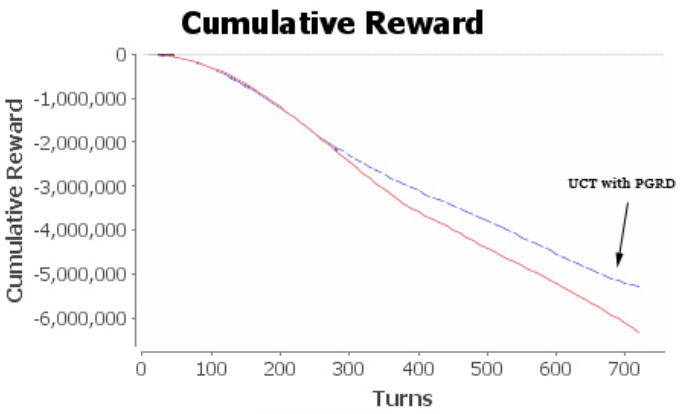
$$Sparsity = \sum_{0}^{numFloors} min_i |floor - elevator_i| \quad (10)$$

Inverse Recency– For each floor we would measure how many times steps it has been since an elevator opened a door on that floor.

People inside elevator – We would at each round count how many people were inside the elevator. The intuition behind this feature is that probably the reward for someone waiting outside the elevator or inside should not be the same but we would not know how to balance that, so we left this balance to be done by PGRD.

The experiment

For our experiments we considered three cases:



Elevator Domain with 2 Elevators, 10 Floors comparing UCT and UCT with PGRD using the Sparsity feature

- 5 floors 2 elevators
- 8 floors 4 elevators
- 10 floors 3 elevators

First we ran them using the planning algorithm UCT[6]. From those experiments we figured out that the only feature that would improve the method was the *Sparsity* one, shown on the left. So from that we learned that from those cases we would only be comparing the coefficients for the Sparsity feature.

From that, we ran the three tests and captured the mean and variance of the feature.

For Sparsity Feature (720 time steps):

	Coefficient	**Mean of values**	**Variance of values**
2Elevators, 5 Floors	-2.050171148315031E10	4.957	4.815
4Elevators, 8 Floors	-6.05094954211733E10	7.261	11.766
2Elevators, 10Floors	6.964319569794902E11	30.525	127.105

We could not find the expected correlation between the feature mean, variance and the ideal coefficient. As it can be seen above, it even changes value.

We believe this happened because as the model evolves in the search for the ideal action as it

runs, for each experiment, this ideal action strategy may vary. As the sparsity feature is directly connected to the action taken, we believe their variance of values is not a good measure given that it not only measures the state dynamic change but it is also influenced by the action.

5. Conclusions

In this project, we propose a method for transfer learning based on optimal reward. We start from very simple examples to show how normalization is done and try to apply to general cases. For the purpose of estimating the statistical properties, we need the unknown optimal policy in the new domain, which naturally gives rise to an iterative method analogues to policy/value iteration. However, the experiment does not show that the method is working well. After analyzing the test result, we find 3 reasons that why it is so:

(1) Agent performance is usually robust to the reward parameters. A large range of parameters may yield near-optimal performances that are equally good, thus do not give us accurate knowledge about the domain.
(2) There are no strong reasons that Eq.(2) holds for general cases where domains are not exactly the same in dynamics.
(3) The iteration process is unstable, and is not guaranteed to converge (and in many cases it does not).

6. Individual Effort

Monica defined the elevator domain and its three features. She implemented the domain and the inverse regency feature. She was responsible for PGRD integration, testing and test analysis. She wrote the elevator domain description and its tests and results of the report. She will also create the poster for presentation.

Nan proposed the transfer methodology and made theoretical analysis. He also collaborated with Weihong to design the worm domain and the experiments on it and implement them. He analyzed the result and concluded why the method was not doing well. He wrote the methodology part and the conclusion part of the report.

Sean was responsible for the UCT and PGRD integration with the model, the people inside elevator and sparsity feature implementation, running tests on the data base and obtaining statistical results for the analysis and the analysis. He wrote the introduction and background of the report.

Weihong was responsible for the exhaustive search for optimal rewards and feature parameters in worm world, running experiments across a wide range of scenarios and obtaining results for the analysis. She wrote the description, experiments, and results of the worm world domain, and then finalizing the report.

References

[1] J. Sorg, S. Singh and R. L. Lewis. (2010) Internal Rewards Mitigate Agent Boundedness, *Proc of ICML 2010*.

[2] J. Sorg, S. Singh and R.L. Lewis. (2010) Reward Design via Online Gradient Ascent. *Advances in Neural Information Processing Systems*, 2010.

[3] M. E. Taylor andP. Stone. (2009) Transfer Learning for Reinforcement Learning Domains: A Survey. Journal of Machine Learning Research, 10(1):1633-1685.

[4] M. E. Taylor and P. Stone. (2011) An Introduction to Inter-task Transfer for Reinforcement Learning. AI Magazine, 32(1):15-34.

[5] Reinforcement Learning Repository at UMass, Amherst, http://www-all.cs.umass.edu/rlr/domains.html

[6] L. Kocsis , C. Szepesvári (2006) Bandit based Monte-Carlo Planning, *Proceedings of the 11th International Conference on Computer Aided Systems Theory.*