In *ICAPS-07 Workshop on AI Planning and Learning* (AIPL-07),
Providence, RI, September 2007.

Accelerating Search with Transferred Heuristics

Matthew E. Taylor, Gregory Kuhlmann, and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{mtaylor, kuhlmann, pstone}@cs.utexas.edu

Abstract

A common goal for *transfer learning* research is to show that a learner can solve a *source task* and then leverage the learned knowledge to solve a *target task* faster than if it had learned the target task directly. A more difficult goal is to reduce the total training time so that learning the source task and target task is faster than learning only the target task. This paper addresses the second goal by proposing a *transfer hierarchy* for 2-player games. Such a hierarchy orders games in terms of relative solution difficulty and can be used to select source tasks that are faster to learn than a given target task. We empirically test transfer between two types of tasks in the *General Game Playing* domain, the testbed for an international competition developed at Stanford. Our results show that transferring learned search heuristics from tasks in different parts of the hierarchy can significantly speed up search even when the source and target tasks differ along a number of important dimensions.

Introduction

> If you cannot solve the proposed problem try to solve first some related problem. Could you imagine a more accessible related problem? ...Could you solve a part of the problem? (Polya 1945, p. xvii)

Polya's 1945 book *How To Solve It* motivates the general principle behind *transfer learning* (TL). In this machine learning paradigm, a learner first solves a *source task* and then uses its knowledge to solve a *target task*. Rather than learning a difficult target task directly, consider the following three-step TL process:

1. The learner must find or construct a source task that is relevant to, but simpler than, a target task. Full details of the specific target task may or may not be available during this phase.
2. The learner must solve the simple source task with relatively little effort compared to solving the full target task.
3. The learner must transfer the knowledge gained from the source task and utilize it to solve the target task.

A typical goal in TL research 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 transfer useful information from the source task into the target task. The majority of TL research to date has focused on this goal (step # 3), demonstrating both the feasibility of transfer and the many dimensions along which the source and target may differ while still allowing transfer. In these TL scenarios the relevant source task or tasks are generally provided to the learner for each 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. The *total time goal* is attainable only if the source task (called an *auxiliary problem* by Polya) 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. To achieve this goal the learner must reason about all three steps. This paper takes a first step at the difficult problem of discovering appropriate source tasks by proposing a *transfer hierarchy*. Such a structure defines types of games that require more or less information to solve and thus may be used to order tasks by their relative solution complexity.

Such an 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. In the future we hope that such a transfer hierarchy will be used to help automate the transfer learning process by assisting in the selection of 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 task are motivated by the transfer hierarchy.

To empirically demonstrate transfer between source and target task 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* (Genesereth & Love 2005) (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 improve the speed of search by as much as 34%, meeting the target time goal, even if our source tasks differ from the target tasks along a number of dimensions. Additionally, we demonstrate how the total training time goal may also be met for this particular pair of source and target types, depending on information gathering costs.

A Transfer Hierarchy for Games

Mapping problem characteristics to the correct solution type is an important open problem for AI. For instance, given a control problem, should the solution be solved optimally or approximately? Is planning or reinforcement learning (Sutton & Barto 1998) (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* (Bellman 1957) (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 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 *queryable*?** 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. [1]

Given these task characteristics, we construct a hierarchy of solution methods in Figure 1a. 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* (Mihalkova & Mooney 2006). In this scenario the learner uses reinforcement learning, but may focus on sections of the MDP with the most uncertainty.

In addition to mapping task characteristics to possible solution methods, Figure 1a 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 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.

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

Test Domain

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

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 General Game Playing (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 (Genesereth 1991). The next section introduces the game used in our experimental work, which is described in GDL.

Mummy Maze

Mummy Maze[2] is a single player 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 staying put. 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 50 turns without reaching the exit.

A mummy following the *vertical behavior* policy will deterministically move towards the explorer on every move, preferring vertical moves over horizontal moves when both types of move would reduce the players' distance. Figure 1b shows an example maze, with the solution for the explorer. As the explorer moves to the grid

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

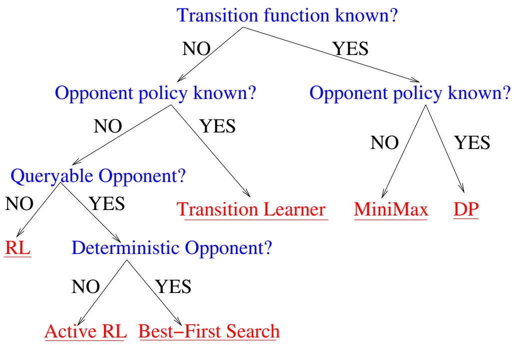

Figure 1a: Characteristics of a given task define which solution method is most appropriate. More knowledge leads to solution methods which require fewer interactions with the environment and/or opponent.

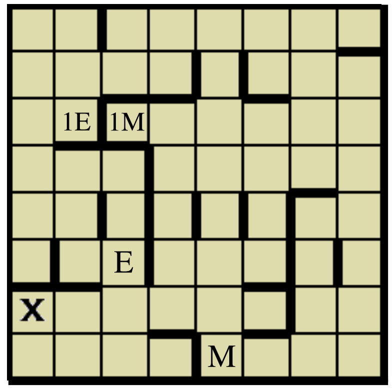

Figure 1b: This figure shows an example solution to a maze with vertical mummy behavior. The explorer moves directly to the 1E space and the mummy is trapped at 1M, allowing the explorer to double back to the exit, denoted by an 'X'.

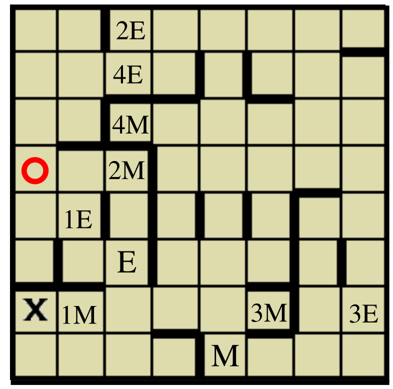

Figure 1c: Solving the maze with horizontal mummy behavior. If the explorer attempted the previous solution, the mummy would catch the explorer at the red circle. The explorer must move to squares 1E-4E, trapping the mummy at squares 1M-4M, before exiting the maze.

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 1c 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 queryable, RL would be the most appropriate solution strategy. The next section discusses Mummy Maze formulations where DP and best-first search are applicable.

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 queryable (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.

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 such 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. This instance of the general dynamic programming algorithm is detailed as Algorithm 1.

The algorithm begins by enumerating all states in the game's state set, S. The $goal(s)$ is the value of a state as determined by the goal conditions described in the game's GDL description. All terminal states are marked as either wins or losses, based on this value. Then, all non-terminal states that transition to a terminal state are marked. The set $next(s)$ contains all states resulting from legal explorer moves taken in state s. 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. The **repeat** loop terminates after marking the states with the longest solution lengths. One can recover the policy for the solution by simply adding some extra bookkeeping to record the winning transitions between states.

Algorithm 1 DYNAMICPROGRAMMING (maze)

for all states $s \in S$ **do**
  if s is terminal **then**
    mark s a WIN **if** *goal(s) = 100*
    mark s a LOSS **if** *goal(s) = 0*
repeat
  for all unmarked states s **do**
    $w \leftarrow 0, l \leftarrow 0$
    for all states $s' \in$*next(s)* **do**
      $w \leftarrow w + 1$ **if** s' marked as a WIN
      $l \leftarrow l + 1$ **if** s' marked as a LOSS
    if $w > 0$ **then**
      mark s a WIN
    else if $l = |next(s)|$ **then**
      mark s a LOSS
until no new states marked

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, $l^*|A||S| = 50 \times 5 \times 64^2$, which is only about one million evaluations.

Best-First Search In the second variation of Mummy Maze this paper considers, we utilize a search to determine a (possibly suboptimal) 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 as specified in Algorithm 2. 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 (line 13). 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.

Algorithm 2 BESTFIRSTSEARCH (maze, heuristic)

1: $Q \leftarrow$ empty priority queue
2: add (initial state of maze, 0) to Q
3: **while** Q is not empty **do**
4: $\quad(s, priority) \leftarrow$ get highest priority element of Q
5: $\quad$**for all** actions $a \in$ explorer's legal moves in state s **do**
6: $\quad\quad s' \leftarrow$ NEXTSTATE(s, a)
7: $\quad\quad a' \leftarrow$ ask mummy what move will be taken in state s'
8: $\quad\quad s'' \leftarrow$ NEXTSTATE(s', a')
9: $\quad\quad$**if** ISWINNINGSTATE(s'') **then**
10: $\quad\quad\quad$**return** (solution found)
11: $\quad\quad$**else if** not ISLOSINGSTATE(s'') **then**
12: $\quad\quad\quad \delta \leftarrow$ heuristic(s'')
13: $\quad\quad\quad$add (s'', $priority + \delta$) to Q
14: **return** (NO solution found)

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[3]. 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 (2005)) in the future.

After solving a source task, the number of wins and losses for each abstract state is tallied. The win percentage ($\frac{\#\,wins}{\#\,wins + \#\,losses}$) 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. [4]

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?" Alternatively, 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 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. [5] All source task mazes have the same start state and goal state, as depicted in Figure 1b. Additionally, all source tasks utilize a horizontal mummy behavior.

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

[3] We do not allow a cell to be surrounded by four walls (i.e. to be unreachable).

[4] 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.

[5] 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 benefited more from transfer on average.

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 1a, 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.

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 any 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.

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.

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).

Different Start State and Goal State

Our final test 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, 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, 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 was 70. This and other experiments are summarized in Table 1b.

Total Time Metric

The results in this paper have focused on the target task metric, as the transfer percentage we report ignores time spent solving the source tasks. However, as suggested previously, this transfer method may also reduce the total time because the source and target are selected from different task types in our transfer hierarchy. Our first experiment solves an 8×8 target task taking an average of 127 best-first search steps to solve without transfer. On each step the learner must query the central GGP server for the 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 delay and opponent's response time: $127 \times (4 \times (\textit{communication time}) + (\textit{opponent response time}))$.

The transfer learner first completely solves 20 source tasks taking roughly 20.5 million evaluations. The time for solving the source tasks is bounded by the time to simulate taking an action in the environment and then simulating the opponent's action: $20.5 \times 10^5 \times (\textit{internal next state time})$. The average solution length of an 8×8 maze is 93 steps after transfer from 20 source tasks. Thus the total time for solving a target task with transfer is $20.5 \times 10^5 \times (\textit{internal next state time}) + 93 \times (4 \times (\textit{communication time} + \textit{opponent response time}))$.

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 inference engine, we can simulate an average of 5.51×10^4 next states per second on a 3.4 GHz machine. Assuming the opponent responds *instantly*, transfer would reduce the total time if DP could simulate 1.52×10^6 next states per second, which may be possible if we use a compiled implementation of the domain to avoid the need for inference.

Solving a 12×12 maze takes an average of 370 steps without transfer and 197 steps after transferring from ten 8×8 source tasks, on average. Thus, DP needs to simulate only 1.5×10^4 next states per second to reduce the total time for a 12×12 target task, which our current implementation achieves. 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 overcome this overhead.

Future and Related Work

In this work we concentrate on tasks where the source task opponent and target task opponent 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

# 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 1a: Results show significant transfer benefit, even with few source tasks.

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 1b: Summary of results comparing 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.

for future work would be to consider more dissimilar policies, such as an opponent policy that allowed the mummy to escape from a cul-de-sac with a certain probability.

The abstract state representation could also be enhanced in future work. For instance, in the Mummy Maze domain it may make sense to explicitly include the direction from the explorer to the goal, the distance from the mummy to the explorer, or the explorer to the goal. Additionally, such an abstraction would ideally be learned automatically rather than hand-coded. 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.

Other future work involves utilizing the proposed transfer hierarchy for different combinations of tasks. For instance, we have tried using a heuristic learned with DP to speed up RL in Mummy Maze, but thus far do not have positive transfer results. The intuition we have attempted to leverage is that RL must use some exploration strategy and a learned heuristic should be able to guide the learner more efficiently than random exploration. We hope that such an approach will represent a second method for leveraging different types of tasks within the transfer hierarchy to reduce learning times.

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 sequentially presented tasks faster. For instance, our previous work (Taylor, Stone, & Liu 2005) showed that it was possible to transfer a value function between related reinforcement learning tasks.

The method of constructing different artificial source tasks is similar in spirit to the *Randomized Task Perturbation* method presented by Sherstov and Stone (Sherstov & Stone 2005) which adds noise into random states in the source task value functions so that source task learning is made more general. Other work by Stone and Veloso (Stone & Veloso 1994) discusses a related problem in a planning context. This paper addresses the situation where a planner fails to solve a difficult problem but a simpler auxiliary problem can be automatically constructed such that their solutions may guide the planner on the full problem.

Although the state abstraction in our work was created manually, 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* (Sacerdoti 1974), abstractions are obtained by dropping conditions of actions to make them applicable in more states. A different approach is to generate a *reduced model* (Knoblock 1994), 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. Generating abstractions automatically for use in our proposed transfer hierarchy is an interesting area of future work.

Conclusion

This paper has proposed a transfer hierarchy which suggests appropriate pairs of tasks for transfer such that the total time needed to solve a task may be reduced. Such a hierarchy is based on characteristics of the tasks with the assumption that when given a difficult task, a simpler task with similar characteristics may be approximated. We have also empirically demonstrated transfer between one such pair of tasks, showing that a useful search heuristic may be learned and then utilized to speed up a later task, even when the source tasks have qualitatively different characteristics from the target tasks.

Acknowledgments

We would like to thank Kristen Grauman and the anonymous reviewers for helpful comments and 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

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

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

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

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

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

Mihalkova, L., and Mooney, R. 2006. Using active relocation to aid reinforcement learning. In *Proc. of the 19th International FLAIRS Conference*.

Polya, G. 1945. *How to Solve It*. Princeton, NJ: Princeton University Press.

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

Sherstov, A. A., and Stone, P. 2005. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentyeth National Conference on Artificial Intelligence*.

Stone, P., and Veloso, M. 1994. Learning to solve complex planning problems: Finding useful auxiliary problems. In *Technical Report of the AAAI 1994 Fall Symposium on Planning and Learning: On to Real Applications*.

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

Taylor, M. E.; Stone, P.; and Liu, Y. 2005. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.