In *The First Conference on Artificial General Intelligence* (AGI-08),
Memphis, Tennessee, March 2008.

Transfer Learning and Intelligence: an Argument and Approach

Matthew E. TAYLOR, Gregory KUHLMANN, and Peter STONE
Department of Computer Sciences
The University of Texas at Austin
{mtaylor, kuhlmann, pstone}@cs.utexas.edu

Abstract. In order to claim fully general intelligence in an autonomous agent, the ability to learn is one of the most central capabilities. Classical machine learning techniques have had many significant empirical successes, but large real-world problems that are of interest to generally intelligent agents require learning much faster (with much less training experience) than is currently possible. This paper presents *transfer learning*, where knowledge from a learned task can be used to significantly speed up learning in a novel task, as the key to achieving the learning capabilities necessary for general intelligence. In addition to motivating the need for transfer learning in an intelligent agent, we introduce a novel method for selecting types of tasks to be used for transfer and empirically demonstrate that such a selection can lead to significant increases in training speed in a two-player game.

Keywords. Transfer Learning, Game Tree Search, Reinforcement Learning

1. Introduction

A generally intelligent agent deployed in any non-trivial environment must be able to learn: no designer can anticipate every possible encountered situation. Specifically, intelligent agents need to learn how to *act*, which is the purview of *Reinforcement learning* [8,15]. Reinforcement learning (RL) problems are defined as those in which learning agents sequentially execute actions with the goal of maximizing a reward signal, which may be time-delayed. The RL framework appeals when considering the design of a generally intelligent agent as it is established, flexible, and powerful.

RL approaches have been gaining in popularity in recent years as methods have matured that are able to handle complex problems with noisy sensors, noisy actuators, and continuous state spaces (e.g., helicopter control [12] and Keepaway [14]). One of the main advantages to RL, unlike many machine learning approaches, is that no labeled examples are required. Such flexibility allows learning from general environmental feedback (a *reward signal*) but it comes at a price. When RL methods begin learning without any background knowledge, mastering difficult tasks may be slow or infeasible. If an agent requires months or years worth of experience to master relatively simple tasks, it will be a stretch to claim that it is generally intelligent. Thus a critical component to achieving intelligent behavior is not only the ability to successfully learn, but also the ability to learn *quickly*, from limited training experience.

In order to utilize RL algorithms to control a generally intelligent agent, we must overcome three shortcomings, one or more of which are typically present in current applications. Firstly, RL algorithms have typically been applied to simple tasks, such as the discrete task of gridworld. Secondly, many algorithms are sample-inefficient and require millions of training examples in order to perform well [17]. Thirdly, substantial amounts of human knowledge must often be used in order to define the learning problem and direct the learner towards good solutions [12,14]. How can we hope to allow an agent to learn complex tasks, without human guidance, and with relatively few examples?

The key difference between traditional RL settings and that of our hypothetical generally intelligent agent is that our agent would be able to leverage experience gained in previous related tasks. The idea of *lifelong learning* has been studied before in traditional machine learning [19] but it has only recently been applied to reinforcement learning, which is of specific interest to general intelligence.

Consider, for example, a generally intelligent household transportation assistant. While current agents are typically special purpose, a generally intelligent agent would be required to act in multiple related tasks. Such an agent may be expected to: retrieve food for a meal from the grocery store, drive the children to school, and retrieve the parents from work. All of these tasks utilize skills for operating a vehicle, obeying traffic laws, scheduling, and navigation. Due to this overlap, one would expect training on subsequent transportation tasks to take substantially less time than on the first such task. If the agent is tasked with a new request, such as delivering laundry to the cleaner, the users would expect the agent to quickly master the new task due to similarities with previously learned tasks. We next describe *transfer learning* for reinforcement learning, a general approach that would allow an RL agent to leverage past knowledge when learning novel tasks, potentially enabling effective lifelong learning for a generally intelligent agent.

In the transfer learning paradigm a learner is presented a novel *target task*. The agent may elect to first train on a (set of) simpler *source task(s)* rather than learning the target task directly. A typical goal of transfer is to reduce the time needed to learn a target task after first learning a source task, relative to learning the target task without transfer. This *target task goal* can be achieved whenever the learner can effectively transfer useful information from the source task into the target task. A more difficult goal is to reduce the total training time so that learning the source task and target task is faster than learning the target task directly. Such a *total time goal* is attainable only if the source task is faster to solve than the target task, and the speedup in target task training time overcomes the time spent on learning the source task.

Transfer between tasks has long been studied in humans [18] and the ability to transfer knowledge between different tasks is one reasonable criterion for intelligence. School curricula are designed around the principle of developing students' abilities and increasing the knowledge gradually over time. For successful autonomous transfer, an agent must effectively identify analogies between different tasks.[1] Hofstadter even argues [6] that analogical reasoning is the core of intelligence because humans form, over their lifetime, a *mental lexicon* of categories and information by using analogies.

In order to enable autonomous transfer, the agent must:

1. select a source task appropriate for a given target task,
2. effectively transfer knowledge from the source task into the target task.

While there has been recent work on step #2 [16,20], relatively little work has concentrated on the more difficult step of task selection. Such a selection ability will be necessary if the agent is to to determine which source tasks to train on before before tackling a more difficult target task, or if the agent has experienced multiple source tasks and it must select a subset which are most similar to the current target task.

In order to facilitate the selection of source tasks for a particular target task, this work introduces the concept of a *transfer hierarchy*. Such a structure defines types of tasks that require more or less information to solve and can be used to rank tasks by

[1] Other research [5] suggests that humans are not reliably proficient at discovering analogies between very dissimilar tasks unless prompted that such an analogy exists.

their relative solution complexity. Such a task ordering can be used to identify source tasks that will take significantly less time to solve than a particular target task, reducing the impact of source task training on the total training time. Our hope is that such a hierarchy will be useful in future work where transfer learners *automatically* select a source task for a given target task. In this paper we begin to evaluate the effectiveness of our proposed hierarchy by manually constructing source tasks for a specified target task where the selection of source tasks are motivated by the transfer hierarchy.

To empirically demonstrate transfer between source and target tasks taken from our transfer hierarchy, we utilize the game of *Mummy Maze*. This game is an appropriate choice for two reasons. First, it has been released as a sample domain in the *General Game Playing* [4] (GGP) contest, an international competition developed independently at Stanford. Second, the Mummy Maze task is easily modifiable so that it can conform to each task type in our transfer hierarchy. Our results show that a transferred heuristic is able to significantly improve the speed of search, even if the generated source tasks differ from the target tasks along a number of dimensions. We show both that transferred knowledge can effectively reduce the amount of time required to learn the target task, and that the total time required to learn the target task may be reduced by first training on a set of simpler source tasks. This result is a small but important step towards autonomous transfer in both planning and RL domains, which we believe to be on the critical path for development of a generally intelligent agent.

2. A Transfer Hierarchy

Mapping problem characteristics to the correct solution type is an important open problem for AI. Given a control problem, should the solution be solved optimally or approximately? Is planning or RL more appropriate? If RL, should the solution be model-based or model-free? This work assumes that such an appropriate mapping exists; given certain characteristics of a game, we propose an appropriate solution method. The characteristics we select are based on the amount of information provided to a player about the game's environment and opponent.

For instance, if a learner has a full model of the effects of actions and knows how its opponent will react in any situation, the learner may determine an optimal solution by "thinking" through the task using *dynamic programming* [2] (DP). At the other extreme, a learner may have to make decisions in a task where the opponent's behavior is initially unknown and possibly stochastic. In this more difficult scenario, the solution strategy must work to sample the environment and opponent's policy repeatedly, which suggests an RL approach.

Interactions with the environment and an opponent accrue cost: simulators use computational resources, physical robots may take significant amounts of wall-clock time, and opponents think before making decisions. When using DP, the only cost is cycles spent determining an optimal policy. When using RL, one must account for both interactions with the environment and opponent. By considering these differences in resource requirements, we propose a hierarchy to define game characteristics which require more resources to solve. We then leverage the solution hierarchy to find an appropriate type of source task to transfer from, given a target task.

Suppose that a learner could make some simplifying assumptions about a target game so that it could derive a simpler version of the task. For instance, in a 2-player maze task, the agent could generate a series of randomly constructed mazes, with some approximate model for the opponent's behavior. The source tasks could be solved very quickly using DP. When the "real" target mazes are presented, the learner should be able to leverage its source task knowledge to solve the target mazes more quickly than if it had not used transfer.

In this work, we consider two-player games set against a specific, fixed opponent. A game is defined as a set of states, a set of (possibly state-dependent) actions for each player, a reward function for each player, and a transition function that maps a state and the players' actions to a next state. To define the transfer hierarchy, we consider four characteristics of the game in question:

1. **Is the transition function known?** If the effect of actions are known, the learner may not have to interact with the environment to determine a good policy.
2. **Is the opponent's policy known?** Can the player anticipate the opponent's action in any state?
3. **Is the opponent *queriable*?** Is the opponent willing to answer the question, "What would you do in this state?" If so, we can assume that there is some cost to querying the opponent, but we may jump to different locations in a game tree rather than being forced to play each game from start to end.
4. **Is the opponent deterministic?** A stochastic policy must be sampled repeatedly while a deterministic policy need only be experienced once in each state. [2]

Given these task characteristics, we construct a hierarchy of solution methods in Figure 1. The method *Transition Learner* concentrates on only learning the effect of moves in the given task since the opponent's policy is completely known. It is difficult to imagine such a scenario where the opponent's strategy is defined but the learner does not know the transition model (none of the games commonly played by humans fall into this category). Another less familiar solution method is *Active RL* [11]. In this scenario the learner uses reinforcement learning, but may focus on sections of the state space with the most uncertainty.

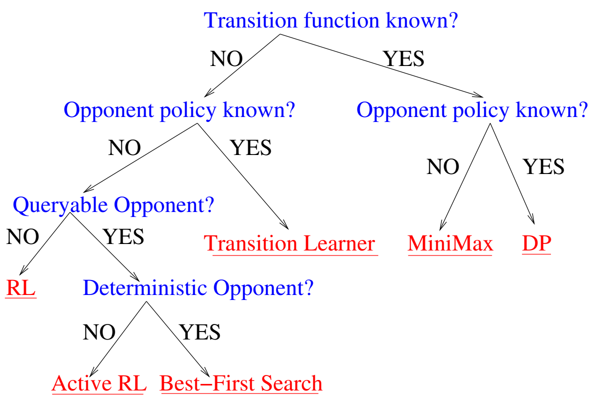

Figure 1. Characteristics of a given task define which solution method is most appropriate. More knowledge leads to solution methods which require fewer interactions with the environment and/or opponent.

In addition to mapping task characteristics to possible solution methods, Figure 1 also defines a *Transfer Hierarchy*. Learners that have more information are able to solve tasks with fewer environmental or opponent interactions. Given a target task with little information, the learner may be able to generate similar tasks but give the learner more information. A central hypothesis for this work is that a learner may train relatively quickly on a simpler source task and then use its learned information to speed up learning

[2]We do not consider non-stationary (e.g., learning) opponents and leave this extension to future work.

the target task which must use a more complex solution method (i.e., one to the left of the source task method which has less information available to the learner). In this paper we empirically test one such pairing: we first learn a series of constructed source tasks via DP to speed up learning a target task via best-first search.

3. Case Study: Transfer in a 2-player Game

To test our transfer hypothesis we utilize the Mummy Maze task, one of many games simulated in General Game Playing. Specifically, we will focus on a target task where the maze is unknown, the opponent's policy is unknown, and the opponent is both queriable and deterministic. To speed up learning this task using best-first search (as described in Section 3.3.2), we first construct a series of source mazes and a test opponent, which are solvable with DP.

3.1. General Game Playing

Creating programs that can play games at a high level has long been a challenge and benchmark for AI. However, traditional game playing systems are limited in that they play only one particular game. In contrast, the GGP challenge motivates research on creating agents capable of playing many previously unseen games, given only a description of the game's rules. Since 2005, AAAI has held an annual GGP competition in which agents designed by different researchers compete on a wide variety of games.

In the Game Description Language (GDL) used in the competition, games are modeled as state machines. An agent can derive its legal moves, the next state given the moves of all players, and whether or not it has won by applying resolution theorem proving on the rules of the game combined with the asserted facts for the present state. The language is fairly low-level and is able to describe multiplayer, deterministic, perfect-information games. Syntactically, GDL is a first-order logical description language based on KIF [3]. The next section introduces the game, described in GDL, used in our experimental work.

3.2. Mummy Maze

Mummy Maze[3] is a game in which the *explorer* attempts to escape a maze. The opponent *mummy* follows a fixed policy to attempt to stop the explorer. The explorer has 5 deterministic actions: moving one step in each of the four cardinal directions {N, S, E, W} or standing still. The mummy has the same action set, but takes *two* serial actions on each turn. The explorer and mummy alternate moves and neither may transition through walls. The challenge for the explorer is to exploit the mummy's fixed policy so that he may reach the exit despite the speed disadvantage. The explorer receives a reward of +100 if he reaches the exit and a reward of 0 if the mummy catches the explorer or if the explorer has taken some maximum number of turns (typically 50) without escaping.

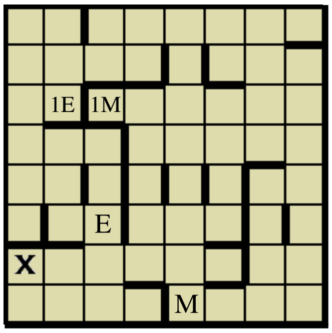

Figure 2. An example solution to a maze with vertical mummy behavior. The explorer moves directly to the 1E space and the mummy is trapped at 1M, allowing the explorer to double back to the exit, denoted by an 'X'.

A mummy following the *vertical behavior* policy will deterministically move towards the ex-

[3]The .kif file which fully describes the game in GDL may be found at `http://games.stanford.edu/gamemaster/games-mummy/mummymaze1p-horiz.kif`

plorer on every move, preferring vertical moves over horizontal moves when both types of move would reduce the players' distance. Figure 2 shows an example maze, with the solution for the explorer. As the explorer moves to the grid location 1E, the mummy moves North on each move until it moves West and becomes trapped at 1M. Once the mummy is trapped in the cul-de-sac, because it never moves away from the explorer, the explorer may proceed South to the exit. A mummy that follows the *horizontal behavior* policy prefers to move East or West towards the explorer if possible. Figure 3 demonstrates how the explorer's policy must change to exploit this mummy policy, given the same wall configuration, start state, and goal state. Notice that if the explorer attempted the previous solution path, the mummy would catch the explorer at the cell marked by the red circle.

Mummy Maze is an appropriate choice for this work because we can easily adjust the game definition so that each of the solutions described in the transfer hierarchy is appropriate. For instance, if the explorer is not told where the walls are located, the mummy's policy is unknown, and the mummy is not queriable, RL would be the most appropriate solution strategy. The next section discusses Mummy Maze formulations where DP and best-first search are applicable.

3.3. Mummy Maze Solution Methods

A number of strategies may be employed to solve Mummy Maze, depending on the amount of information the explorer has. In this paper we consider two cases:

1. The transition function is known (i.e. the placement of all the walls in the maze is known) and the opponent's policy is known.
2. The transition function and opponent's policy are unknown, and the opponent is both queriable (i.e. the explorer can ask the mummy, "If I were here and you were there, how would you act?") and deterministic.

In the following sections we explain how Mummy Maze tasks can be solved with dynamic programming, with best-first search, and with transfer from dynamic programming to best-first search.

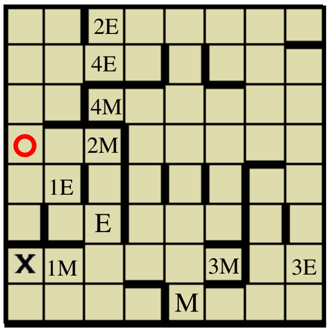

Figure 3. A solution for horizontal mummy behavior. If the explorer attempted the previous solution, the mummy would catch the explorer at the red circle. The explorer must move to squares 1E-4E, trapping the mummy at squares 1M-4M, before exiting.

3.3.1. Dynamic Programming

In its original construction, Mummy Maze is a single player puzzle game, in which the mummy is controlled by a known deterministic policy, specified as part of the environment. Given a task in which the transition function and opponent behavior are deterministic and known, the optimal agent policy may be found by simply enumerating all of the game's states and transitions between them. Such a problem may be solved with dynamic programming.

The dynamic programming algorithm begins by enumerating all states in the game's state set, S. All terminal states are marked as either wins or losses, based on the game's description. Then, all non-terminal states that transition to a terminal state are marked. Any action leading to a win is a win. If all actions lead to a loss, then the originating state is a loss. The iteration continues, marking states that transition to states marked in the previous iteration. One can recover the policy for the solution by simply adding some extra bookkeeping to record the winning transitions between states.

DP is able to find the optimal solution from all possible initial states for a given goal state. Although the algorithm is very generally applicable, it is only practical on games with reasonably small state spaces. The running time of the algorithm is $O(l^*|A||S|)$ where l^* is the longest solution length, in steps, and A is the set of actions available to the agent. For Mummy Maze on an 8×8 grid, $l^*|A||S| = 50 \times 5 \times 64^2$, which is only roughly one million evaluations.

3.3.2. Best-First Search

In the second variation of Mummy Maze this paper considers, we utilize a search to determine a (possibly sub-optimal) solution to a given maze, if one exists. We utilize a learned heuristic (as specified in the next section) to perform greedy best-first search. If we do not use a heuristic, best-first search reduces to breadth-first search.

We modified the standard best-first search algorithm in a subtle but important way to incorporate domain knowledge. In Mummy Maze, a solution can be broken down into a series of subgoals, each of which trap the mummy and allow the explorer to move to another location. We capture this knowledge by prioritizing a state not solely by its heuristic value, but by the sum of the values of its ancestors. States with high heuristic values are likely subgoals and thus search is guided to explore the children of states that encounter subgoal states along the way.

In the worst case, best-first search must expand the entire game tree. Thus, its running time is proportional to the number of states in the game. Although the computational complexity of the algorithm is less than that of Dynamic Programming, it has a significantly higher constant factor. In each state it must query the opponent for their move, which is an expensive operation.

3.4. Transfer Methodology

In this paper we concentrate on learning a search heuristic for best-first search by solving one or more source tasks with dynamic programming. In this section we discuss how to construct a search heuristic from source task solutions. In the following section, we empirically verify that such a heuristic can speed up search in the target task, even if the source task and target task differ in wall configuration, opponent behavior, size, start state, or goal state.

The main insight for heuristic learning is that rather than learn a heuristic for a *particular* source task, that is one for a particular maze, we learn over a state abstraction. For this task, we chose an abstract representation centered on the Mummy which considers the walls adjacent to it and the direction from the mummy to the explorer. The intuition is as follows. A state where the mummy is in a corner or in a cul-de-sac and the explorer is on the opposite side of the wall is a relatively good position for the explorer. On the other hand, a state where the mummy is in an open area with no walls is less desirable for the explorer because the mummy has a high degree of mobility. In this simple abstraction there is no notion of distance between the mummy and explorer, nor between the explorer and the exit.

We use a function GETABSTRACTSTATE which takes the current board configuration and returns the index for the mummy's current abstract state. There are 15 possible wall configurations for the walls directly adjacent to the mummy.[4] There are 8 possible directions from the mummy to the explorer, which yields 128 possible abstract states, while a standard 8×8 game has 4,096 true states (64 explorer positions $\times$ 64 mummy positions). Although this abstraction is hand coded, we would ideally like to use automated abstractions (e.g., Jong and Stone [7]) in the future.

After solving a source task, the number of wins and losses for each abstract state is tallied. The win percentage $\left(\frac{\#\ wins}{(\#\ wins)+(\#\ losses)}\right)$ for each abstract state is calculated, as well as the average win percentage and the standard deviation. When calculating the heuristic for a state in the target task, we first find the corresponding abstract state. If *winPercentage* $\geq$ *aveWinPercentage* + *stDev* then the heuristic returns +1. If *winPercentage* $\leq$ *aveWinPercentage* - *stDev* then the heuristic returns -1. Otherwise the heuristic returns 0. [5]

4. Case Study: Transfer Results

To test our transfer methodology we perform a number of experiments in which the source and target tasks have different characteristics. In every experiment we construct a set of target tasks and record how many steps the best-first search takes to solve the task with and without transfer. In this setup, the "steps taken" is equivalent to how many times the Explorer must ask the Mummy, "What action would you take in this state?" This is equivalent to the number of connections the Explorer agent must make to the GGP server to query for the opponent's move. Each target maze is solved 10 times as the best-first search breaks ties randomly. Roughly 25% of the mazes constructed have *no* solution because of the start state and/or wall configuration. Impossible tasks are ignored in the evaluation as no search method could possibly find a solution.

When using transfer, the source task mazes are randomly generated using the same wall-generation algorithm that the target tasks are generated with and thus the mazes in the source and task are drawn from the same distribution of possible mazes. However, because the opponent behavior is different in the two sets of tasks, the distributions of source and target tasks are qualitatively different. [6] All source task mazes have the same start state and goal state, as depicted in Figure 2. Additionally, all source tasks utilize a horizontal mummy behavior.

4.1. Different Opponent Behavior

All transfer experiments in this paper utilize different mummy behaviors in the source and target tasks. As stated above, the source tasks all use a horizontal behavior Mummy. In the target task the Mummy uses a deterministic mixture of the horizontal and vertical behaviors, denoted *HV-behavior*. HV-behavior specifies that the mummy utilize horizontal behavior if its x and y cell coordinates have the same parity (both are even or both are odd) and act like a vertical mummy if the parity of its x and y cell coordinates are

[4] We do not allow a cell to be surrounded by four walls as it would be unreachable.

[5] Rather than using the *winPercentage* directly as heuristic values, which would tend to explore the states with the highest individual values first, we instead cluster states into three categories: good, neutral, and bad. By doing so, the priority of a state during best-first search is dominated by the number of good states in its history rather than by how good those states are independent of their history. We intend to explore using the continuous version of this heuristic in future work.

[6] If the learner had access to the target task mazes and trained on them, rather than using random mazes for the source task, transfer could be trivially accomplished by memorizing the solution to each maze.

different. Thus the mummy's behavior is deterministic but is qualitatively different from the source task's mummy behavior.

To evaluate experiments in this domain, we define *transfer percentage* to be the ratio between the total number of steps to solve all mazes with and without transfer:

$$100 \times \frac{\sum_{TargetMazes}(\texttt{Steps to solve maze with transfer})}{\sum_{TargetMazes}(\texttt{Steps to solve maze without transfer})}.$$

To test transfer between source tasks with a horizontal behavior mummy and target tasks with an HV-behavior mummy, we first generate 200 HV-behavior target task mazes. Each is solved 10 times without transfer. Next, 20 horizontal-behavior source tasks are analyzed and the learned heuristic is used to solve each target task 10 times with transfer. We find that the transfer percentage is 73, which means that, on average, using transfer results in a 27% reduction in the number of queries the explorer must make of the GGP server. As may be expected, we found that more difficult target tasks (those requiring relatively more steps to solve) benefited more from transfer on average.

4.2. Different Numbers of Source Tasks

To test the effect of the number of source tasks on transfer, we ran experiments with different numbers of source tasks. The results are reported in Table 1, which shows that even with a very small number of source tasks, transfer can significantly reduce the number of steps needed to solve target tasks.

4.3. Comparison to a Simple Hand-coded Heuristic

In order to better evaluate our learned heuristic, we compared our results to those generated from a simple hand-coded heuristic. If the mummy was able to move in every direction we labeled the state as *bad* and if the mummy was unable to move towards the explorer the state was *good*. Using this metric we observed a transfer percentage of 75, which our learned heuristics either tied or beat (unless fewer than 3 source tasks are used). This suggests that our algorithm is not only able to learn a heuristic autonomously, but that the learned heuristic captures more useful information than a simple hand-coded heuristic.

# Source Tasks	Transfer Percentage
1	97
2	79
3	74
5	73
10	75
20	73
50	71
100	70
200	71
400	73

Table 1. Results show significant transfer benefit, even with few source tasks.

4.4. Different Target Task Sizes

The 10×10 maze has 10,000 unique states and we expected that our transfer percentage would improve when solving larger target mazes. To test this theory, we again generated 20 8×8 source task mazes, but the target task mazes were 10×10. We found the resulting transfer percentage to be 66, a slight improvement over 73.

4.5. Different Start State

Up to this point all source and target tasks have been generated such that the mummy and explorer always began at the same coordinates and the exit was in the same location. To evaluate how dependent our method was on the start state, we kept the source task start state fixed but allowed the target task start state to be chosen randomly. We found that the transfer percentage was effectively unchanged, as it now averaged 69 (as compared to 73 when the target tasks' start states were fixed).

4.6. *Different Start State and Goal State*

We next allowed both the start state and the exit to vary in the target tasks. Our setup allowed the exit to be anywhere on the board, which resulted an average transfer percentage 92. We hypothesized this was because our abstract states did not account for relative placement of the exit. Thus our heuristic learned a bias that favored the explorer's mobility towards the Southwest corner of the board in source tasks and when the exit was in a different location, this bias was less helpful (although it was still better than searching without a heuristic). To test this, we then allowed random start states and exit positions in the target tasks, but constrained the exit to be in the Southwest quadrant of the board (thus reducing the number of possible exit locations by a factor of four). With the bias now restored, the resulting transfer percentage was 70. This and other experiments are summarized in Table 2.

Target Task Size	Hand-coded Heuristic?	Target Task Random Start State?	Target Task Random Goal State?	Transfer Percentage
8×8	Yes	No	No	75
8×8	No	No	No	73
10×10	No	No	No	66
8×8	No	Yes	No	69
8×8	No	Yes	Yes (anywhere)	92
8×8	No	Yes	Yes (SW quadrant)	70

Table 2. These results summarize a comparison of searching without transfer to searching after analyzing 20 source tasks. All source tasks are 8×8 with H mummy behavior, with fixed start and goal states. Results are averaged over 200 target tasks with HV-behavior mummy behavior.

4.7. *Total Time Metric*

In order to demonstrate a reduction in the total time we must measure both the time used to solve the source tasks with dynamic programming as well as the time used to solve target tasks solved with best-first search. When using dynamic programming, the solution time is determined by the time to simulate taking an action in the environment and then simulating the opponent's action: *num next states simulated* $\times$ *internal next state time*. When solving a task with best-first search, the learner must query the central GGP server for each next state because the learner does not have the transition function. Furthermore, the GGP server must query the opponent to determine its action for a given state. Solving a target task is dominated by the communication delays and opponent's response time: *num search steps* $\times$ $(4 \times$ *communication time* $+$ *opponent response time*$)$.

When connecting to the Stanford Game Manager, the time to compute the next state is about 0.1 seconds, the average communication time with the remote server. Using our own python inference engine, we can simulate an average of 5.51×10^4 next states per second on a 3.4 GHz machine. We assume that the opponent responds in one second (which is much faster than is typical in GGP competitions).

Table 3 shows that the transfer percentage increases for larger target tasks. Additionally, we compare the average number of seconds it takes to solve a target task without transfer (breadth-first search) with the total time needed to solve the source tasks and a target task (best-first search). Such an analysis demonstrates that it is likely when using larger target tasks, or if the opponent takes some time to choose its move, total time can be reduced by using the transfer hierarchy to select source tasks. Transfer requires solving extra source tasks, but the speed-up achieved in the target task may outweigh this initial overhead; the last experiment in Table 3 shows a total time reduction of 19%.

Target Task Size	Ave. Target Task Time (no Transfer)	# Source Tasks	Transfer Percentage	Ave. Total Source Task Time	Ave. Total Time (with Transfer)
8×8	178	20	73	372	502
10×10	400	20	66	372	636
12×12	563	20	47	372	657
12×12	563	10	53	186	461

Table 3. Summary of results comparing searching without transfer to searching after 20 source tasks (times are in seconds). All source tasks are 8×8 with an H-behavior mummy, fixed start and goal states. Results are averaged over 200 target tasks with an HV-behavior mummy.

5. Future and Related Work

In this work the opponents in the source and target tasks have slightly different policies. In preliminary experiments there were not qualitatively different results when using identical policies (horizontal behavior to horizontal behavior) or more dissimilar policies (horizontal behavior to vertical behavior). We speculate that this is because all of these policies are similar enough that transfer can provide a useful heuristic.

One direction for future work would be to consider more dissimilar opponent policies, such as a Mummy that could escape from a cul-de-sac with a certain probability. The abstract state representation could also be enhanced in future work, and ideally would be learned automatically. Likewise, rather than using the transfer hierarchy to selecting a type of source tasks for a given target task, it should be possible to have a TL learner use the hierarchy to *automatically* construct a source task, given a target task. Testing more source and target task pairings would further validate the proposed transfer hierarchy.

While this work has focused on determining how to select a type of source task for a particular target task, we have not addressed what properties a source task should have to best assist learning a target task. For instance, if the transfer hierarchy directs the agent to learn a source task with DP, how can the agent ensure with high probability that the sample tasks it constructs are not misleading? If a generally intelligent agent is to transfer successfully in a fully autonomous setting, it should be able to reliably construct, or select, source tasks that do not cause *negative transfer* so that it avoids the situation where transfer hurts performance, rather than helps.

The main novelty of the experiments in this paper is to present a method for heuristic learning via transfer learning. There is a growing body of work using transfer learning to learn tasks sequentially. For instance, our previous work [16] showed that it was possible to transfer a value function between related reinforcement learning tasks. Other work showed that it is possible to speed up learning between related tasks via advice [20]. GGP tasks have also been used successfully for previous transfer work [1,10].

Prior planning research has demonstrated the possibility of generating state-space abstractions automatically from domain descriptions. These methods may be divided into two forms. In *relaxed models* [13], abstractions are obtained by dropping conditions of actions to make them applicable in more states. A different approach is to generate a *reduced model* [9], in which certain terms are dropped entirely from the problem description. Although neither of these methods could produce our particular abstraction, it is possible that, if applied to Mummy Maze, they may yield different useful abstractions.

6. Conclusion

In this paper we have argued that transfer learning is a critical component of any intelligent system. Transfer learning, particularly in a RL context, has recently been growing in popularity due to empirical successes demonstrating significant speed improvements. We have introduced a transfer hierarchy which assists in selecting a type of source task for transfer, given a specified target task. Additionally, we have demonstrated that transfer between two such tasks types is able to both reduce the target task training time and the total training time in a game drawn from the GGP domain. We view this work as one small, but important, step towards the lofty goal of enabling human-level knowledge transfer, a critical component of any generally intelligent agent.

Acknowledgments

We would like to thank the anonymous reviewers for helpful suggestions. This research was supported in part by DARPA grant HR0011-04-1-0035, NSF CAREER award IIS-0237699, and NSF award EIA-0303609.

References

[1] B. Banerjee and P. Stone. General game learning using knowledge transfer. In *The 20th International Joint Conference on Artificial Intelligence*, pages 672–677, January 2007.

[2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[3] M. Genesereth. Knowledge interchange format. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second Intl. Conference (KR'91)*, 1991.

[4] M. Genesereth and N. Love. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2), 2005.

[5] M. L. Gick and K. J. Holyoak. Analogical problem-solving. *Cognitive Psychology*, 12:306–355, 1980.

[6] D. Hofstadter. Analogy as the core of cognition. In D. Gentner, K. J. Holyoak, and B. Kokinov, editors, *The Analogical Mind: Perspectives from Cognitive Science*, pages 499–533. MIT Press, 2001.

[7] N. K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 752–757, August 2005.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[9] C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[10] G. Kuhlmann and P. Stone. Graph-based domain mapping for transfer learning in general games. In *Proceedings of The Eighteenth European Conference on Machine Learning*, September 2007.

[11] L. Mihalkova and R. Mooney. Using active relocation to aid reinforcement learning. In *Proceedings of the 19th International FLAIRS Conference*, pages 580–585, 2006.

[12] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[13] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[14] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[15] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[16] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[17] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[18] E. Thorndike and R. Woodworth. The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review*, 8:247–261, 1901.

[19] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646, 1996.

[20] L. Torrey, T. Walker, J. W. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *The 16th European Conf. on Machine Learning*, 2005.