Memory-Based Problem Solving and Schema Induction in Go

Alex Heneveld, Alan Bundy, Michael Ramscar
{heneveld,bundy,michael}@cogsci.ed.ac.uk
Institute for Representation and Reasoning
Division of Informatics
University of Edinburgh
Edinburgh, Scotland EH8 9LW

Julian Richardson
julianr@cee.hw.ac.uk
Deptartment of Computing and
Electrical Engineering
Heriot-Watt University
Edinburgh, Scotland EH14

Abstract

This project presents a memory-based, analogical model of complex problem solving with a technique of schema formation. Cases in the game of Go are described in a predicate logic representation of spatial stone arrangements near recent moves on the board, and then structure-mapping (Gentner 1983) is used to suggest candidate moves in novel situations based on exemplar cases from expert games. The analogy process is also used to generalise across previous cases to form new schema cases. Problem solving using these prototype schemas is compared with the exemplar-only model. The exemplar run effectively found solutions to about 50% of the problems; schemas performed very similarly, taking half as long and identifying a few useful Go principles. This suggests to us that pure-exemplar models of memory-based processing can be made faster and more compact by introducing schemas. Analysing the model's weaknesses highlights the need for richer board representation and for a reminding stage to select relevant cases. Future work will also focus on using a move evaluation stage to filter spurious generalisations, and using both the evaluation and the generated schemas to enrich the representation.

Introduction

This paper explores a model of memory-based problem solving to see whether analogical reasoning can be effective at suggesting solutions to complex problems and to see whether schema induction via analogy can serve as a basis for learning useful generalisations. We describe two machine learning experiments designed to test how well exemplars and abstracted schemas perform when used to suggest candidate moves in the game of Go, and review some computational and cognitive implications of this work.

In the memory-based psychological paradigm, experience cases stored in memory are the starting point for solving a target problem, roughly in a three step process.

(1) *Reminding:* surface features prime cases

(2) *Matching:* cases analysed to suggest solutions

(3) *Evaluation:* solutions considered in context

Available features of the problem cue the retrieval of experiences containing similar features, in the first, reminding step. These potentially relevant experiences are then analysed — such as by looking for analogous propositional structure — and if it is a good match, a portion of the remembered experience may be transferred as a potential solution. (In iterative models, the matching step may uncover new surface features which cause new cases to be retrieved.) Potential solutions are evaluated relative to the context and the goals of the target problem, in the third step, and used to form the eventual solution. This approach is closely related to case-based reasoning, and is also apparent in exemplar models of categorisation (Nosofsky, 1984) and some recent work on natural language processing (Daelemans, van den Bosch, & Weijters, 1997).

A schema is a description of general experiences, often formed from a family of episodes with elements in common. They can supplement or organise a simple exemplar model by offering a concise source of the essential factors in many experiences without the incidental details present in episodic memories. Frames, scripts, and model-based reasoning are examples of their use in AI. The theory of pragmatic reasoning schemas (Cheng & Holyoak, 1984) is a clear account of how schemas can combine the best attributes of competing memory-based and rule-based views to model logical reasoning. In the categorisation literature, both prototype and theory (see Komatsu, 1992, for review) models can be considered as relying solely on a schema-definition of categories. Unfortunately in all these theories how to form these schemas is a difficult problem: the second set of experiments below tries a rudimentary method of inducing schemas from analogical matches performed during the memory-based process.

The strategy game Go was selected as a domain for our experiments for three main reasons: it is a plentiful source of difficult problems, many of which computers cannot currently solve; a vast amount of data is available on the Internet Go Server (IGS, 2000); and it does not involve much outside knowledge. The game is played by two players, one with black stones and the other with white, who take turns placing their stones in any unoccupied space on a 19x19 board, with the goal of amassing the greatest amount of territory. Players can capture their opponent's stones, individually or in orthogonally connected groups, by surrounding them in such a way that the captured group is not adjacent to any empty squares of the board. Captured groups are removed from the board, and the newly-unoccupied region typically becomes the capturing player's territory.

Many books can give more information on the rules, the strategy, and the history of the game (Bozulich, 1992, is good for this). Archives for research in Go-playing computer programs are on-line (Reiss, 2000), as are archives for psychological studies in Go (Burmeister, 2000).

One of these psychological studies (Saito & Yoshikawa, 1996) indicates that expert Go players quickly focus on their eventual move, (order of 200 ms in TsumeGo); and that they usually consider the outcome of only one or two possible moves. For these candidate moves, there is a lengthy lookahead evaluation which may go as far as 11 moves deep. Traditional Go-playing programs, even those which have been proposed as cognitive models, perform a broader search on a much greater number of candidate moves, necessarily to a lesser depth and with the result that no Go programs play any better than an amateur. It is fascinating that skilled players are able to focus rapidly — intuitively — on the best moves. In this research, we explore a memory-based model of how this might be done and also whether the abstraction of schemas can benefit performance.

Experimental Design and Setup

In our view, the expert Go player relies on a large number of case experiences in memory, efficiently represented, and when a new problem is presented, retrieves a small number of possibly-relevant exemplars for fuller evaluation. Schemas may also be involved in representing common, recurring segments of exemplars as easily-accessible, general cases. The major issues involved in developing a computational cognitive model of this view are how the experiences are internally represented, how relevant experiences are selected from memory, and how schemas are formed.

Representation

As in many machine models of complex problem solving, the representational format used here was a propositional description language recording a small number of perceptually basic, salient features. Specifically, for each problem, this encodes the colours of neighbouring stones around the two previous moves (`X` and `O`); the relative position between these two moves (in the format `(rel X (rel-1 w 1))`, meaning one position to the west of the last move); and, for cases including the expert's solution, the relative position between the actual response (`Q`) and the two previous moves.

```
E1   (my-last-move O)
E2   (is-colour white O)
E3   (is-colour black (rel O (rel-1 w 1)))
E4   (is-colour black
        (rel O (rel-1 n 1)))
E5   (is-colour black
        (rel O (rel-1 s 1)))
E6   (is-colour white
        (rel O (rel-1 s 2)))
E7   (make-move Q)
E8   (is-colour white Q)
E9   (equal-position Q (rel O (rel-1 e 1)))
```

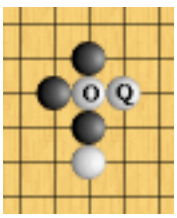

Figure 1: A fragment describing an atari opening east

This subset of information was selected because it corresponds generally to the initial observations that a player makes, guided by attentional cues to recent moves and nearby stones, and because as limited and easily-compiled as it is, it already contains enough information to begin drawing conclusions about where to play in certain circumstances. A more complete model of the expert's initial representation might note a great many more features, symbolising more complicated concepts, but we hoped that this simplification would yet give promising results. The routines we developed to build the description from on-line game records and the routines to evaluate the descriptions as LISP code for visual output are designed to work with a family of vocabularies.

Analogical Processing

The target problems were compared with cases in memory using the Structure-Mapping Theory of Analogy (Gentner, 1983). We chose this theory because it has been widely examined in the literature on the psychology and computer modelling of analogy, and because the Structure-Mapping Engine program (Falkenhainer, Forbus, & Gentner, 1989) is ideally suited to our representation. Routines in SME can easily read our descriptions, perform the analogy in keeping with a demonstrated psychological theory, score potential matches, and return inferences which correspond to candidate move suggestions.

In the implementation of our model, individual analogies are taken between a target problem and each of 1500 cases in memory. In practice, this is a slow, sequential process that takes between 10 and 30 minutes per problem on a Sun Ultra-10 workstation. In theory, however, each analogy is independent, and this might correspond to a very quick, parallel neural computation done by the brain. It could also be made significantly quicker by incorporating a more diverse description language with a "reminding" approach. A smaller number of cases which seem relevant can be primed by surface features in the target problem, with only those cases containing similar predicates used for the analogical matching (Forbus, Gentner, & Law, 1995). Whether these improvements could get the time per problem down to the order of 200 ms is unclear; up to 30 minutes per problem, though, is not prohibitive at this stage of the experimentation.

Schema Induction

Once an analogical match has been made between a target problem and a source (base) case, the result can also be used to abstract the common substructure. Instead of looking only at the inferences, the process copies all the matching expressions into a new, schema case. This encodes the essential description elements from pairs of exemplar cases into more compact, abstracted descriptions which can then be used as base cases for solving future problems. Additionally, by repeating the induction on schema cases, the process can

also capture patterns recurring across many experiences in memory. This cognitive model of generalisation learning has been argued for on the basis of order effects by Kuehne, Forbus, & Gentner (1999). A similar algorithm, the Least General Generalisation (Plotkin 1971), has been widely used in AI to induce descriptions in logic, and a related approach has been applied to pattern learning in Go (Stoutamire, 1991). In our model, applying this type of schema induction after analogy has been used to generate candidate solutions has the advantage that the generalisation is returned with very little additional computational effort.

To perform the induction, we developed a LISP module within the SME package that transforms the result of a completed analogy into an abstracted schema. The schema uses the same description language as the parent cases, copying identical predicates exactly and inventing new tokens where the corresponding labels in the parents differ. The resulting schema can be used directly as a base case for analogical reasoning or output to a file. The second set of experiments reviewed below investigates the utility of this induction algorithm in solving complex problems and learning patterns.

Problem Solving from Examples

The first experiment tried to solve 100 random Go problems by analogy to a library of exemplar cases. We take *solve* to mean that the move chosen by the expert player in the actual game is ranked in the top 50 in the program's list of suggestions (sometimes also top 3 or top 10). In a full-fledged Go-playing model this would be followed by an elaborate evaluation stage which would consider the effects of the candidate moves, typically using some form of lookahead search. Go players perform this lookahead on 1.5 moves on average (Saito & Yoshikawa, 1996), whereas most computer Go programs will evaluate between 20 and 70 moves.

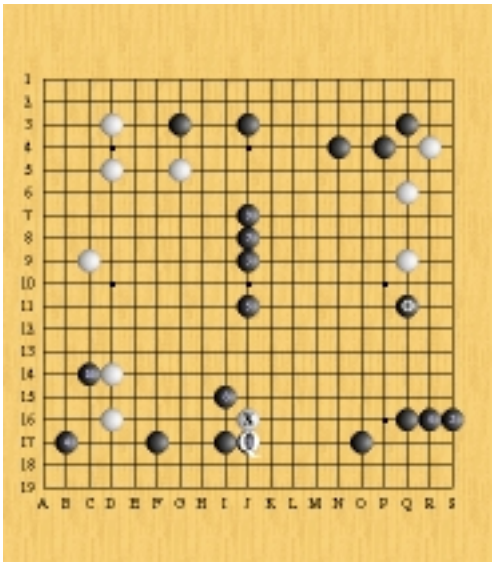

Figure 2: Sample results to a target problem. Numbers indicate ranked suggestions; `X` and `O` are the last two moves; and `Q` is the expert's move.

Materials

We simulated the human's experience as consisting of 1500 exemplar cases drawn from ten tournament games (freely available on the Internet Go Server, IGS, 2000). These source cases, *ten-games*, are each a LISP-style description in the vocabulary outlined above, recording the neighbourhood of the last two moves with an indication of the move the expert made at that point in the game. One-hundred target problems, *query-random-100*, were compiled from random turns (before the end-game) in other IGS tournament games in the same manner but without any indication of the expert's response.

Results

Figure 2 illustrates one of the better responses to a target problem. The program's top suggestion (`1`) is the expert's "right" answer (`Q`), and is quite close to the opponent's previous move (`X`). Many of the other candidates were in other sections of the board, reflecting problems locally similar to this one where the expert did play in other areas. (In this situation, playing elsewhere might be better, but as this model attends to stones near `X` and `O`, it cannot draw very good conclusions about distal play.)

The program solved 51 of the 100 problems after running for 13 hours (taking the top 50 suggestions). If all suggestions are considered, solutions to 93 problems were found; however, the program made an average of 6791 suggestions per problem. (There are only 361 positions on the board). There were a large number of repetitions — on average, the right answer is suggested 104 times. Looking at different suggestion depths gives a better picture of the program's performance: among the top 3 suggestions, the right answer was found for 7 problems; among the top 10, for 20; and among the top 50, for 51. Figure 3 shows the performance as up to 200 suggestions are considered.

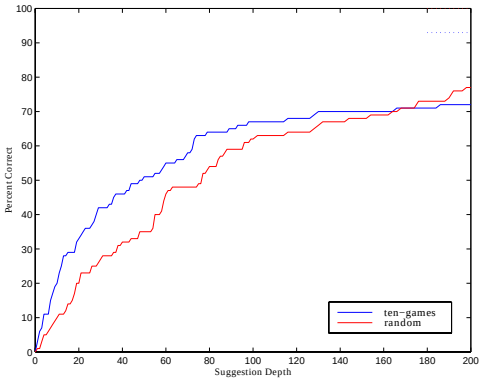

Figure 3: Experiment 1 results. This graph shows the percentage of problems solved by considering the x highest-scoring suggestions. The heavy line is our program, and the light line a chance player based on the 1970 Zobrist program. The dotted line is the asymptotic percent solved when all inferences were considered.

To put these numbers in perspective, we developed an informed random heuristic on the basis of the 1970 Zobrist program (Burmeister, 2000), also shown in Figure 3. This program selected positions at random, weighted heavily nearby the two prior moves. It took 6 minutes to query the same set, finding solutions to 3 of the 100 problems within its top 3 suggestions; 10 in its top 10; and 35 in its top 50. These numbers are not very sensitive to the precise weights used, so we take this as a baseline for how a rudimentary statistical learning algorithm would perform given only `X`, `O`, and `Q`.

There were a large number of repeated suggestions in the lists of candidate moves, as well as quite a few invalid moves, either off the board or in an occupied square. A human player would immediately filter these out, and a machine evaluation routine would also quickly eliminate them from the lookahead search; it is interesting to review the effect this has. For our system, after removing these candidates, 13 of the correct solutions were in the top 3 suggestions, 32 in the top 10, and 64 in the top 50. For the informed random player, 5 were in the top 3 suggestions, 14 in the top 10, and 51 in the top 50.

Analysis

A comparison of our analogical problem solver with the weighted random player shows that, for very small numbers of suggestions, our program performs much better, solving twice as many problems, though taking orders of magnitudes longer. At greater suggestion depths, the chance player improves relative to our program, for the trivial reason that will eventually guess every space on the board; considering more than 50 or 100 suggestions is not very practical either for input to a machine evaluation routine or as even a rough model of human candidate move generation. If we further focus on the source and target problems where the expert played within three stones of one of the last two moves (between 30 and 50 possibilities), the correct answer is in the top 10 different valid suggestions for 67 of the 100 problems. (This compares with 20 for the chance player). This shows that our analogical Go solver performs best on localized problems, which are in fact those instances where our solver has the largest amound of relevant information. This points to the fact that the representation was the biggest weakness when reasoning from exemplars.

Schema-Based Processing

The second set of experiments was designed to explore the use of schemas in memory-based problem solving. Hundreds of thousands of pairs of cases from the *ten-games* set were passed to our Structure-Mapping Engine schema induction module, and the highest-scoring schemas (after normalisation) were kept as the set *schemas-1500*. This set was then used to solve the same selection of 100 random problems as in the previous experiments, using the same procedure as described above. Next, we examined the schemas which were most effective in solving problems and repeated the induction process to investigate how well the generalisation technique captures the essential, common aspects in families of cases.

Comparison with Exemplars

The most significant result with the schema-based run was that the computations took about half as long, achieving approximately the same success rate. On the same problem set (*query-random-100*), this run took 7 hours; in the top three suggestions, the *schemas-1500* source set found answers to 7 of the 100 problems; in the top 10, 23; and in the top 50, 51. In the limit, schemas suggested the solution to 88 problems.

Two main factors explain the speed difference. Firstly, the schema cases are much smaller, about 1/3 the size. Secondly, a much smaller number of suggestions were made per problem, 3204 on average. Nonetheless, the performance at low suggestion depths was about the same. In fact, as a percentage of the total number of suggestions, the correct move was suggested 60 more often by the schemas (77 of 3204) than by the exemplars (104 of 6791). This implies two things:

- When the schema set contained a case which solved the problem, it contained a lot of cases which solved the problem.
- Schemas were more likely to make suggestions to appropriate problems than were exemplars; *i.e.* they were less likely to give wrong answers.

The first point tells us that there was even more repetition in the schema set than in the exemplar set, which might have been expected, considering that the schemas encode common patterns among the exemplar set. The second point was quite surprising, though: one might expect the schemas to be more general, and hence more applicable than the exemplars. What happened, however, was that the exemplars, because they contained so much background information, could match more situations. With many exemplar cases, analogies were based on irrelevant criteria but were still strong enough to form inferences — inappropriately — about `Q`. Asymptotically, however, exemplars solved 93 of the 100 problems; in some cases, exemplars appropriately made inferences about `Q` based on information that had been lost in the schema formation process.

After filtering out repetitions and invalid moves, the schema-based run was slightly better than the exemplar-based run. The top three suggestions gave solutions to 14 problems (compared to 13 for exemplars); the top 10 solved 39 (versus 32); and the top 50 solved 70 (versus 64). This strengthens our conclusion that in this experiment, the schema induction process was somewhat successful in discarding irrelevant, distracting information from cases and achieving better performance much more quickly.

The Effectiveness of Schemas

If this conclusion is correct, then many of the schemas should correspond to common Go situations or aphorisms. Figure 4 below shows one schema that was

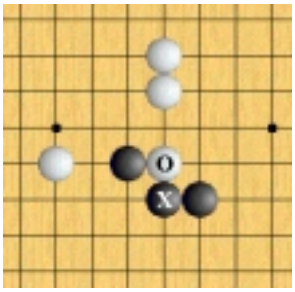

Figure 4: A particularly good schema. Playing `Q` to the left of `X` in a situation like this solved many of the target problems.

particularly useful: it gave correct suggestions to 20% of the posed problems, usually in the top 50 and several times in the top 10. Other effective schemas also helped to solve a large number of problems, to a much greater extent than individual exemplars. On the other hand, there were some problems which no schemas matched but which were matched closely and solved by some exemplars. In summary, it appears that some good schemas can effectively replace a large number of common exemplars, but that in outlying cases, exemplars are important to keep around.

An approach we are currently investigating is to lump exemplars and schemas together, developing new schemas at random (weighted by the SME match score) and adding them to the pool. Some of the high-generation schemas, *i.e.* those formed after multiple generalisations, match patterns in standard Go reference books. One of these is the principle to "hane at the head of two stones", to jump out in front of an opponent's line of stones (shown in the left of figure 5 below). Another interesting configuration left out the colour of `X` and `Q`, suggesting that whether black or white played `X` relative to the other two stones, then the other player should set `Q` down to the lower right of `X`.

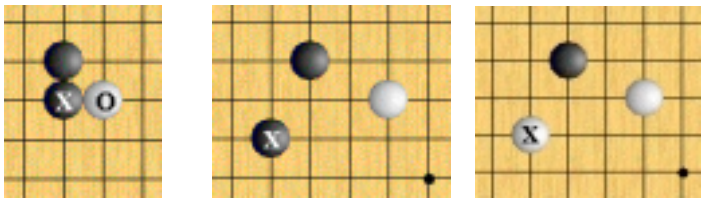

Figure 5: Some very generalised schemas. Interesting schemas found included the "hane", on the left (play just below `X`), and the loose rhombi, both on the right (play to the lower right of `X`).

However, not all the high-generational schemas correspond to nicely stated principles or even to reasonable Go play. Anytime there is a large intersection between cases, even if it is meaningless and accidental, a schema can form: for reliable, iteratable induction, a better technique is needed.

Conclusion

This memory-based model of the candidate move generation phase of Go has promising and interesting results. One of the three highest-scoring valid suggestions matched the expert's move in one out of eight problems, and the best 50 suggestions solved more than half the problems despite a simple and locally-confined representation. Still, a large number of problems could not be solved by this model, and in considering future research directions it is useful to analyse these failings.

Reminding and Richer Representations

Most of the problems that were not solved were ones where the subsequent move was in a different region of the board than either of the two previous moves. In these instances, the source exemplars simply did not store the information to enable our approach to find the answer. It seems likely that human players attend to a much wider range of features; if more of these could be recognised and encoded by the routines that generate the descriptions, we might expect better performance.

On the other hand, it is expensive to keep all this information: the cases would become too large to perform analogies on the entire set. One possible resolution would be to encode initially only the most salient features of the target; a reminding stage, such as in MAC/FAC (Forbus *et al.*, 1995), could select a small number of cases on which to perform the analogical matching. Some inferences would posit the presence of certain features in the target problem, which could be added to the initial representation if they hold, and the analogy could continue iteratively, re-representing, re-reminding, and re-matching, until it flounders or suggests something about where to play.

There is also the question of where these features will come from. Most Go programs have an extensive feature recognition routine, and it would be possible at first just to duplicate some of these. Ideally, these features could then evolve, with better ones developing as the program collects more experience. Schemas might be useful here, to replace common, structured phrases in the representation by a single reference to a schema, similar in a way to chunking. This model would be interesting to test, in Go, or in any domain where expertise might take the form of good feature recognition for case retrieval.

Evaluation and Improving Abstraction

A major criticism of this particular approach to schema formation is that it only identifies patterns in static input descriptions. It does this without any regard for the significance of stones, and so encodes a lot of useless, coïncidental substructures. An interesting AI perspective would be to grade cases and individual description lines according to their performance. A similar, more cognitive approach comes from Riesbeck & Schank (1989) who stress the importance of building logical explanations for generalisations to eliminate this sort of spurious abstraction. This is precisely what is done in the evaluation phase. While we have so

far ignored this phase as distinct and unrelated to representation and matching, it could conceivable be used to build explanations for good schemas; by storing this information as part of exemplar cases, it could also serve to enrich the representation.

General Discussion

This model gives an example of how experiences and generalisations might be used, by people and by machines, to solve difficult problems. It essentially performs pattern matching using analogy, with good initial results in a very complex domain. Additionally, it embeds some powerful logic and machine learning techniques in a cognitive framework of schema induction. The major weaknesses of our model seem to be in the simplicity of the representation and the absence of the reminding and evaluation stages. These issues, sometimes considered separate from the matching stage, must on the contrary be addressed simultaneously and in depth when developing a memory-based problem solver.

Our approach can also be viewed as finding solution categories for a target problem, as analogical problem solving and categorisation are closely related. In this light, our results suggest that it is more efficient (in terms of time, memory, and to a lesser extent success rate) to define categories on the basis of schemas when there are many similar cases. This offers circumstantial evidence in favour of multiple-prototype and theory-based views of categorisation with the added benefit of describing how schema definitions might be formed from structural and surface features of exemplars. Instead of relying on a complete set of episodic exemplars, a memory-based approach can benefit from the clustering and compression given by analogical induction and the formation of schematic (semantic?) memories.

On the other hand, our schemas did not even suggest solutions to some of the problems that were easily solved by exemplars; forming good schemas, if it is possible in most cases, is more difficult than our technique recognises. No matter what, exemplars will always be needed for those areas where experience is minimal and where categories are not neatly defined. For Go, the data suggest that the best categorisation and problem solving would be achieved by a mixed source set containing a few very general schemas, more specific schemas, and exemplars in areas not well represented by the schemas.

In conclusion, we have implemented and analysed a model of memory-based cognition — in a symbolic architecture — and applied it to complex problem solving in Go, achieving better-than-chance performance with a very limited representation. At this stage, it seems that schemas can assist but not supplant pure exemplars in this type of problem solving. It seems also that the central matching stage may be more intricately dependent on the reminding and the evaluation stage than is typically acknowledged, particularly regarding the representation. This indicates compelling research directions both for Computer Go and for the psychology of problem solving.

References[1]

Bozulich, R., ed. (1992) *The Go Player's Almanac.* San Jose: Ishi.

Burmeister, J. (2000) Research Page, `http://www.psy.uq.edu.au/~jay/`

Cheng, P. W., & K. J. Holyoak. (1985) Pragmatic reasoning schemas, *Cognitive Psychology 17*:391-416.

Daelemans, W., A. van den Bosch, & T. Weijters. (1997) Empirical Learning of Natural Language Processing Task, workshop position paper, The 9th European Conference on Machine Learning.

Falkenhainer, B., K. D. Forbus, & D. Gentner. (1989) The structure-mapping engine: algorithm and examples, *Artificial Intelligence 41*:1-63.

Forbus, K. D., D. Gentner, & K. Law. (1995) MAC/FAC: a model of similarity based retrieval, *Cognitive Science* 19:141-205.

Gentner, D. (1983) Structure-mapping: a theoretical framework for analogy, *Cognitive Science 7*:155-170.

Holyoak, K. J., & P. Thagard. (1989) *Mental Leaps.* Cambridge, MA: MIT Press.

IGS (2000) The Internet Go Server. `http://igs.joyjoy.net/`

Komatsu, L. K. (1992) Recent views of conceptual structure, *Psychological Bulletin 112*:500-526.

Kuehne, S. E., K. D. Forbus, & D. Gentner. (1999) Category Learning as Incremental Abstraction using Structure-Mapping, poster presentation, 21st Annual Meeting of the Cognitive Science Society. Vancouver.

Nosofsky, R. M. (1984) Choice, similarity, and the context theory of classification, *Journal of Experimental Psychology: Learning, Memory, and Cognition 10*:104-114.

Plotkin, G. D. (1971) *Automatic Methods of Inductive Inference*, PhD Thesis, University of Edinburgh.

Reiss, M. (2000) Mick's Computer Go Page, `http://www.reiss.demon.co.uk/webgo/compgo.htm`

Riesbeck, C. K., & R. C. Schank. (1989) *Inside Case-Based Reasoning.* Hillsdale, NJ: Erlbaum.

Saito, Y., & A. Yoshikawa (1996) A Protocol Study of Problem Solving in Go, Poster presentation, abstract in *Proceedings of the 18th Annual Conference of the Cognitive Science Society* 833.

Stoutamire, D. (1991) Machine Learning, Game Playing, and Go, Technical Report TR 91-128. Cleveland, OH: Case Western Reserve University.

[1] In addition to the referees, we are indebted to Jon Oberlander, Ian Frank (now at ETL Japan), and others in the Edinburgh Informatics community for helpful discussions; and to Ken Forbus's group at Northwestern University for their insights and the SME code. The work was funded principally by a British Marshall Scholarship.