Cross-Domain Transfer of Planning Strategies: Alternative Approaches

Bruce Krulwich, Gregg Collins, and Lawrence Birnbaum
Northwestern University
The Institute for the Learning Sciences and
Department of Electrical Engineering and Computer Science
Evanston, Illinois

ABSTRACT

We discuss the problem of transferring learned knowledge across domains, and characterize two possible approaches. *Transfer through reoperationalization* involves learning concepts in a domain-specific form and transferring them to other domains by recharacterizing them in each domain as necessary. *Abstraction-based transfer* involves learning concepts at a high level of abstraction to facilitate transferring them to other domains without recharacterization. We discuss these approaches and present an example of the abstraction-based transfer of a method of *projection*, or selective lookahead, from the game of chess to the game of checkers, as implemented in our test-bed system for failure-driven learning in planning domains. We then discuss a continuum of abstraction to characterize learned concepts, and propose a corresponding continuum characterizing the time at which the computation necessary for cross-domain transfer is accomplished.

1 Introduction

Human beings have the ability to learn a concept in one domain and apply it in a different domain. Achieving such *cross-domain transfer* is a current goal of machine learning systems as well. Transferring a concept learned in one domain into a different domain involves both determining that the concept is applicable to the new domain, and casting it in a form that is appropriate to that domain.

Previous studies of the human ability to transfer knowledge between domains have investigated the way in which this ability depends on the learning method employed and on the learner's prior knowledge [Katona, 1940; Hilgard, Irvine, and Whipple, 1953; Mayer and Greeno, 1972]. In particular, it has been observed that the ability to transfer learned concepts between domains depends on the level of abstraction at which the concepts are represented and indexed [Hilgard, Ergren, and Irvine, 1954; Mayer and Greeno, 1972; Singley and Anderson, 1989]. Brown has proposed a hypothetical continuum of knowledge abstraction in the context of transfer that ranges from complete *theories*, explaining how a concept relates to the other knowledge of the system, to *arbitrary solutions* that do not include any explanation of either correctness or appropriateness, and includes a number of intermediate levels [Brown, 1989].

A computer system might represent and index concepts that it learns at any of these levels of knowledge abstraction, and its ability to transfer this learned knowledge to another domain would depend upon the level employed. If the system formulates a concept in terms of a general theory, the concept will, as a result, be applicable to any domain in which the

theory is applicable. If, on the other hand, the system develops only a partial explanation of a concept, without tying it to a more general theory, it can nevertheless transfer this concept to domains in which the explanation is applicable. Concepts for which no explanation is generated can of course be applied to other domains only on a hit-or-miss basis.

The approach that must be taken to transferring a concept across domains depends upon the level of abstraction of the vocabulary in which it is expressed. In particular, we can distinguish two basic approaches, each appropriate to a different end of the abstraction continuum. *Transfer through reoperationalization* applies when transferring a concept that has not been abstracted out of its particular domain, and which therefore must be recharacterized in terms appropriate to the new domain. *Abstraction-based transfer*, on the other hand, is appropriate when transferring a concept expressed in terms of a general theory that is already applicable to the new domain. In this paper we will discuss these two approaches, and the trade-off between them, and present a computer implementation of *abstraction-based transfer* in competitive planning.

2 Transfer through reoperationalization

In *transfer through reoperationalization*, a concept is initially learned in a vocabulary specific to a particular domain, and must therefore be recharacterized, or *reoperationalized* (see, e.g., [Mostow, 1983]), when it is needed in another domain. The distinction between this approach to transfer and others is that no processing is done to facilitate transfer of a concept before the concept is needed in the new domain. This approach has been taken by a number of researchers, include many in the areas of case-based reasoning (see, e.g., [Kolodner, 1988; Riesbeck and Schank, 1989]) and analogy (see, e.g., [Gentner, 1983; Carbonell, 1986]). In this approach, transfer is accomplished in two steps: first, determining that a particular concept learned in one domain is applicable to another domain, and second, expressing the concept in the vocabulary of the new domain.

This approach raises several difficult issues. The first step, determining that a concept is applicable, leads to the *indexing problem*, the problem of organizing domain-specific concepts in memory in a way that facilitates retrieval in appropriate situations in other domains (see, e.g., [Schank, 1982]). The second step, expressing the concept in the new domain, requires a theory of how to map concepts between the two domains (see, e.g., [Gentner, 1983]). While a number of fruitful approaches to these problems have been developed, the process of applying a concept in a new domain remains computationally expensive. On the other hand, the benefit of transfer by reoperationalization is that no effort is expended when the concept is learned to prepare it for use in domains other than the one in which the system is currently operating.

3 Abstraction-based transfer

Abstraction-based transfer involves learning new concepts in a form that facilitates their transfer directly to other relevant domains. This processing involves immediately generalizing the concept to the highest level of abstraction that is supported by its explanation (see, e.g., [DeJong and Mooney, 1986; Mitchell, Keller, and Kedar-Cabelli, 1986]). When the concept is applied in the new domain, it can either be used directly or specialized to the vocabulary of the new domain.

This approach to transfer has the advantage of minimizing the cost of indexing and mapping, since applicable concepts will already be expressed in terms relevant to the current domain. On the other hand, it raises the question of how, or even *whether*, the concept can be learned in terms that are abstract enough to apply in all domains for which the concept might be appropriate.

4 An implementation of abstraction-based transfer

Cross-domain transfer of knowledge is a central concern of our current research in failure-driven learning in planning domains (see, e.g., [Birnbaum and Collins, 1988]). To explore these issues more concretely, we have implemented a game-playing model for two player turn-taking games, along with specific rules for such games as chess and checkers, within our test-bed system for exploring failure-driven learning in planning domains (see, e.g., [Collins, Birnbaum, and Krulwich, 1989]). This system provides a unified framework for inference, justification maintenance, expectation monitoring, and explanation of failures. All of the beliefs, rules, and expectations in the system are tagged with justification structures indicating the basis for the system's belief in their correctness. The system learns from its failures [Sussman, 1975; Schank, 1982; Hayes-Roth, 1983; Kolodner, 1987; Hammond, 1989] by monitoring the truth of any predictions it makes about the world in the course of its decision-making. When any of these expectations fail, the system constructs an explanation of the failure using the justification for that expectation. Should this explanation reveal a bug in the system's decision-making mechanism, the system will attempt to modify its rules to correct the problem, thus avoiding similar failures in the future.

Decision-making in our system is accomplished by a fairly general, albeit rudimentary, mechanism. First, the decision component computes the opportunities available to the computer, using game-specific notions of threats and moves. The results of each possible move are then predicted by the projection component, which uses a general method for all games. These results are then ranked by the evaluation component, which is again specific to the game being played. The computer then chooses the highest-ranked such move.

We will focus here on the second of these three components, the *projection* component. Because the general method of projection used by our system is game-independent, it is capable of supporting abstraction-based transfer between games. In general, we would like our system to project the results of a move as far into the future as possible, because this will improve its value as an estimator of the quality of that move. However, the further into the future a projection is carried, the more expensive it will be to compute, and a system will have no basis for making an *a priori* decision about this tradeoff upon entering a new domain. Our approach to this problem is to have the system start out with a very simple projection mechanism that it will augment as necessary [Krulwich, Collins, and Birnbaum, 1989]. In particular, our system will initially project the results of a move by assuming that on the following turn the opponent will make the best move that is *currently* available to it. The rule that implements this projection method, which is shown in figure 1, says roughly that *the situation resulting from making a move is the situation after making the move and after the opponent makes the move which is the best move at the current time.* This unsophisticated procedure considerably simplifies the projection problem, because it obviates the need to repeatedly recompute the opponent's response for each move contemplated by the planner. However, its validity, and hence the validity of the predictions that it generates concerning the opponent's moves, depends upon an

```
(def-brule proj-factor-2  <...variable declarations...>
      (project-factor proj-factor-1.5 world move player result 1)
   <=
      (and (= world (world-at-time time))
           (decide (player-opponent player)
                   (possible-moves-at-time (player-opponent player) time)
                   world simple-dec-factors opp-move)
           (= result (world-after-move opp-move (world-after-move move world))) ))
```

Figure 1: Initial Projection Method

assumption that nothing will occur to enable the opponent to make a higher-priority move than those currently available to it. This is an instance of what is sometimes referred to as a *persistence assumption,* namely, an assumption that things will stay as they are as much as possible. This assumption is itself justified by a conjunction that says, roughly, *Nothing will happen to give him a better move because, (1) he can't do anything to give himself one, (2) no outside forces will give him one, and (3) I won't do anything to give him one.* These assumptions make up the justification for the projection method in figure 1. These assumptions are clearly not always true; the problem for our system is to determine the situations in which they are not true, and hence in which a more sophisticated projection method is necessary.

Consider how our system would behave in the situation shown on the partial chess board in figure 2a. Taking the opponent's knight with the rook looks like the best move, because the computer expects that the opponent will take *its* knight in the following turn, and it believes that trades are to its benefit. However, the opponent will not make this move, as we can see in figure 2c, because the computer's move gives him the opportunity to take the computer's rook, which is a more valuable piece than the knight. When the opponent makes his move to take the computer's rook, the system's expectation about the opponent's move—that he will take the knight—will fail.

In response to this expectation failure, the system will analyze the justification for the faulty expectation in order to explain the error. Traversing this justification will lead it to test the assumption that *the computer won't do anything to give the opponent a better move,* and the system will find that this belief is to blame for the expectation failure. In particular, the computer's own move enabled the better move that the opponent made. The system responds to this by patching its projection rule to take into account the possibility that

```
++ -- OP -- ++        ++ -- OP -- ++        ++ -- ++ -- ++
-- ON -- CN --        -- CR -- CN --        -- OP -- CN --
++ -- ++ -- ++        ++ -- ++ -- ++        ++ -- ++ -- ++
-- ++ -- ++ --        -- ++ -- ++ --        -- ++ -- ++ --
++ CR ++ -- ++        ++ -- ++ -- ++        ++ -- ++ -- ++

 (a) RxN   ...         (b) ...   BxR         (c) ...   ...
```

Figure 2: Faulty Projection in Chess: Computer to Move at Start

```
(def-brule proj-factor-3  <...variable declarations...>
   (project-factor proj-factor-1.5-mod world move player result 1)
 <=
   (and (= world (world-at-time time))
        (decide (player-opponent player)
                (possible-moves-at-time (player-opponent player) time)
                world simple-dec-factors opp-move)
        (= result (world-after-move opp-move (world-after-move move world)))
        (no (and (move-enables-move move better-move)
                 (move-possible better-move (current-time))
                 (move-legal better-move)
                 (evaluate result eval-factors (player-opponent player) orig-value)
                 (evaluate (world-after-move better-move (world-after-move move world))
                           eval-factors (player-opponent player) better-value)
                 (> better-value orig-value)  )) ) )
```

Figure 3: The Modified Projection Method

the computer's move enables a better move for the opponent, and to ignore the persistance assumption in such cases. This improved projection method, shown in figure 3, is the old method with the added condition that the computer's move not enable a better move for the opponent. This modification to the projection method will prevent the computer from making the same mistake in the future. When placed in the same situation as before, it will instead move its threatened knight to safety (as shown in figure 4).

Transferring to another domain

While many aspects of the process of projection are specific to the game of chess, the system's description of the improved projection process is as general as the original projection method was, and is thus expressed in vocabulary that is applicable to all turn-taking games. This should enable the system to use *abstraction-based transfer* to apply the more sophisticated method to similar games. To test this, we disabled the learning component of the system, and put the system in an analogous situation in the game of checkers. Using the original projection method described above, the system played as shown in figure 5. The computer predicts that the opponent's move will be to jump the computer's piece in the upper right-hand corner. As a result, the computer decides that its own best move would

```
++ -- OP -- ++        ++ -- OP -- ++
-- ON -- CN --        -- ON -- ++ --
++ -- ++ -- ++        ++ -- ++ -- ++
-- ++ -- ++ --        -- ++ -- ++ CN
++ CR ++ -- ++        ++ CR ++ -- ++

      (a)                 (b)
```

Figure 4: Repaired Projection in Chess: Computer to Move at Start

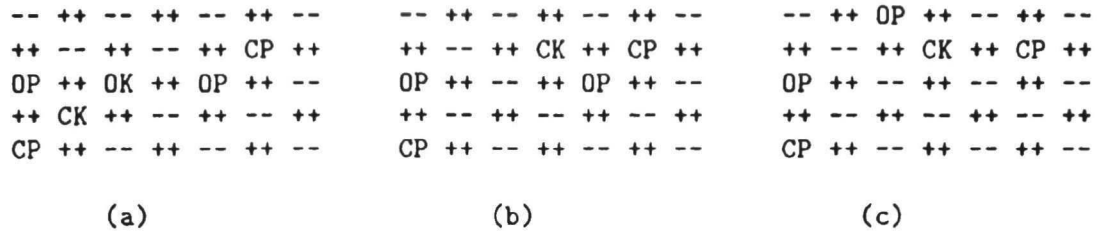

Figure 5: Faulty Projection in Checkers: Computer to Move at Start

be to take the opponent's king with its king. As we can see in figure 5b, the computer's prediction fails for the same reason that the chess prediction failed in the previous example namely that the computer's move enabled a better move for its opponent.

Next, we re-enabled the learning component of the system, and ran it through the chess situation described previously, allowing it to learn the improved projection rule described in figure 3. When we subsequently reran the checkers situation, the computer applied what it learned, and improved its behavior, as shown in figure 6.

5 Conclusion

Both approaches to transfer that we have discussed have their advantages and disadvantages. *Transfer by reoperationalization* requires realizing that the concept to be transferred is appropriate to the new domain, which involves computationally expensive processing at the time of problem solving in order to retrieve and apply the concept appropriately. On the other hand, *abstraction-based transfer* requires expressing the concept at a high enough level of abstraction to allow its utilization in all appropriate domains, which again involves computationally expensive processing, in this case at the time the concept is learned. It seems clear that the optimal approach will lie somewhere in between these two extremes.

Our research has, in part, been an exploration of this trade-off. One intermediate approach we have been pursuing is *explanatory transfer*, in which a concept is learned in response to a failure, and is generalized as much as is necessary to explain that failure. When similar failures occur in other domains, the concept may be retrieved and applied in the

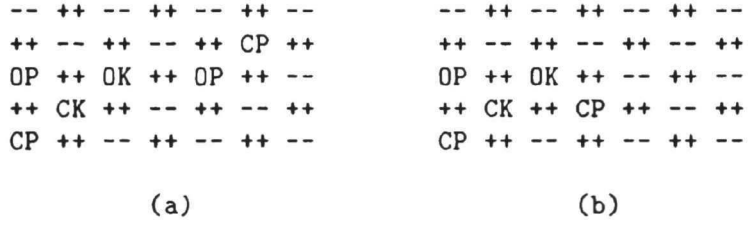

Figure 6: Repaired Projection in Checkers: Computer to Move at Start

course of constructing an explanation for the current failure [Collins and Birnbaum, 1988; Kass, 1989].

This approach combines several of the advantages of each of the previous two approaches. As in *abstraction-based transfer*, the concepts are generalized when they are learned, which reduces the computational effort necessary in retrieving and applying them in new domains. However, as in *transfer through reoperationalization*, the applicability of concepts to other domains need not be determined at the time the concept is learned. Furthermore, determining the applicability of the concepts to be transferred is more focussed—by the need to explain a particular failure in the new domain—than it is in the other approaches. Future research is necessary to develop other approaches to transfer that are between the two extremes, and to determine the areas in which the different approaches are applicable.

Acknowledgments: We would like to thank Matt Brand, Ann Holum Faillettaz, Mike Freed, Eric Jones, and Dick Osgood for many useful discussions. This work was supported in part by the Office of Naval Research under contract N00014-89-J-3217, and by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058.. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization.

6 References

Birnbaum, L., and Collins, G. 1988. The transfer of experience across planning domains through the acquisition of abstract strategies. [Kolodner, 1988], pp. 61-79.

Brown, A. L. 1989. Analogical learning and transfer: What develops? [Vosniadou and Ortony, 1989], chapter 14, pp. 369-412.

Carbonell, J. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. *Machine Learning: An Artificial Intelligence Approach, Vol. 2*, Morgan Kaufmann, Los Altos, CA, pp. 371-392.

Collins, G., and Birnbaum, L. 1988. Learning strategic concepts in competitive planning: An explanation-based approach to the transfer of knowledge across domains. Research report no. UIUCDCS-R-88-1443, University of Illinois, Dept. of Computer Science, Urbana, IL, 1988.

Collins, G., Birnbaum, L., and Krulwich, B. 1989. An adaptive model of decision-making in planning. *Proceedings of the Eleventh IJCAI*, Detroit, MI, pp. 511-516.

DeJong, G., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning*, vol. 1, pp. 145-176

Gentner, D. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, vol. 7, pp. 155-170.

Hammond, K. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, San Diego, CA.

Hayes-Roth, F. 1983. Using proofs and refutations to learn from experience. In R. Michalski, J. Carbonell, and T. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, Tioga, Palo Alto, CA, pp. 221-240.

Hilgard, E., Ergren, R., and Irvine, R. 1954. Errors in transfer following learning by understanding: Further studies with Katona's card trick experiments. *Journal of Experimental Psychology*, vol. 47, pp. 457-464.

Hilgard, E., Irvine, R., and Whipple, J. 1953. Rote memorization, understanding, and transfer: An extension of Katona's card trick experiment. *Journal of Experimental Psychology*, vol. 46, pp. 288-292.

Kass, A. 1989. Adaptation-based explanation: Extending script/frame theory to handle novel input. *Proceedings of the Eleventh IJCAI*, Detroit, MI, pp. 141-147.

Katona, G. 1940. *Organizing and memorizing.* Columbia University Press, New York.

Kolodner, J. 1987. Capitalizing on failure through case-based inference. *Proceedings of the Nonth Cognitive Science Conference*, Seattle, WA, pp. 715-726.

Kolodner, J. 1988. *Proceedings of the 1988 Workshop on Case-Based Reasoning*, Morgan Kaufmann, San Mateo, CA.

Krulwich, B., Collins, G., and Birnbaum, L. 1989. Improving decision-making on the basis of experience. *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY, pp. 55-57.

Mayer, R., and Greeno, J. 1972. Structural differences between learning outcomes produced by different instructional methods. *Journal of Educational Psychology*, vol. 63, pp. 165-173.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. 1986. Explanation-based generalization: A unifying view. *Machine Learning*, vol. 1, pp. 47-80

Mostow, D. 1983. Machine transformation of advice into a heuristic search procedure. In R. Michalski, J. Carbonell, and T. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, Tioga, Palo Alto, CA, pp. 367-403.

Riesbeck, C., and Schank, R. 1989. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Schank, R. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, Cambridge, England.

Simmons, R. 1988. A theory of debugging plans and interpretations. *Proceedings of the 1988 AAAI Conference*, St. Paul, MN, pp. 94-99.

Singley, K., and Anderson, J. 1989. *The Transfer of Cognitive Skill.* Harvard University Press, Cambridge, MA.

Sussman, G. 1975. *A Computer Model of Skill Acquisition.* American Elsevier, New York.

Vosniadou, S., and Ortony, A. 1989. *Similarity and Analogical Reasoning.* Cambridge University Press, Cambridge, England.