A Concept Formation Based Algorithmic Model for Skill Acquisition

Linda Briesemeister*, Tobias Scheffer†, Fritz Wysotzki
Technische Universität Berlin, Artificial Intelligence Group, FR 5-8,
Franklinstr. 28/29, D-10587 Berlin

October 9, 1996

Abstract

We present an algorithmic model for acquisition of cognitive skills that is based on machine learning and problem solving algorithms. The principle is to use a problem solving approach for new problems that are not covered by the routine knowledge obtained from generalizing previous samples, and to use a machine learning algorithm to generalize these samples to an abstraction of the state space. We show the admissibility of our approach, discuss complexity results and present empirical results for Rubik's cube and a maze problem.

1 Introduction

The problem of skill acquisition has been approached from various perspectives, ranging from a psychological, e.g. [And93] to a machine learning point of view, e.g. [MC68, BSA83]. While most work on skill acquisition in machine learning focuses on control skills, e.g. [ML892, DBS93], this paper mainly deals with cognitive skills. Known models for cognitive skills include SOAR [LNR87] and Act-R [And93]. In contrast to SOAR, which is a mainly implementation-oriented and very complex system, that in spite of its psychological background cannot be considered a well-defined mathematical model, we aim at setting up a cognitive model from theoretically well-understood basic concepts, which enables us to obtain provable properties, possibly with the drawback of less psychological adequacy.

Our work is based on combining problem solving in the sense of artificial intelligence with machine learning. The process of problem solving, e.g. [Nil71], means finding an operator sequence, that maps an initial state to one desired goal state. A number of algorithms with well-defined properties (admissibility, optimality) are known, that can solve this problem, e.g. branch and bound algorithms [NKK84] or A* [Nil71, Gel77]. The main problem of problem solving is the high computational complexity; various approaches to speeding up this process by incorporating some kind of learning algorithm are known: STRIPS learns macro operators [FHN72] that perform "large steps" in the search space, other systems learn evaluation functions, operator strengths, and operator-selection and rejection rules (e. g. LEX [MUB83], SOAR [LNR87], PRODIGY [Min90]).

Concept formation is studied most intensively in both, psychology and artificial intelligence. The objective of concept formation is to find a description of some target concept, that generalizes a set of given positive examples, but does not cover any of the negative examples. Many psychological models of learning are based on chunking of production rules, e.g. [LRN86]. Chunking is a bottom-up learning mechanism based on abstraction from those part of some situation, that are not involved in the focused mental process. From a theoretical point of view, this operational definition is less precise and formal than mathematical models of generalization [Hau89, Plo70, Sch95a]. In

*e-mail: `xantippe@cs.tu-berlin.de`
†e-mail: `scheffer@cs.tu-berlin.de`

other psychological models of concept formation [HMS66] and machine learning, e.g. [UW81], it is often performed by specializing the most general or generalizing the most special hypothesis. A number of decision tree [UW81, Qui86, MW92] and rule extraction algorithms [MMHL86, Sch95b] perform this task efficiently, if the examples are described propositionally, i.e. in terms of attribute vectors. Propositional concept formation is well understood, a comprising theory is available [Val84].

Müller and Wysotzki [MW95] propose an approach to combining problem solving and concept formation in the context of skill acquisition. During the learning phase, their system uses a problem solver to generate optimal control actions for randomly chosen sample states, a learning algorithm then sets up a decision tree the output of which is the optimal control action for each input situation, thus generalizing on the samples shown before. During application, the system uses only the decision tree to determine actions. In contrast, the algorithm proposed in this paper has no strict distinction between a learning and an application phase. The algorithm uses learned knowledge to solve problems without search if possible, and learns new knowledge gained on-line (i.e. by search), if this is necessary.

2 A model for skill acquisition

2.1 The system's architecture

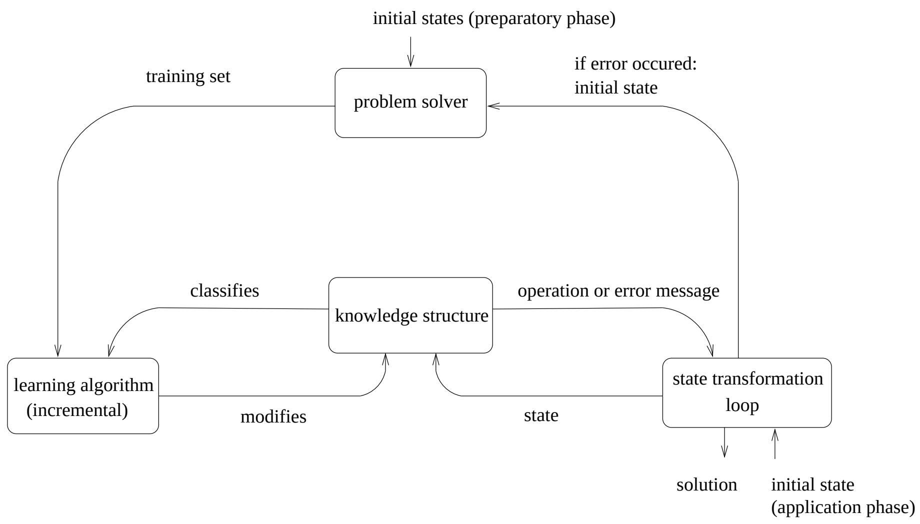

Figure 1: System architecture

We propose a model that is set up from four basic components: A problem solver, a concept formation component, a state transformation loop and a knowledge structure. Figure 1 gives an overview on the architecture. The suggested system is used in two phases: the preparatory and the application phase.

In the preparatory phase, the problem solver is applied to a given number of initial states. The solution of each initial state produced by the problem solver is a path from the initial state to a goal state. Thus, an operation which leads from one state of the path to the next one, is assigned to that state. These pairs indicate the control action (operation) which should be applied in a particular situation (state). The concept formation component consisting of a learning algorithm, then processes these pairs and induces a decision tree which is able to classify states and returns control actions (classes). The decision tree is stored as a knowledge structure to preserve the previously gained knowledge during the problem solving process.

Consider the following scenery illustrating the preparatory phase. A grocery sells five different goods: apples, bakery, cheese, dog food and eggs (see figure 2). An initial state can be seen as a given shopping list of how many items to buy. Each situation or state is a combination of the current grid square (described by an x- and y-coordinate) and the remaining list of goods which have not been put already in the trolley. An operation is either moving in one of the four directions or taking an item from a shelf, which must be located in one of the four grid squares next to the current place. Invalid actions are stepping into walls, shelves or taking an item which is not part of the remaining shopping list. The goal state is reached if the exit grid square is reached and no items remain on the shopping list.

Figure 2: Goods sold by the grocery

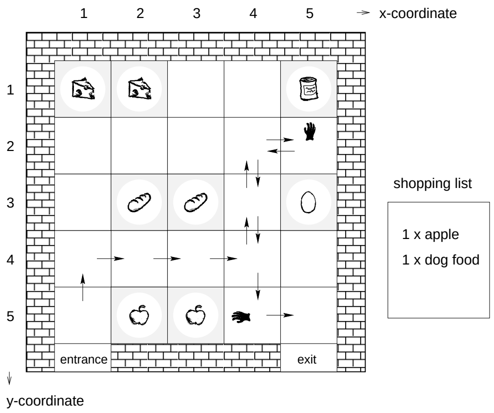

Figure 3: Visit the grocery for the first time

Suppose, the initial state seen in figure 3 is given to the problem solving component, simulating the first visit at the store to buy dog food and one apple. An optimal way through the shelves is delivered by this component (see also figure 3), split up into pairs and passed to the learning algorithm. The learning algorithm yields the decision tree printed in figure 4.

In the application phase, the system tries to solve new problems (i.e. initial states) by using the knowledge structure first. Starting with an initial state, the state transformation loop uses the state description as input for the knowledge structure, which returns the operation to be applied. Since the knowledge structure is a decision tree, the tree must be traversed up to a leaf node beginning with the root node. A node is either a leaf and contains the operation to be returned, or a test of an attribute of the state description. In this case, the value of the tested attribute indicates the branch which leads to the next node to be traversed. If there is no branch labeled with the value, the nearest neighbored branch can be taken instead to conduct some kind of classification based on state similarity. In case that no neighbored branch exists or this kind of "similarity classification" is not desired, the decision tree returns "no action" and the problem solver has to be invoked. If there is an action returned, the state transformation loop tries to apply this one to the current state, which results in a new state. The resulting state is again passed to the knowledge structure and so on. This process stops if one of the following conditions holds:

(a) A goal state is obtained.

(b) The decision tree returns "no action" or

(c) the returned action is invalid, because it leads to a forbidden state, or

(d) the number of actions exceeds a threshold n_t.

In case (a), the process halts successfully. In case (b), (c) and (d) the initial state is passed to the problem solving component to generate a solution using some search algorithm. Similar to the preparatory phase, the solution computed by the problem solver is split up into pairs. Afterwards, the concept formation component takes these pairs and incorporates the directly gained knowledge

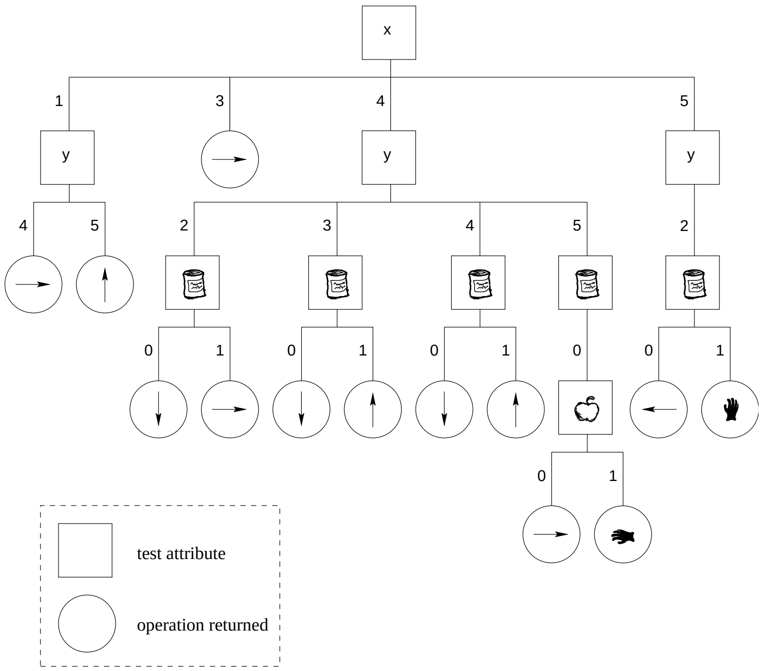

Figure 4: Decision tree induced after the first visit

(pairs) into the existing knowledge structure (decision tree). This requires an incremental learning algorithm.

Proceeding with the example shown above, the next time visiting the grocery only one apple should be bought. Entering the shop for the second time corresponds to starting the state transformation loop. As described above, the decision tree (printed in figure 4) is used recursively, which can be performed easily by the reader[1]. A test node of the tree delivers either a value of a grid coordinate or a number of an item remaining on the shopping list. After seven steps the goal state is reached and the solution path being found is printed in figure 5.

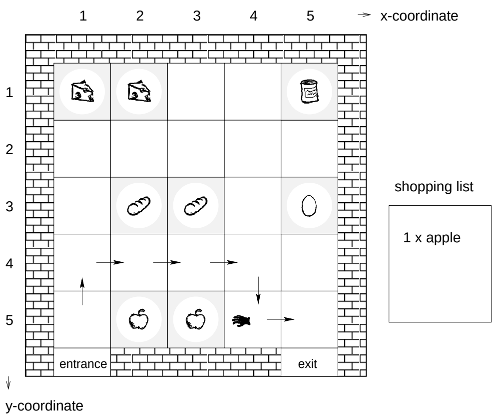

Figure 5: Second visit at the grocery

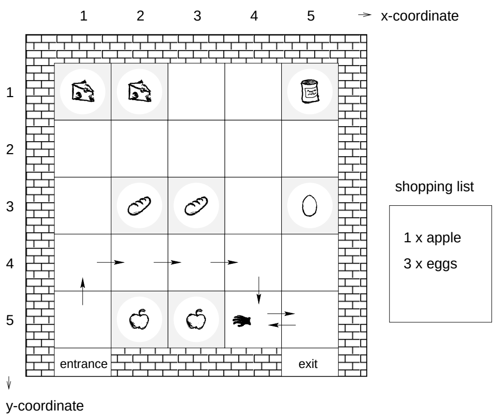

Figure 6: Third visit at the grocery

Depending on the selection of the initial state(s) used to prepare the system, more or less sufficient knowledge has been achieved to avoid problem solving during the application phase. In

[1]using "similarity classification" preferring the right neighbored branch

contrast to the just accomplished task of buying one apple, which exemplified that the knowledge was sufficient, consider now the task of buying one apple and three eggs (see figure 6). After the state transformation loop has conducted seven steps, the exit grid square is reached, but since three eggs are still remaining on the list, this situation cannot be considered as the goal state. Therefore, no condition holds to stop the state transformation loop. Passing the current state to the decision tree, it returns the action to move left. After moving left, the resulting state is classified as moving right. It can be seen easily, that a loop is performed while finding a solution. Thus, the threshold n_t is reached before the goal state is obtained, due to insufficient knowledge (refer to case (d)). The state transformation loop gives up finding a solution and passes the initial state to the problem solving component. Note, that in this particular example the usage of "similarity classification" causes the performance of a loop. In case of avoiding "similarity classification" loops can also occur while applying the knowledge structure accurately, namely in problem spaces apart from mazes (e.g. Rubik's Cube).

Considering also state space problems with an infinite state space, it is even more important to specify n_t as a fixed boundary. The system is incapable of deciding whether the sequence of states produced by the running state transformation loop will lead to a goal state or not. Only when a goal state is reached, success can be announced. Therefore, the threshold n_t has to be specified to prevent the system from running endlessly the state transformation loop. n_t should not be smaller than the optimal solution path of any initial state to give the system a chance to find a solution without problem solving.

2.2 The system's properties

It is not necessary to conduct the preparatory phase. Skipping it, the knowledge structure is empty and delivers always "no action". In this case, the first problem (initial state) that arises is immediately passed to the problem solving component (refer to case (b)) and the knowledge is built up during application time.

The system does not necessarily produce the optimal solution to a problem, even if the problem solver does. Presenting an unknown state, the knowledge structure is intended to classify it because of similarity to known states, e.g. they have some attributes in common. As soon as this kind of generalized knowledge is used, the assigned control action may be not optimal.

The proposed system is admissible in a problem solving sense, i.e. if for a given problem a solution exists at all, then the algorithm is guaranteed to find it. The proof for this statement is mainly based on arguing that either the knowledge structure produces a solution or else, if we suppose the admissibility of the problem solver, the problem solver will find a solution if a solution exists.

Proposition 1 (admissibility) *The system finds a solution for the input problem with at most n_t operations, if such a solution exists, the search algorithm is admissible and finds always an optimal solution and the learning algorithm produces a decision tree, that is correct for at least the training set.*

Proof 1 *If the system reaches a goal state in at most n_t operations by using the knowledge structure, the solution was found. Otherwise, the system exceeded the limit of n_t operations or the knowledge structure returned an invalid action or "no action". In these cases, the problem solver is called and obtains a solution, if a solution exists, because we assumed the admissibility of the problem solver. The training set consists of pairs of states and operations. As we assumed an optimal solution found by the problem solver, no state can be included more than once in the path, hence the training set is consistent and there exists a knowledge structure, that is correct w.r.t. this set. As we assumed the existence of a path of length at most n_t, the optimal solution fulfills these assumption. If the incremental learning algorithm constructs a new knowledge structure that is correct for the training set, what we have assumed, then the state loop reproduces this path in the next step.*

Suppose that for some particular scenario there is a finite state space described by n binary valued variables. Suppose that from each point in this space there is a path of at most m actions to some target state. Hence, the problem space is of size 2^n, a problem solver would have to search this space to a depth of at most m, thus the problem solver has exponential time complexity (NP-completeness) an upper limit is $O(2^{n^m})$. If we determine the optimal operator for each state and write it on a table, to solve any problem we only have to repeatedly look up the current state, perform the optimal operation until a goal state is reached. The size of such a table is 2^n, hence such an algorithm is $O(m \cdot 2^n)$. The algorithm proposed in this paper has to compute the optimal operation for each state in the worst case once, then builds a decision tree that has in the worst case one leaf node for each state and a depth of n. Hence, an algorithm that looks up optimal operations in a decision tree is only $O(m \cdot n)$. The drawback is, that the decision tree possesses order of 2^n leaf nodes in the worst case and of course the time required for learning is growing exponentially. On the other hand, if we assume that somehow the problem space is structured homogeneously, due to the generalization ability of decision tree algorithms, the decision tree should contain significantly less nodes. Even in the worst case that the problem space is chaotic, we achieved a shift of computation time for problem solving into the training phase and after the complete space is learned, problem solving can be performed with linear time complexity. But note, that the decision tree seen as a "problem solver" has no flexibility, i.e. is fully adapted to an environment represented by the training set as a set of support.

3 Generalization and memorization ability

Generalization is considered to be the ability of a system to draw correct conclusions for unknown problems from a hypothesis set up from specific sample problems. The inductive step of setting up general hypotheses from special examples clearly plays the most important role in this type of behavior. In our model, this induction splits up into a global type of generalization and memorization ability (GMA), and pure generalization ability (GA) in the sense of concept formation.

Consider the system being confronted with one initial state during the preparatory phase. If the problem solver yields a solution path consisting of m operations, m training objects (pairs) are formed and processed by the learning algorithm. During application, a new problem (i.e. an initial state) that differs from that one being used for preparation, might be solved successfully by the state transformation loop. From a global perspective, the system has then computed a solution for an unknown problem (initial state) only using the knowledge structure. This illustrates the generalization and memorization ability of the system.

Taking a closer look at the solution path produced by the state transformation loop, it is possible that it contains states which were part of the training set used to set up the knowledge structure. I.e. an initial state may be equal to an intermediate state on the solution path delivered by the problem solver during preparation. Since the system has been already presented this state previously, this can hardly be understood as a generalization ability but as a memorization ability.

For our experiments that means, that two different factors are to be considered. In terms of our system, the global generalization and memorization factor (GMA) equals the amount of successfully solved initial states of a testing set in proportion to the number of initial states used for preparation.

$$GMA = \frac{\#\text{ solved initial states (testing set)}}{\#\text{ initial states given to the problem solver}}$$

To determine the pure generalization ability (GA) in sense of concept formation, the factor is computed by dividing the amount of successfully solved initial states of a testing set by the quantity of the training set that results from splitting up the solution paths the problem solver has delivered after processing a number of initial states.

$$GA = \frac{\#\text{ solved initial states (testing set)}}{\#\text{ training objects passed to the learner}}$$

If the average solution path consists of l steps, the denominator of the GA factor will be approximately l times as large as the denominator of the GMA factor. To compare results from variations of problems, all numbers should be set in relation to the size of the corresponding problem space.

4 The implemented system

The implemented system is a general-purpose architecture, which accesses the problem specific information via interface. It has been implemented in LISP and reimplemented in C++.

As problem solver, the A* algorithm [Nil71] is used. This algorithm finds an optimal solution, if a solution exists.

As incremental learner, the CAL2 learning algorithm [UW81], pp. 25 has been chosen. It induces classification rules which are expressed in form of a decision tree. As it operates on discrete valued attributes, our system can only solve problems in a discrete problem space. CAL2 requires disjoint classes. If there are several optimal paths to the goal, one state can be classified in different ways; but as each action is optimal, it is possible to delete all but one before learning. Thus, the training set will always contain disjoint classes.

5 Experiments

The system has been applied to different problems with a finite, discrete and static state space.

The efficiency of the system during the application phase depends on the learner, the problem itself and the number of initial states solved during the preparatory phase. As the learner is fixed, the experiments focussed on the influence of the last two factors.

To test the influence of the size of the initial state set, it was enlarged successively. For each size, the preparatory phase was executed. The resulting decision tree was then applied to a fixed test set of initial states, simulating the application phase except from restarting the problem solver. The proportion of states solved successfully indicates the efficiency of the system, because for them, the problem solver does not have to be started again. To get reliable results, all state sets used during the experiments were representative selections from the problem space.

To determine the influence of the problem itself, these experiments were conducted using variations of the problem.

5.1 Rubik's Cube

The famous Rubik's Cube was developed by Ernö Rubik in 1975. Twisting the slices results in a disordered state. Finding a way back to the goal state (without using the inverse moves) is a task too complex to be solved just by trial and error: There are 43,252,003,274,489,856,000 different states and only one of them encodes the goal.

A state in the problem space is defined as a vector of the length 54 (6 surfaces $\cdot$ 9 small faces on each). Each position equals a small surface and can hold one of six colors. In total there are 9 layers, which can be moved (3 slices $\cdot$ 3 dimensions). As each layer can be turned clockwise or anticlockwise, we deal with 18 different operations.

We varied the problem by changing the *problem depth (PD)*. PD denotes the number of randomly chosen turns that are applied to the goal state to create an initial state. This state can then be solved by the problem solver in PD or less steps. We have tested problem depths from 1 to 5 moves, because the problem exceeds computational resources while problem solving for greater PD.

The results of the five experiments are shown in figure 7 and 8 and in table 1 and 2. Figure 7 and table 1 display the percentage of solved initial states from the test set as a function of the amount of initial states being used in the preparatory phase to illustrate the generalization and memorization ability (GMA) of the system (see chapter 3). For PD 1–3, the curves lie above the identity function, which is plotted additionally. Thus, more initial states were successfully processed in the state transformation loop than had been solved in the preparatory phase. Growing

PD	number of initial states during preparation	solved initial states from testing set	GMA
4	0.11%	11.0%	100
5	$< 0.019\%$	8.4%	> 442

Table 1: Results of Rubik's Cube for PD 4 and 5 (GMA)

PD	number of training objects during preparation	solved initial states from testing set	GA
4	0.35%	11.0%	31.4
5	$< 0.065\%$	8.4%	> 129

Table 2: Results of Rubik's Cube for PD 4 and 5 (GA)

rapidly in the beginning, at some point the growth is slowing down. For PD 4 and 5, table 1 summarizes the results for the largest set sizes which the implemented system was able to handle. Figure 8 and table 2 illustrate, that the number of solved test states exceeds also the number of training samples being learned. This shows, that the learner has not only memorized the given solutions but generalized them in sense of concept formation (see chapter 3).

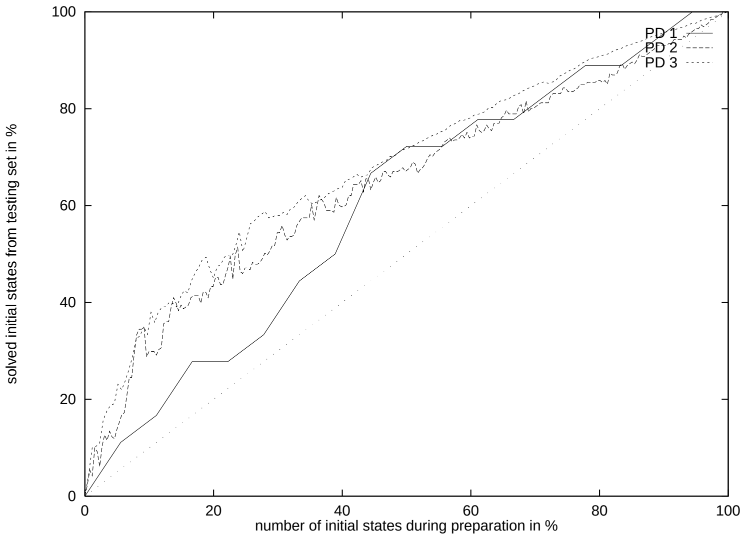

Figure 7: Results of Rubik's Cube for PD 1–3 (GMA)

As the coherence between greater PD and greater ascent of the function indicates, the extension of the problem space seems to lead to better generalization and memorization. Nevertheless, the available data is not sufficient to determine whether this is a general tendency or not.

5.2 Toad in a maze

A toad is placed into a given two-dimensional maze with only one exit. The toad has to find the exit from different places. After solving these problems the toad should find the way even from unknown positions using the obtained knowledge.

The two-dimensional state space is given by the points of the maze. The walls are represented as forbidden regions in the state space. The toad can perform one of four possible operations by

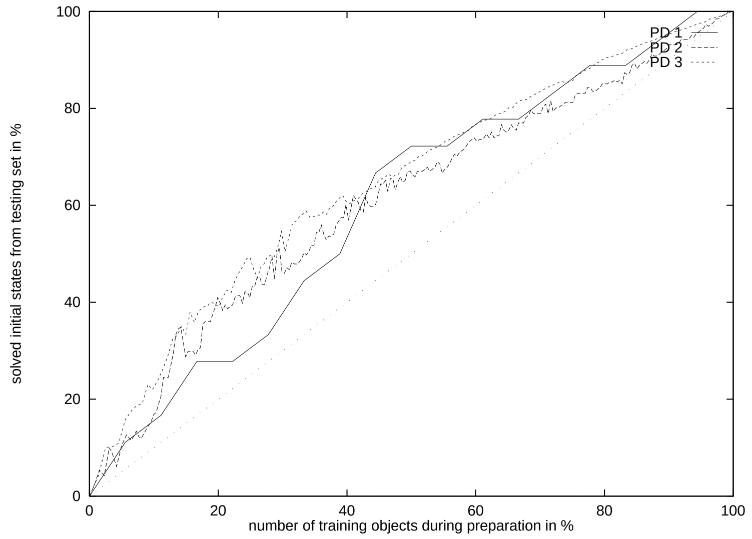

Figure 8: Results of Rubik's Cube for PD 1–3 (GA)

moving "up", "down", "right" or "left".

For the final experiments, mazes of 30 x 30 points were used. An example is shown in figure 9. The initial state set was enlarged successively until it comprised the whole state space. Due to the manageable size of the problem space, it was also possible to use the whole state space as the test set. The system has been applied to five mazes with growing complexity. Starting with a simple maze, increasingly complex mazes were generated by adding new walls successively.

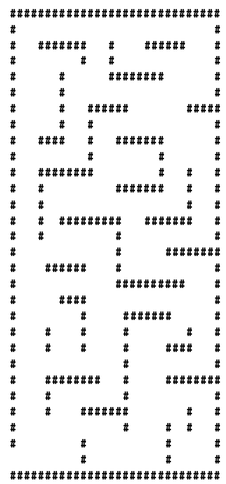

Figure 9: Maze4

The results of the experiments are plotted in figure 10 and 11. Maze1 is the simplest, maze5 the most complex maze. Figure 10 shows the percentage of correct classifications of the state space as a function of the relative size of the initial state set during the preparatory phase. All functions are significantly greater than the identity function. For all mazes, the system rapidly reaches a large percentage of correct performance, but then improves very slowly. As the problems become more complex, the speed of growth decreases: The more complex the maze, the lower the curve. As the huge differences between the curves and the identity function reveal, a large amount of knowledge is acquired while learning.

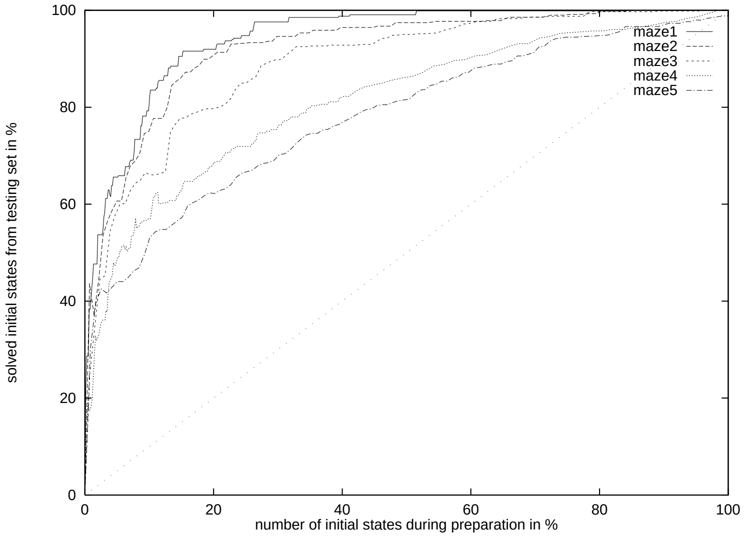

Figure 10: Results of maze problem (GMA)

Figure 11 exemplifies the coherence between the size of training sets and the amount of solved initial states. Taking a closer look at generalization abilities, the GMA (see chapter 3) which is expressed in figure 10 is much greater than the generalization ability GA in sense of concept formation visualized in figure 11. For the most complex problems maze4 and maze5, the curves of training sets with at least 65% of the whole problem space, equal the identity function, illustrating that only the training set is memorized when new problems arise. This phenomenon is not due to the system but more or less to the learning algorithm CAL2 we used: CAL2 is incapable of detecting intervals within a dimension of the state space. Since we deal with a two-dimensional state space, only areas like whole columns or rows can be assigned to one control action. If a single state within such a column or row is misclassified, the learning algorithm splits up the whole row to improve its hypothesis about the problem. For ordinary mazes, i.e. not too simple mazes, it seems more appropriate to distinguish rectangular areas by building up intervals.

Altogether, a lot of problem solving effort during the preparatory phase can be wasted, if the set of initial states is too large. It is more efficient to choose a restricted number of initial states so that a large amount of knowledge is gained through generalization. During the application phase, the few states still misclassified by the knowledge structure are solved by the problem solver. Thus, the problem solving effort is only spent when the missing knowledge is actually needed.

As the results show, the amount of generalized knowledge induced from a given set of initial states depends on the complexity and the structure of the problem. The more complex the problem, the more difficult it is to infer valid generalization from the information provided by the same percentage of the state space. Therefore, the best size of the initial state set has to be determined individually for each variation of a problem.

6 Conclusion and further research

We proposed an algorithmic model for skill acquisition that is based on a combination of problem solving and machine learning. We have shown that this system is admissible in a problem solving sense and we discussed, that the time complexity for problem solving is reduced from exponential to linear if the state space is finite, can be enumerated and a decision tree algorithm is used. We discussed, that the system possesses an inherent ability of generalizing and memorizing problems. We presented empirical results for the Rubik's cube and a maze problem.

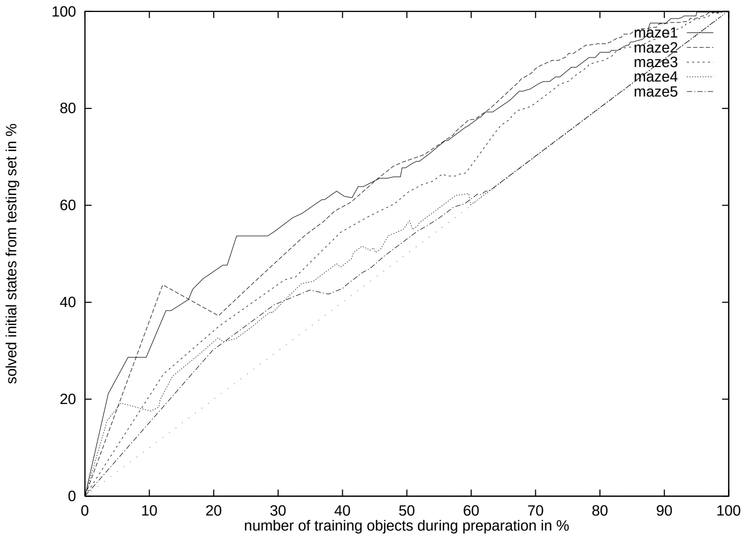

Figure 11: Results of maze problem (GA)

Although Rubik's Cube is a very complex and difficult task for problem solving algorithms, some results are achieved by using only a small part of the whole problem space to teach the system. Surprisingly, the knowledge was sufficient not only to memorize previously seen states but to solve totally unknown configurations of the cube. Therefore, the system must have used pure generalization in sense of concept formation, e.g. for configurations which are obtained after twisting the cube's slices four times, this generalization ability is given by a factor of 31.4. A closer look on the decision trees being set up by the learner, gives the possibility to explain the way the system models the structures of the problem.

Dealing with much smaller state spaces than Rubik's Cube, the maze problem hardly exceeds computational resources while finding a way to the exit. Therefore, the system was trained piece-wise up to the whole problem space to study the behavior in all phases. The best generalization factor is reached in the very beginning; later the lack of concept formation is due to the fact that the learning algorithm is not able to represent the structure of the mazes. More sophisticated learning algorithms could lead to much better results.

If the problem space is finite and the learning algorithm is correct for at least the training set, the system can be prepared with the whole state space, provided the computational resources are not exceeded. In this case, the learned knowledge is sufficient to solve any new problem without restarting the problem solver. The response time for a control action can be guaranteed by the time the decision tree needs to classify a state. Assuming additionally that the problem solver yields the optimal solution, the gained knowledge is optimal as well. This property shows the usability for real-time systems with controlling tasks.

For future research, connecting the problem solver itself with the knowledge structure seems to model human behavior in a better way. The main problem here is, that the problem solver might then interact with an inconsistent knowledge structure and yet is required to be admissible. Additionally, it would be interesting to address problem spaces with structural states together with more powerful hypothesis languages, like graphs or first order logics, for the learning algorithm.

ACKNOWLEDGMENT

We wish to thank especially Barbara van Schewick, who participated in the experiments with the system and who was always a valuable partner when discussing the results, and Jörn Freydank, who participated in the implementation of the system.

References

[And93] J.R. Anderson. *Rules of the Mind.* Lawrence Erlbaum, Hillsdale, NJ, 1993.

[BSA83] A. B. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. In *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

[DBS93] T. Dean, K. Basye, and J. Shewchuk. Reinforcement learning for planning and control. In S. Minton, editor, *Machine Learning Methods for Planning*, 1993.

[FHN72] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Gel77] D. Gelperin. On the optimality of A*. *Artificial Intelligence*, 8:69–76, 1977.

[Hau89] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, 1989.

[HMS66] E. B. Hunt, J. Marin, and P. T. Stone. *Experiments in Induction.* Academic Press, New York, 1966.

[LNR87] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[LRN86] J.E. Laird, P.S. Rosenbloom, and A. Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[MC68] D. Michie and A. Chambers. Boxes: An experiment in adaptive control. In *Machine Intelligence 2*, 1968.

[Min90] S. N. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363–392, 1990.

[ML892] Machine learning. vol. 8(3-4): Special issue on reinforcement learning, 1992.

[MMHL86] R. S Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045, 1986.

[MUB83] T. M. Mitchell, P. E. Utgoff, and R. B. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine learning: An artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.

[MW92] Wolfgang Müller and Fritz Wysotzki. Automatic construction of decision trees for classification. *Annalen für operations Research*, 1992.

[MW95] Wolfgang Müller and Fritz Wysotzki. Automatic synthesis of control programs by combination of learning and problem solving methods (extended abstract). In Nada Lavrač and Stefan Wrobel, editors, *Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning, 1995)*, Lecture Notes in Artificial Intelligence 912, pages 323 – 326, Berlin, Heidelberg, New York, 1995. Springer Verlag.

[Nil71] N. Nilsson. *Problem Solving in Artificial Intelligence.* McGraw Hill, 1971.

[NKK84] D. S. Nau, V. Kumar, and L. Kanal. General branch and bound and its relation to A* and AO*. *Artificial Intelligence*, 23:29–58, 1984.

[Plo70] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163, 1970.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[Sch95a] T. Scheffer. Beschreibungsunabhängiges Lernen und Generalisieren. In K. Morik and J. Hermann, editors, *Beiträge zur Fachtagung Maschinelles Lernen*, 1995.

[Sch95b] T. Scheffer. Learning rules with nested exceptions. In *Proc. International Workshop on Artificial Intelligence Techniques*, Brno, Czech Republic, 1995.

[UW81] S. Unger and F. Wysotzki. *Lernfähige Klassifizierungssysteme.* Akademie Verlag Berlin, 1981.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.