Center for Cognitive Architecture
University of Michigan
2260 Hayward St
Ann Arbor, Michigan 48109-2121

TECHNICAL REPORT
CCA-TR-2009-04

Transferring Learned Search Heuristics

Investigators
Joseph Xu
Samuel Wintermute
Yongjia Wang
John Laird

9 November 2009

1. Abstract

We study transfer learning in the context of single instance learning and single instance performance on a wide variety of discrete space/time, deterministic, single player games for which our agent has minimal amounts of background knowledge. We approach this as a problem of using knowledge extracted from source games to design search heuristics for use with an iterative deepening algorithm to solve target games faster than just uninformed iterative deepening. Our transfer algorithm relies on invariances in source and target game structure but also accounts for some possible differences in the encoding of game rules by explicitly mapping source game rules and concepts onto those in the target. Our test scenarios include both near (within problem domain) and far (across different problem domains) transfer cases, and we demonstrate significant amounts of transfer for both. Post-hoc analysis of the test results suggest that the successful transfer was due to taking advantage of both intended and unintended source and target invariances.

2. Introduction

Transfer learning refers to applying knowledge learned in one context (the *source)* to another (the *target*), either to improve performance or future learning. Transfer can be realized in multiple ways. If the source and target share some identical structures, then learned knowledge about those structures can transfer directly. Alternatively, if the learning is able to induce general knowledge structures from the source, those structures can be instantiated in the target. Finally, there may exist some mapping between the structures in the source and target that can be exploited to explicitly transform (map) the knowledge learned in the source so that it applies in the target.

Most previous transfer learning research has concentrated on a single type of transfer between problems in a single domain, with little restriction on the available background knowledge that aids generalization and finding mappings between the source and target. In this project, we investigated general transfer learning, where there is minimal background knowledge, where there are multiple types of transfers, and where the same transfer learning mechanism applies within multiple domains, as well as across domains. In order to achieve generality in the face of limited background, we resort to weak methods, both for problem solving and learning, that lead to surprisingly good transfer across the four evaluation domains.

3. Transfer Characterization

In this section, we characterize the types of transfer we studied. Transfer can vary in the number of source problems that are available for learning, as well as whether all source problems are relevant to the target. Moreover, transfer can vary in the number of target problems that are used to evaluate performance. In this research, we focus on single source, single target transfer scenarios, where transfer must immediately improve performance.

Transfer Levels

The simplest transfer is from a problem to another instance of the exact same problem. Transfer becomes more challenging as the target problem differs from the source in different ways. In this paper, we focus on the following variations (DARPA 2005); the numbering of the levels from 6 to 11 is inherited from a larger project:

- Composition (6): The system first solves two source games, and then solves a target game that combines aspects from both source games.

- Abstraction (7): At some level of abstraction, the source and target games have similar concepts.
- Generalization (8): The target game is a less constrained version of the source.
- Reformulation (9): The symbols in the source game are scrambled so that the target game only preserves the structure of the source.
- Low Differing (10): The source and target games are from different domains, but share some common parts.
- High Differing (11): Same as low differing, but in addition all symbols are scrambled as in reformulation.

Given our goal of generality, our system is not informed as to the type of transfer being attempted nor does the system try to determine this from analysis. It uses the same methods for all transfer levels (and problem domains).

Problem Domains

The problems we consider are games specified in Game Description Language (GDL), the language for the General Game Player (GGP) (Genesereth 2005). GDL is capable of describing all games specifiable as finite state machines, but define game mechanics with rules rather than transition tables. The game state consists of a set of ground sentences in the form `(predicate c1 c2 …)` with `c`'s being constant symbols. The rules are all of the form *precondition → action*. The precondition of a rule is a conjunction of predicates that test for ground sentences in the current state or the previous player move. The action is either the addition of a predicate ground to the next state, the allowance of a player move, or informing the player about game success or failure. The game state consists of a set of grounded true predicates that is updated in parallel by all rules whose preconditions are met in the state, at every discrete time step.

The four problem domains from which specific games were generated were designed by hand by three academic research teams (*not listed in for sake of anonymity*). Our group created the Escape game, described below. All games are single player.

- Wargame: The agent controls a soldier that must avoid contact with terrorists while escaping a maze. The soldier can obtain weapons with which he can then eliminate terrorists.
- Escape: The agent controls an explorer who must gather items in a maze to overcome obstacles that bar him from the exit. Items can be combined to create new items with different properties.
- Rogue: The agent controls a hero who must fight monsters and collect items in a maze. The ultimate goal is to get to the exit, after fulfilling some sort of mission (killing a monster, or collecting an item).
- Build: The agent must manipulate building materials such as concrete blocks and steel reinforcements to build specific structures in a two dimensional world.

There were few restrictions on how the games were encoded in GDL. If identical symbols existed in two domains, they had to have similar meaning. For example, if `health` is used in two domains, it cannot mean the agent's health in one domain and the name of a weapon in another. In addition, there were a limited number of fixed predicates for simple 2D space concepts that were shared across tasks.

4. Approach

Given our goal of generality, we make few assumptions about the structure of the games and rely on weak methods for problem solving and learning. Our basic approach is as follows:

- Find the solution to the source game with uninformed iterative deepening.

- Learn heuristic knowledge from that solution.
- Map the heuristic into the target game domain based on the syntactic structures of the two games.
- Solve the target game using modified iterative deepening informed by the transferred heuristic.

Solving the Source

We use uninformed iterative deepening to solve source games. The available background knowledge is sufficient to avoid exhaustive search for Wargame, Rogue, and Escape because they are all situated in two dimensional grid worlds in which only orthogonal movements are allowed and there is an identified goal square. We take advantage of this invariant by including a Manhattan distance heuristic in our uninformed iterative deepening search. The search at a level will cut off early if the Manhattan distance between the agent-controlled character and the goal exceeds the remaining depth for that state. The search will also set its initial depth to the Manhattan distance between the agent-controlled character's start location and the goal, since anything less is doomed to fail. In addition, the spatial properties of these domains allow a few other simple heuristics, such as avoiding purposeless circular movement. Without the additional knowledge described above, only problems of trivial depth are solvable because of the ensuing exponential explosion. The effect of these heuristics on transfer is minimal, as they speed up both source and target searches.

Learning a Heuristic

Since our goal is to transfer learned knowledge, we focus on learning heuristics that would be difficult to learn in the absence of a source game. Many heuristics that can be learned directly from the GDL rules could be equally extracted from the target problem itself, negating the need to transfer knowledge extracted from the source. Thus, we concentrate on learning knowledge that requires experiencing the states of the specific problem. For example, an instance of the Escape game might have a river that separates the agent-controlled character's start location from the exit, and the agent must construct a boat to cross it. The fact that water partitions the map in a goal-relevant manner is not encoded in the GDL rules, but is in the initial state of the specific game that is played. Hence the fact that the agent must construct the boat is not a property of the domain itself (the rules), but of the *instance* of the game (the rules and initial state). It is difficult to determine that the river must be crossed until the source game has been solved and crossing the river was an event that occurred in the solution.

Events in the game such as building a boat are reflected in the game state as predicate transitions. When the boat is built, the sentence `(holding boat1)` might appear. We record evidence of these events, by taking all ground predicate additions, removals, and changes that occur in the source solution path. We call these events *indicators*. We can establish a simple similarity metric between any path in the game tree and the solution by counting the number of indicators that appear on that path. We take this metric to be our heuristic.

Exact locations of objects, including the agent-controlled character, are never taken as indicators (object locations have a fixed representation in the three spatial domains). This prevents indicators that are specific to the map itself, rather than the things in it.

The remaining indicators vary widely in generality, depending on the encoding of the game. In some cases, there may be predicates that concisely describe the kind of salient events we wish to capture as indicators. In other cases, those events may be distributed across multiple predicates, or even reflected as quantity changes instead of additions and removals of qualitative predicates. A useful indicator detector algorithm must handle all of these cases. To handle distributed event representations, we take a simple approach: all transitions that occur at a given step in the game are stored as a single indicator. If any subset of those transitions appears in the target, that indicator is matched. Quantity changes are handled by generalizing exact changes in numbers to increases and decreases. If the source solution has an indicator of the enemy's health changing from 3 to 2, any decrease in the enemy's health in the target will match that indicator.

Both of these techniques—partial matching of transition sets and quantity generalization—drive the indicators to be more general than they might be under alternative approaches (matching entire sets, or only exact quantities). In our experience, we have found this to be a good extreme in the spectrum of possible strategies: it is rare that an indicator is inappropriately matched.

Transferring the Heuristic

Once we have indicators from the source problem, can we be certain they will be useful in the target? Ideally, only indicators describing invariant properties of the games should be used, so that what worked in the source will work in the target. In general, though, it is difficult to determine what properties are invariant between domains.

The property that makes indicators good for transfer also makes them intrinsically difficult to confirm as useful. As indicators cannot be determined for arbitrary source games without simulation, they also cannot be confirmed to be useful in arbitrary targets without simulation. For example, an ideal agent might solve the river-crossing Escape scenario (above) as a source problem, and learn that building a boat is a great indicator. Imagine this agent is then subjected to a target problem, where only the initial state has changed: now, there is a gap in the water. The learned indicator will now actually hurt performance, since crossing the river is a waste of time, so it would ideally be discarded. However, since the difference between source and target is in the initial state (not the rules), there is no way the agent can determine its implications without actually searching. In addition, once the agent has searched and discovered that it can reach the goal faster without building the boat, and that the indicator should be discarded, it is too late for that knowledge to be useful, because the target solution has already been found.

For this reason, we must make an assumption in designing our transfer system: we assume that similar rule structures are reflected in similar game solutions. In the example above, this clearly does not hold—the rules are identical, but the solutions do not share many commonalities. Nonetheless, we have found that this assumption holds in the scenarios that we are interested in. We will call this the *game similarity assumption*.

Given this assumption, we can arrive at a strategy for transferring indicators across domains: we create a mapping of source predicates and constants onto target predicates and constants by determining a correspondence between the source and target rule structures. In creating the correspondence, we should account for the possibility that predicates and constants in the source and target rules may be renamed, or rules may be encoded differently. If the symbol `cord` in the source game's rules is referenced as `rope` in the same context in the target, that symbol is renamed. An example of different rule encodings would be different ways of specifying the same effect of using a medical kit. One set of rules may specify that using the medical kit increases a character's health points, while another set of rules could introduce a level of indirection by specifying that using a medical kit results in healing wounds, and healing wounds results in an increase in the character's health points.

Our approach to resolving these kinds of discrepancies between source and target game encodings is to perform a best overall syntactic fitting of source to target rules and predicates. We do this in a two step process consisting of predicate mapping and constant mapping.

Predicate Mapping. Each rule in the GDL description of a game specifies a certain local causal relationship between the predicates in the precondition and action sides of the rule. The meaning and significance of a predicate is determined by its global interactions with other predicates and the goal conditions. Ideally, we would like to map source predicates onto the target predicates with the same meaning. However, since we cannot determine the global predicate relationships in the target without solving it first, we follow the game similarity assumption, settling for mapping predicates with the most similar local relations.

Our algorithm performs a greedy match of individual GDL rules, where rule similarity is defined as the number of matching predicates the rules share in their preconditions and actions. Since these measures of similarity are circular (predicate similarity depends on rule similarity, which depends on predicate similarity), our system bootstraps the process by initializing the predicate similarity measure based on the types of their arguments. Our system then iteratively select the two most similar rules, committing to a matching between the predicates in the rule preconditions and actions, boosting the similarity measures of those predicates, then repeating. It stops when a maximal matching between the source and target rules, and hence predicates, has been committed to.

Constant Mapping. Symbolic constants must also be mapped. An indicator in source problem might be the predicate `holding` bound to the constant `hammer1`. The predicate mapper might determine that the target-game equivalent of the `holding` predicate is `carrying`, but if the constant `hammer1` does not exist in the target game, this mapping is useless, as the indicator will never occur. In general, the constant mapper follows the rule that constants are similar if they can be ground to similar predicates. Suppose the predicate mapper has determined that `holding` and `carrying` are similar, as are `canDestroy` and `canCrush`. If the source constant `hammer` can be bound to `holding` and `canDestroy`, and the target constant `sledgehammer` can be bound to `carrying` and `canCrush`, it is likely that `hammer` and `sledgehammer` are similar.

The problem of constant mapping is then chiefly the problem of determining which constants in a given game can be ground to which predicates. In general, this cannot be exactly done without searching game states: if the hammer object is unreachable by the player in a given map, `holding` can never be ground to `hammer`. So an overestimation, derived by statically analyzing the rules must be used. The resulting mapping is not necessarily one-to-one: if the rules are not sufficiently constrained, a set of source constants might be mapped to a set of target constants (for example, all weapons in the source might be mapped to all weapons in the target). In general, this is a good consequence: it provides generalization power.

The runtimes for both mapping algorithms are insignificant compared to the time taken to solve the target games, except for games with very short solutions.

The predicate and constant mapping algorithms are inspired by the Structure Mapping Engine (Falkenhainer 1989). However, SME is committed to modeling human analogical reasoning and hence is constrained by several theoretical commitments that we ignore. For one, SME requires a one-to-one mapping of source and target constants, whereas our algorithm often maps one source constant to many target constants. While SME also maps based on structural similarities, the structures it is intended to work on are internal representations of the reasoner, whereas we work directly with the GDL rules and predicates. One consequence of this is that SME requires all relations be named identically across sources and targets, whereas we explicitly try to match differently named relations. Overall, SME is a general approach to mapping based on structure, whereas our algorithms are engineered for GDL-style specifications.

Solving the Target

Unlike traditional heuristics that calculate how far the goal is from a given state, indicators only discriminate between paths leading to the goal and paths that fail. To translate this information into search guidance in the target game, we modify the iterative deepening search so that it searches deeper for paths that are more likely to lead to the goal. Since we are correlating the success probability of a path with the number of indicators in its history, we increment the path's cutoff depth whenever an indicator occurs.

The benefit of this heuristic derives mainly from its ability to search deeper on promising paths and thus reach the goal before the search is completely expanded to that depth. Let n be the baseline cutoff depth that was reached in the target with transfer when a solution is found. Let k be the number of indicator increments that occurred on the solution path. Thus, the solution length is $s = n + k$. If we didn't use indicator increments, the search would have had to increment its baseline cutoff depth to s in order to find the same solution, so the number of state expansions saved by the heuristic search is approximately $\sum_{l=n}^{n+k} NS(l) - M$ where $NS(l)$ is the number of states in the search tree up to level l, and M is the number of states expanded by indicator increments that did not lead to a solution. We obtain good transfer when k is large and M is small, meaning the indicator increments look ahead very far and in the right places. Transfer will be diminished if either k is small, meaning there were not many increments, or M is large, meaning the indicators were not discriminating enough about which paths are likely to lead to solutions.

5. Evaluation Methodology

To evaluate performance, we present our system with sets of transfer scenarios. Each scenario consists of a source and target problem and is divided into two cases: the transfer case and the no-transfer case. In the no-transfer case, the system is timed on how long it takes to solve the target problem without first being presented the source problem. In the transfer case, the system is first presented with the source problem and allowed to spend unlimited time to learn from it. The system is then timed on how long it takes to solve the target problem. The benefit of transfer, called regret, is calculated as

$$regret = \frac{ntp - tp}{\max(ntp, tp)}$$

where ntp is the performance of the no-transfer case and tp is the performance of the transfer case in terms of time.

In total, forty-two scenarios were used in the evaluation. There were four test scenarios from the Build domain, while the other three domains each had seven to eight scenarios. The last sixteen scenarios were cross-domain scenarios chosen from Escape, Rogue, or Wargame. All test scenarios except eight of the differing scenarios were constructed by Naval Research Labs with input from the university groups. We constructed the last eight high differing scenarios by scrambling the symbol names in the original eight differing scenarios (as with reformulation).

6. Results

The performance of our algorithm across all test scenarios is shown in Figure 1. Figure 2 and Figure 3 show the regret averaged across all scenarios in the same level and domain, respectively. The regret value for each scenario is an average over 10 runs in both the transfer and no-transfer cases. The time-to-solve values used to calculate regret were in terms of number of states visited, which is proportional to real time, but not subject to clockspeed variations across CPUs used for generating the results.

Transfer was achieved in all scenarios, with the exception of some high differing scenarios described below, and on average across all levels and all domains. The Wargame scenarios have less overall transfer than the other domains, as that state space did not have as many salient events that exhibit themselves as indicators. The general trend across the transfer levels is that higher levels have

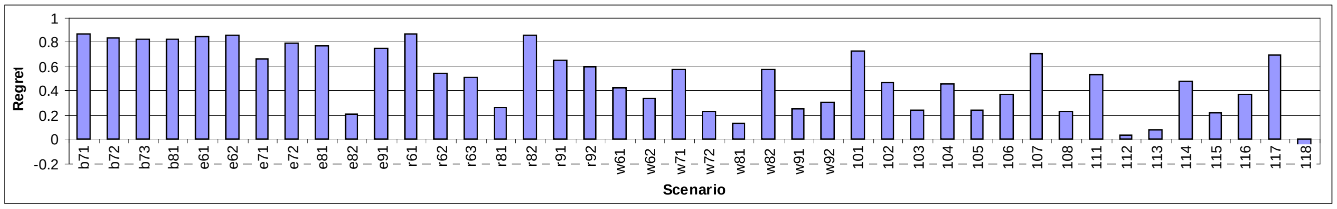

Figure 1. Performance of the transfer agent on all scenarios. The scenarios were named as domain-level-run, where the domains were b=build, e=escape, r=rogue, w=wargame. Since the level 10 and 11 scenarios are cross-domain, the domain name prefixes are not used.

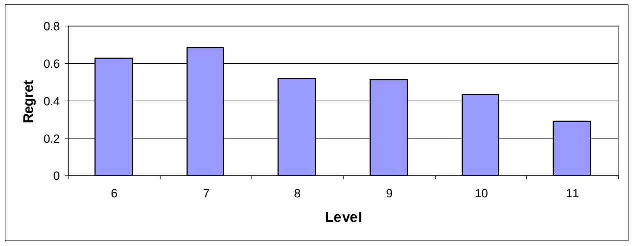

Figure 2. Average regret by level.

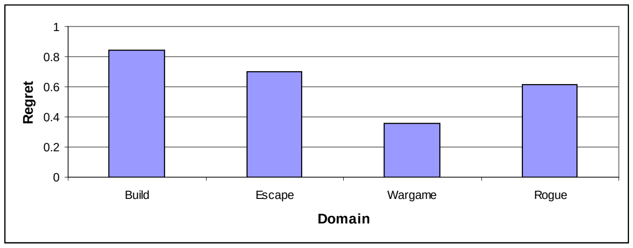

Figure 3. Average regret by domain.

lower average regret. This is expected as source and target games in higher levels share fewer surface similarities. The source-target pairs in levels 6, 7, and 8 were sufficiently similar at the syntactic level that running the scenarios without mapping indicators would have obtained roughly the same results.

Mapping was necessary for level 9 since all predicate and constant names were scrambled. However, since the source and target games in level 9 were otherwise exactly the same, the mapping algorithms were successful in matching the source indicators with the correct events in the target based on rule structure. In these lower levels, the indicators matched tended to be very specific: picking up certain objects, or overcoming certain obstacles.

In level 10, mapping was also necessary to establish correspondences between similar concepts in different domains that had different names. This was significantly more challenging since the source and target rule structures differed significantly, and not every concept in the source was present in the target and vice versa. However, there was enough common structure in the domains that indicators were successfully mapped in every scenario. These tended to be much more general than in the lower levels—events like a decrease in an enemy's health.

Finally, level 11 presented the most challenging scenarios since the source and target problems shared no common names or structure except at an abstracted level. For scenarios 2, 3 and 8 in level 11, there was no successful mapping of the indicators from the source to the target, so that there were no indicator increments in the transfer conditions. In these cases, the deviations from 0 regret were due to random variation rather than transfer. However, considering the purely syntactic approach we took in transfer and the abstract invariances in level 11, it is remarkable that scenarios 4, 5, 6, and 7 had as much transfer as they did.

The overall competence of our transfer agent on the variety of different source-target transformations suggests that the properties we based our algorithms on are quite general for single player, single controlled character, grid-world type games, with limited background knowledge.

An Unexpected Source-Target Invariance

A critical dimension that influences the search time in these domains is the difference between the initial Manhattan distance to the goal and the length of the solution path: this is the number of times iterative deepening must iterate. Since our transfer method interacts with this aspect (by incrementing the cutoff depth when indicators are encountered), it is important to consider the influence of it alone.

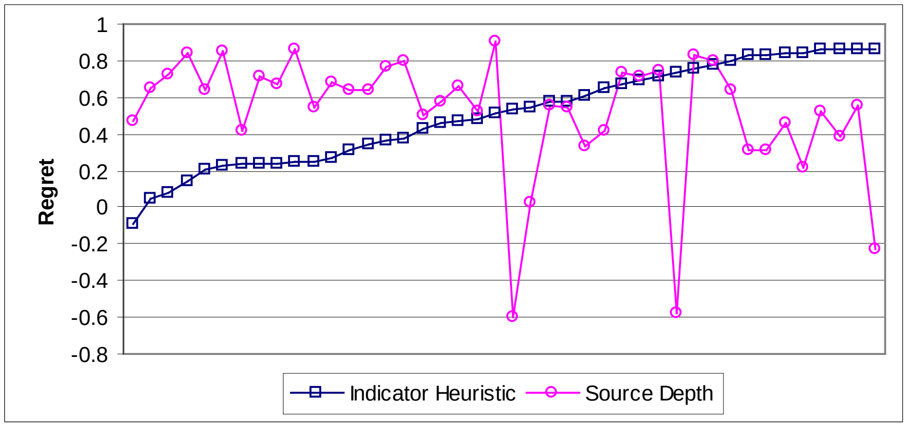

Figure 4. Performance of using the indicator heuristic versus initializing transfer-case iterative deepening with source solution length. The data points are sorted by increasing indicator performance.

To do this, we ran the tests without the heuristic, and instead initialized the uninformed iterative deepening in the transfer cases to the source solution depth. The non-transfer cases remained the same. A comparison of the regret obtained using the two methods is shown in Figure 4. The plot shows the regret obtained in each test scenario from running the transfer-case search using the indicator heuristic versus initializing the cutoff depth to the length of the source solution. The average regret obtained over all scenarios using the indicator heuristic is 0.506, while for initializing cutoff depth is 0.519. Although it is initially surprising that transferring the source solution length gives an overall performance roughly equal to that of using the indicator heuristic, it is also apparent that the two methods' performances differ significantly for each scenario.

The reason for the high performance of transferring the source solution depth is an unintentional bias in the design of the test scenarios where the solution depth of each scenario was inadvertently constrained by two factors. The first factor was the need to solve the games in a reasonable amount of time using uninformed iterative deepening, which set an upper bound on solution length. The second factor arose from the desire to make the games as difficult as possible, which set the lower bound. As a result, the source and target solutions of each scenario did not differ by more than 4 in most cases. Furthermore, in all scenarios except one, the source solution was longer than the target solution. A deeper-than-necessary cutoff depth for iterative deepening did not result in a large number of extraneous states being expanded, since most paths circled back to previously visited states and were cut off. In terms of the previous analysis, the high k value from the source solution depth has made the summation term overpower M in these scenarios, because the growth of M as a function of the baseline cutoff happened to be slow due to the limited search space.

Solution depth is clearly not a property that should be leveraged in transfer situations and the indicator heuristic is not exploiting this coincidental feature: it tries to find a trade-off between k and M, and the lack of correlation between the results of the two approaches in Figure 4 shows that the indicator heuristic does not rely on indirectly transferring solution depth.

7. Related Work

Taylor et al. (2008) have also used source solutions to generate heuristics for transfer within GGP. However, they work within a single game domain and their heuristic provides a state-to-action mapping rather than a solution-likelihood estimate. Moreover there technique requires multiple sources per target because they guard against expected source-target differences by averaging their heuristic over a large number of distinct source trials.

Several results have been published using SME and analogical reasoning to achieve transfer learning. Klenk & Forbus (2007) created an agent to solve AP physics problems by mapping solved source problems to target problems and transferring the source answer directly. This is an alternative to solving problems by search, but does not apply to GGP games. The system of Hinrichs & Forbus (2007) plays the turn-based strategy game FreeCIV by mapping HTNs with SME. This differs from our approach in that the FreeCIV agent does not have access to its domain theory and must transfer an internal representation. Furthermore, transferred knowledge is not in the form of heuristics, but rather relationships established in the source.

Work has also been done in automatic heuristic extraction from GGP problems, chiefly in the context of GGP competitions. These systems (e.g., Schiffel and Theilscher, 2007) typically work by extracting heuristics directly from the GDL specification, without simulation, by relying on a fairly explicit goal description in the game rules. As discussed above, however, in our domains most useful knowledge is implicit in the instance of the game, not the game rules. For example, the goal of the game might only encode that the agent wins if it kills a monster, where the steps needed to kill that monster are dependent on the initial state of the world, where the walls are, which weapons are accessible, etc. For this reason, we used different learning techniques.

8. Conclusion

We have presented our approach to transfer learning in GGP domains: augmenting iterative deepening search with an indicator heuristic, where predicate changes on the solution path of the source problem are mapped to the target problem, and detected during target search. The domains are mapped based on rule similarity, an approach powerful enough that indicators such as decreasing an enemy's health can be successfully transferred across game domains and encoding techniques, increasing performance in the target domain. This system can be taken as an implementation of a theory that transfer can be accomplished by relating situations in the target problem to situations previously experienced in the source, by comparing the structure of the environments that created those situations. These results provide experimental evidence for this conception of transfer learning.
References

9. Bibliography

DARPA 2005. *Transfer Learning* (BAA 05-29).

Falkenhainer, B., Forbus, K., and Gentner, D. The Structure-Mapping Engine: Algorithm and Examples. *Artificial Intelligence* 41. 1989.

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

Hinrichs, T. and Forbus, K. Analogical Learning in a Turn-Based Strategy Game. In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence. 2007.

Klenk, M**.** and Forbus, K. Measuring the level of transfer learning by an AP Physics problem-solver. *Proceedings of AAAI-07: Twenty-Second Conference on Artificial Intelligence*. 2007.

Stephan Schiffel and Michael Thielscher: Fluxplayer: A Successful General Game Player. *Twenty-Second AAAI Conference on Artificial Intelligence (AAAI07).* 2007.

Taylor, M. E., Kuhlmann, G., and Stone, P. Transfer Learning and Intelligence: an Argument and Approach. To appear in In Proceedings of the First Conference on Artificial General Intelligence, March 2008.