In Proceedings of IJCAI'03. ©IJCAII

Learning Minesweeper with Multirelational Learning

Lourdes Peña Castillo
Otto-von-Guericke-University
Magdeburg, Germany

Stefan Wrobel
Fraunhofer AIS, Sankt Augustin
and University Bonn, Germany

Abstract

Minesweeper is a one-person game which looks deceptively easy to play, but where average human performance is far from optimal. Playing the game requires logical, arithmetic and probabilistic reasoning based on spatial relationships on the board. Simply checking a board state for consistency is an NP-complete problem. Given the difficulty of hand-crafting strategies to play this and other games, AI researchers have always been interested in automatically learning such strategies from experience. In this paper, we show that when integrating certain techniques into a general purpose learning system (Mio), the resulting system is capable of inducing a Minesweeper playing strategy that beats the winning rate of average human players. In addition, we discuss the necessary background knowledge, present experimental results demonstrating the gain obtained with our techniques and show the strategy learned for the game.

1 Introduction

Minesweeper is a popular one–player computer game written by Robert Donner and Curt Johnson which was included in Microsoft Windows© in 1991. At the beginning of the game, the player is presented with a $p \times q$ board containing pq tiles or squares which are all blank. Hidden among the tiles are M mines distributed uniformly at random on the board. The task of the player is to uncover all the tiles which do not contain a mine. At each turn the player can select one of three actions (moves): to mark a tile as a mine; to unmark a tile; and to uncover a tile. In the last action, if the tile contains a mine, the player loses; otherwise, the number of mines around the tile is displayed. In the 4×4 board depicted in Fig. 1 left, the number 1 located on the second row from top indicates that there is one and only one mine hidden among the eight blank neighbouring tiles.

Although the simplicity of its rules makes Minesweeper look deceptively easy, playing the game well is indeed challenging: A player requires logic and arithmetic reasoning to perform certain moves given the board state, and probabilistic reasoning to minimize the risk of uncovering a mine when a safe move cannot be done. Given the difficulty of hand-crafting playing strategies for this and other games, AI researchers have always been interested in the possibility of automatically learning such strategies from experience. However, with the exception of reinforcement learning [Tesauro, 1995], most of the playing strategies and heuristics used in game playing programs are coded and tuned per hand instead of automatically learned. In this work, we use a general purpose ILP system, Mio, to learn a playing strategy for Minesweeper. Multirelational learning or ILP consists in learning from examples usually within the framework provided by clausal logic.

The task of learning rules to deduce Minesweeper moves proved itself to be an arduous test for current multirelational learning systems. In this paper, we describe how recent optimizations make possible for Mio to discover a Minesweeper playing strategy. Experimental results obtained by playing Minesweeper using this strategy show a better performance than that obtained on average by non-expert human players.

The remainder of this paper is organized as follows. The next section discusses the complexity of the game and describes the learning task. Section 3 describes the background knowledge, the learning system and the new techniques used. Section 4 shows our empirical results on the effectiveness of the learning techniques, the strategy obtained and its performance at game playing. Related work is surveyed in Section 5 and Section 6 concludes.

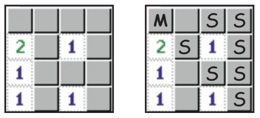

Figure 1: Left: Available information on a board. Right: Seven tiles can be determined safe (s) and one a mine(m)

2 Minesweeper

2.1 Why Is Minesweeper Interesting?

Minesweeper has been shown to be NP-complete by simulating boolean circuits as Minesweeper positions [Kaye, 2000]. Kaye describes the *Minesweeper consistency problem* as the problem of determining if there is some pattern of mines in the blank squares that give rise to the numbers seen in a given board partially filled with numbers and marked mines, and thus determining that the data given is consistent.

One realizes the complexity of the game by calculating an estimate for the size of its search space. Consider an 8×8 board with $M = 10$ mines; in this case at the beginning of the game the player has $pq = 64$ tiles from which to choose a move (i.e., a tile to uncover) and in the last move, assuming the player does not uncover a mine, there are 11 tiles from which to choose one. This leads to $54! \approx 10^{71}$ possible move sequences to win a game. Alternatively, one can calculate the probability of a random player winning a game. In the first move the probability that the random player chooses a tile which does not contain a mine is $54/64$, and in the last move it has $1/11$ chance to choose the only tile without a mine. Then, the probability of a random player winning a game is $1/\binom{64}{54} \approx 10^{-12}$ and that is only for the easiest playing level!

Another measure of the complexity of Minesweeper is the number of games won on average by non-expert human players. To estimate the average human performance playing Minesweeper, we carried out an informal study. In the study, eleven persons who have played Minesweeper before were asked to play at least ten times in an 8×8 board with 10 mines. Every participant was told to aim for accuracy rather than for speed. In this study, a person won on average 35% of the games with a standard deviation of 8%.

2.2 The Learning Task

In Minesweeper there are situations that can be "solved" with nontrivial reasoning. For example, consider Fig. 1 left where the only available information about the board state are the numbers. After careful analysis one finds that the squares with an s (see Fig. 1 right) do not contain a mine, the square with an m is a mine, and the state of the blank tiles cannot be determined if we do not know how many mines are hidden in the board. There are other Minesweeper situations where the available information is not enough to identify a safe square or a mine, as in Fig. 2, and the best option available to the player is to make an informed guess, i.e., a guess that minimizes the risk of blowing up by uncovering a mine.

In this work, we consider the learning task in Minesweeper to be the induction of rules to identify all the *safe squares*[1] and squares with a mine which can be deduced given a board state. For instance, we want the system to learn rules to classify all the blank tiles in Fig. 1 either as *safe* or *mine*.

3 The Learning Tools

In machine learning it is possible to choose between a propositional representation (in the form of attribute-value tuples) and a multirelational representation (in the form of logic predicates). A multirelational representation has the expressiveness required to describe the concepts involved when reasoning about Minesweeper, and is thus more intuitive than the propositional one. For this reason we use multirelational learning for the learning task described above. Usually a multirelational learning system takes as input background knowledge B, positive (E^+) and negative (E^-) examples of a target concept such that $E = E^+ \cup E^-$ and $E^+ \cap E^- = \emptyset$, and has to find a clausal theory T which minimizes the classification error on future instances. Next we describe the background knowledge and the system used.

[1]A safe square is a blank tile which given the current board state cannot contain a mine.

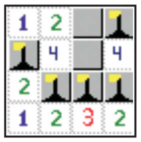

Figure 2: The position of the last mine cannot be determined

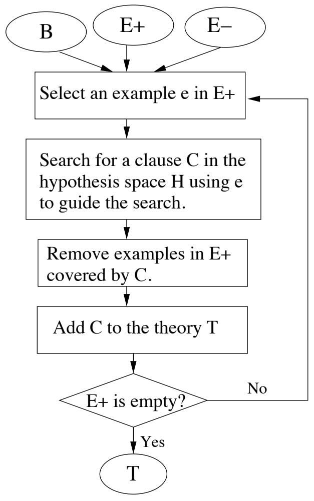
B
E+
E–
Select an example e in E+
Search for a clause C in the hypothesis space H using e to guide the search.
Remove examples in E+ covered by C.
Add C to the theory T
E+ is empty?
No
Yes
T

Figure 3: Example-driven covering algorithm

3.1 Background Knowledge

The background knowledge provides the system with information about the domain and is given in the form of logic predicates (facts and rules, or clauses). A predicate is described as $name(arg_1, \ldots, arg_n)$ and, in our case, arg_i indicates the argument's type. The background knowledge provided to the learning system about Minesweeper is shown in Table 1. The predicates in the background knowledge were defined by trying to abstract the concepts used by humans when explaining their own Minesweeper playing strategies. These concepts were obtained from the first author's Minesweeper playing experience and from Minesweeper pages on the web.

In the predicates listed in Table 1, a *TD* is a determined or uncovered tile, i.e., a number $0 \ldots 8$ is shown on the tile; a *TU* is an undetermined or blank tile; *Board* is the board state given as a list of $p \times q$ characters $0 \ldots 8, m, u$; *Zone* is a list of determined tiles, and *Set* is a set of undetermined tiles together with the number of mines hidden among those tiles. In addition, each symbol preceding an argument denotes how that argument should be instantiated when calling the predicate. A $+arg_i$ is an input argument and should be instantiated to a non-variable term; a $-arg_i$ is an output argument and should be uninstantiated, and a $\#arg$ indicates that the constant value of an output argument can appear in a rule.

Predicate	Description
zoneOfInterest(+TU, +Board, -Zone)	returns in Zone the tiles which are determined neighbours of TU and the determined tiles which share an undetermined neighbour with them (see Fig. 4 center).
totalMinesLeft(+Board,-Int)	returns how many mines remained to be marked.
allMinesInFringe(+Board, -Set)	gives the set of tiles in the fringe[3] where all the remaining mines are.
setHasXMines(-TD, +Board, +Zone, -Set)	gives in Set the undetermined neighbours of TD (TD is in Zone), and the number of mines hidden among them (see Fig. 4 right).
diffSetHasXMines(+Set1, +Set2, -Set)	returns in Set all and only the tiles of Set1 which are not also in Set2 and the number of mines hidden among the tiles in Set.
inSet(+TU, +Set)	is true when TU is a member of Set.
lengthSet(+Set,+Int)	is true when Set contains Int tiles.
minesInSet(+Set,-#Int)	returns the number of mines hidden among the tiles in Set.

Table 1: Minesweeper background knowledge

3.2 The System

Mio is an example-driven covering system (see Fig. 3) introduced by Peña Castillo and Wrobel [2002b] which uses a Progol-like declaration language and, the same as Progol [Muggleton, 1995], lower bounds the search space with a *most specific clause* $\perp$ (also called bottom clause). This $\perp$ is a maximally specific clause which entails (covers) a positive example e. Mio performs a general-to-specific (top-down) IDA* [Korf, 1985] search to find a clause to add to the theory. In addition, Mio selects stochastically the examples from which it learns, performs parallel search to increase the stability of the example-driven approach, and enforces type strictness. Three other techniques are implemented in Mio to allow the learning of Minesweeper rules: macro-operators (or macros, for short) to reduce the search space, greedy search with macros to speed up the learning process, and active learning to guide the exploration of the instance space.

Macros

Macros in multirelational learning [Peña Castillo and Wrobel, 2002a] are a formal technique to reduce the hypothesis space explored by a covering system. A macro is a sequence of literals, chosen from the bottom clause, which is created based on provider-consumer relationships between the literals. A literal is a provider if it has output arguments, and it is a consumer if it receives as an input argument a variable provided by another literal in the bottom clause. Peña and Wrobel show that by adding macros instead of literals to a clause, the number of clauses evaluated by the system is significantly reduced.

Greedy Search with Macros

In [Peña Castillo and Wrobel, 2002a] macros are used with IDA*. It is well known that greedy search explores on average less nodes than IDA*; however, greedy search could miss a solution because it underestimates the importance of provider literals without discriminative power which are nonetheless necessary to introduce new variables (this is known as the myopia problem). Since macros add several literals at once to a clause, they might reduce the myopia of greedy search allowing us to gain in efficiency without losing too much in effectiveness. We implement a greedy search with macros which consists of a lookahead step where all the macros are combined with each other and the best evaluated clauses are selected. Then if the selected clauses can be extended (refined) the system tries to combine these clauses with all the macros available and selects the best candidates. This last step is repeated until there is no clause which can be extended and the best candidates are returned.

[3]Fringe refers to all the blank tiles with a determined neighbour.

Active Inductive Learning

In the covering algorithm a clause is learned which covers (explains) some positive examples and none (or few) negative ones; however, in domains such as games and puzzles, thousands of examples are required to contain most of the possible game situations; on the other hand, considering thousands of examples when evaluating a rule slows down the learning process. Thus, to improve the efficiency of the exploration of the instance space, active learning [Cohn *et al.*, 1994] is included in Mio.

Active inductive learning consists of the following steps. At the beginning, Mio learns from few randomly drawn examples and when it has learned some clauses gives these clauses to an active learning server. The active learning server returns to Mio *counterexamples*[4]. These counterexamples are selected from examples given by a random example generator (or random sampler). While Mio iterates on the new examples received, the server tests the rules obtained against randomly drawn examples, discards all the rules below a user-defined accuracy value and collects new counterexamples. This validation step on the server side avoids overfitting[5]. These steps are repeated until a user-defined maximum number of iterations is reached or no counterexample is found.

Active inductive learning is similar in spirit to integrative windowing [Fürnkranz, 1998] with two main differences: in our approach random sampling is done dynamically and a client-server architecture is used which allows to treat testing and learning as separated processes.

[4]A counterexample is a positive example not covered by a set of clauses T or a negative example covered by at least one clause in T.

[5]Overfitting refers to obtaining results with high classification error over new data despite null or almost null training error.

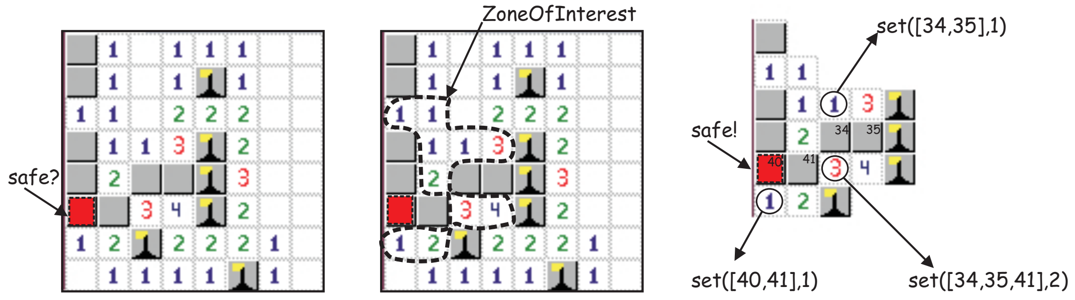

Figure 4: A rule learned by Mio. Left: Is the highlighted tile safe? Center: zoneOfInterest corresponding to the highlighted tile. Right: Applying difference operations to the sets determined by the tiles inside a circle is concluded that the tile 40 is safe

4 Empirical Results

4.1 Improvements Obtained with each Technique

Experiments were carried out to determine the effects on the rules obtained and on the system efficiency, of macros, greedy search with macros, and active inductive learning. To produce the training examples, we randomly generate board configurations and take all blank tiles with at least one determined neighbour as examples. If the blank tile does not contain a mine is labeled as safe, otherwise it is labeled as mine. Afterwards, contradictory examples are removed. In the experiments, the learning task was to learn rules to identify safe tiles. The rule to discover mines was learned using the best setting (i.e., using active learning, macros and greedy search).

For the completeness of this work, we ran Progol [Muggleton and Firth, 2001] and Foil [Quinlan and Cameron-Jones, 1995] on the same learning task as Mio; however, we failed to make the systems learn correct rules about safe tiles. Progol search was interrupted due to search limits implemented in the system (although the maximum stack depth and resolutions steps were set to 50000 and 10000, respectively), and Foil pruned determined literals which are needed. This might imply that the optimizations included in Mio are indeed necessary to learn a Minesweeper playing strategy. Table 2 shows the empirical results with four Mio settings.

All the experiments with active learning were performed with the same seeds which means that the same training examples are generated by the random sampler and that Mio selects the same examples to guide the search. For the experiments without active learning, we took five random samples from the set of examples used in the active learning experiments. The size of the sample is equal to the number of examples received by Mio when performing active learning (40 positive and 34 negative examples). We carried out an extra experiment where Mio was given the complete set of examples (2890 positive and 1306 negative) used by the active learning server to test Mio's rules and select counterexamples; however, this experiment was stopped after Mio ran for 10 days. To reduce the running time of the experiments, we set the maximum number of clauses explored per search to 4000 clauses.

Table 2 shows that each optimization added to Mio reduces the average number of rules (nodes) explored per search and the number of times the search is interrupted because of the search limit. Without active learning, overfitting occurs and erroneous rules are obtained. The performance of the rules obtained with IDA*+M is worse than that of the IDA* rules because Mio with macros explores a larger part of the hypothesis space and thus the IDA* + M setting overfits more the training data. However, if no limit in the maximum number of clauses explored per search is set, both settings (IDA* and IDA*+M) obtain the same rules. The running time of the fastest setting (AL+GS+M) is 56hrs.

4.2 Rules Learned

Table 3 shows the rules with the highest winning rate which were obtained by using both AL+GS+M and AL+IDA*+M. An extra rule was obtained with the latter setting; however, this extra rule does not improve the playing performance. One important feature of the rules learned by Mio is that they can be applied independently of the size of the board and the number of mines. The rules vary in complexity. Rule S-1 and Rule M-1 correspond to the trivial situations where a determined tile needs k mines and k mines are already marked, and where a determined tile needs k mines and it has k blank neighbours, respectively.

On the other hand, Rule S-3 can be seen as one of the most complex rules because it involves three determined tiles to deduce a safe tile. Fig. 4 left shows a board state where Rule S-3 is the only one which allows to identify a safe tile. The rule obtains the zoneOfInterest corresponding to the undetermined tile considered (Fig. 4 center). Then by applying difference operations on the sets determined by three uncovered tiles from the zoneOfInterest (see Fig. 4 right), the set $([40], 0)$ is obtained and thus it is deduced that tile 40 is safe.

4.3 Game Playing

To evaluate the performance at game playing of each set of rules obtained, we used each set of rules as the playing strategy of an automatic Minesweeper player and calculated the percentage of games won by the player in 1000 random games (see Table 2). The playing conditions were the same as the ones presented to the human players; i.e., at the beginning the player is presented with an empty 8×8 board with $M = 10$ mines and can uncover a mine in the first move. Note that in most Minesweeper implementations, one never hits a mine in the first move.

	Mio			
	IDA*	**IDA*+M**	**AL+IDA*+M**	**AL+GS+M**
Ave. No. clauses explored per search	3135	1462	1008	566
% of searches which reached 4000-clause limit	70%	23%	4%	3%
Ave. No. of rules obtained	2.8	4	4	3
Ave. % of games won with rules	42.6%	34.0%	52.4 %	52.0%

Table 2: Performance of various Mio settings used to learn rules about safe tiles (AL = Active Learning, GS = Greedy Search, IDA* = Iterative Deepening A*, M = Macros)

Rules about Safe Tiles

```
                          %% RULE S-1 %%
safe(TILEUK,BOARD):-
 zoneOfInterest(TILEUK,BOARD,ZONE), setHasXMines(TILEK,BOARD,ZONE,SET),
 inSet(TILEUK,SET), minesInSet(SET,0).
                          %% RULE S-2 %%
safe(TILEUK,BOARD):-
 zoneOfInterest(TILEUK,BOARD,ZONE), setHasXMines(TILEK0,BOARD,ZONE,SET0),
 setHasXMines(TILEK1,BOARD,ZONE,SET1), diffSetHasXMines(SET1,SET0,SET3),
 inSet(TILEUK,SET3), minesInSet(SET3,0).
                          %% RULE S-3 %%
safe(TILEUK,BOARD):-
 zoneOfInterest(TILEUK,BOARD,ZONE), setHasXMines(TILEK0,BOARD,ZONE,SET0),
 setHasXMines(TILEK1,BOARD,ZONE,SET1), setHasXMines(TILEK2,BOARD,ZONE,SET2),
 diffSetHasXMines(SET1,SET0,SET3), diffSetHasXMines(SET2,SET3,SET4),
 inSet(TILEUK,SET4), minesInSet(SET4,0).
```

Rules about Mines

```
                          %% RULE M-1 %%
mine(TILEUK,BOARD):-
 zoneOfInterest(TILEUK,BOARD,ZONE), setHasXMines(TILEK,BOARD,ZONE,SET),
 inSet(TILEUK,SET), minesInSet(SET,INT), lengthSet(SET,INT).
```

Table 3: Minesweeper rules learned by Mio

Let us analyze the performance of the best rule set (see Table 3). In 1000 games, the player made 15481 moves from which 2762 where random guesses, 169 used Rule S-1, 10285 used Rule S-2, 54 used Rule S-3, and 2211 used Rule M-1.

In addition, we examined the effect of adding probabilistic reasoning. In the experiment, we instructed the player using the rules shown in Table 3 to select a tile which minimizes the probability $P(T_u)$ that an undetermined tile T_u is a mine when none of the rules can be applied. $P(T_u)$ is equal to $max_{T_d}(\frac{f_m(T_d)}{f_n(T_d)})$ where T_d is a determined neighbour of T_u, $f_m(T_d)$ returns the number of mines needed by T_d and $f_n(T_d)$ returns the number of blank neighbours of T_d. Every time the player has to guess, it selects the tile which minimizes $P(T_u)$. This player wins 579 of 1000 random games.

5 Related Work

5.1 Minesweeper Playing Programs

There are several Minesweeper programs available on the web. These programs are not learning programs but playing programs where the authors have embedded their own game playing strategy. Among these programs, John D. Ramsdell's PGMS is quite successful winning 60% of 10000 random games in a 8×8 board with 10 mines.

PGMS plays using the *Equation Strategy* based on finding approximate solutions to derived integer linear equations, and probabilities. As mentioned by Ramsdell [2002], PGMS represents the information available on the board as a set of integer linear equations. Associated with an undetermined tile is a variable x that has the value 1 if the tile hides a mine, or 0 otherwise. An equation is generated for each uncovered tile with an adjacent undetermined tile. Each equation has the form $c = \sum_{i \in S} x_i$, where S is a set of undetermined tiles, and c is the number of mines hidden among S. To simplify notation, this equation is written as $c \doteq S$. Since the total number of hidden mines is known, an additional equation simply equates this number with the sum of all of the undetermined tiles.

Every time a tile t is determined safe or a mine, the board changes are propagated to all the equations containing t and a new equation for the undetermined neighbours of t is added. In addition, if $c_0 \doteq S_0$ and $c_1 \doteq S_1$ are two equations such that S_0 is a proper subset of S_1, the equation $c_1 - c_0 \doteq S_1 \setminus S_0$ is added. To determine whether a tile is safe or a mine, PGMS iteratively applies the following rules until none are applicable[7]:

[7]In these rules $|S|$ is the cardinality of S and $S_0 \setminus S_1$ is the difference between S_0 and S_1.

View publication stats

- If $0 \doteq S$, all tiles in S are safe.
- If $c \doteq S$ and $c = |S|$, all tiles in S are a mine.
- Let $c_0 \doteq S_0$ and $c_1 \doteq S_1$ be two equations and t_u be an undetermined tile such that $c_0 < c_1$, and $t_u \in S_0$ and $t_u \in S_1$. If $c_1 - c_0 = |S_1 \setminus S_0|$, all the tiles in $S_1 \setminus S_0$ are a mine and all the tiles in $S_0 \setminus S_1$ are safe.

PGMS must guess when presented with a board to which none of the rules apply. For each tile t it computes the value $P(t)$ as follows. Given an equation $c \doteq S$, define its single equation probability to be $c/|S|$. $P(t)$ is equal to $max_{t \in S}(c/|S|)$. PGMS picks the tile t that minimizes $P(t)$. A random choice is made when there is more than one tile that minimizes $P(t)$.

We were surprised to notice that although Mio was only given general background knowledge about Minesweeper, the rules it learned are similar to the rules programmed in PGMS. For example, Mio's Rule S-1 and Rule S-2 correspond to the first and third rule in PGMS, respectively; and Mio's Rule M-1 is similar to PGMS second rule. To compare PGMS performance with the performance of Mio's best playing strategy, we let our best player (i.e., the player using the rules in Table 3 and probabilities) play 10000 random games in a 8×8 board with 10 mines. Its winning rate is also 60%.

5.2 Multirelational Learning for Games

Other work has been done which applies ILP systems to learn heuristics or playing strategies for games. Ramon et al. [2001] used Tilde [Blockeel and Raedt, 1998] to learn a theory that predicts the value of a move in Go. Morales [1996] applied the system PAL to learn chess patterns for constructing chess playing strategies. Nakano et al. [1998] presented an approach to generate an evaluation function for Shogi mating problems using ILP.

6 Conclusions and Future Work

In this paper we described how the use of new ILP techniques such as macros, greedy search with macros, and active inductive learning allow Mio to learn a Minesweeper playing strategy. This learning task proved itself to be a challenging testbed for general purpose multirelational learning systems.

The best rules obtained by Mio win 52% of the games in a 8×8 board with 10 mines, while on average a non-expert human player wins 35% of the games. The performance of the playing program using these rules as playing strategy improves to 60% when adding the use of probabilities.

By examining the games played using Mio's rules, we notice that there are still situations where the player guesses without need (i.e., a sure move can be deduced). As future work, we want to use other ILP systems (e.g., Tilde), and other machine learning approaches to learn Minesweeper playing strategies and compare their performance.

Acknowledgments

We thank all the persons who participated as guinea pigs playing Minesweeper, the anonymous reviewers for their comments, and Oscar Meruvia for proofreading. The second author was partially supported by DFG (German Science Foundation), projects WR40/1-3 and WR40/2-1.

References

[Blockeel and Raedt, 1998] Hendrik Blockeel and Luc De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[Cohn *et al.*, 1994] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[Fürnkranz, 1998] Johannes Fürnkranz. Integrative windowing. *Journal of Artificial Intelligence Research*, 8:129–164, 1998.

[Kaye, 2000] Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, Spring 2000.

[Korf, 1985] Richard E. Korf. Iterative-deepening A*: An optimal admissible tree search. In *Proc. of the 9th IJCAI*, pages 1034–1036, 1985.

[Morales, 1996] Eduardo Morales. Learning playing strategies in chess. *Computational Intelligence*, 12(1):65–87, 1996.

[Muggleton and Firth, 2001] Stephen Muggleton and John Firth. Relational rule induction with CProgol4.4: a tutorial introduction. In *Relational Data Mining*, pages 160–187. 2001.

[Muggleton, 1995] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.

[Nakano *et al.*, 1998] Tomofumi Nakano, Nobuhiro Inuzuka, Hirohisa Seki, and Hidenori Itoh. Inducing Shogi heuristics using inductive logic programming. In *Proc. of the 8th Int. Conf. on ILP*, pages 155–164, 1998.

[Peña Castillo and Wrobel, 2002a] Lourdes Peña Castillo and Stefan Wrobel. Macro-operators in multirelational learning: a search-space reduction technique. In *Proc. of ECML'2002*, pages 357– 368, 2002.

[Peña Castillo and Wrobel, 2002b] Lourdes Peña Castillo and Stefan Wrobel. On the stability of example-driven learning systems: a case study in multirelational learning. In *Proc. of MICAI'2002*, pages 321–330, 2002.

[Quinlan and Cameron-Jones, 1995] J. Ross Quinlan and R. Michael Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing, Special issue on ILP*, 13(3-4):287–312, 1995.

[Ramon *et al.*, 2001] Jan Ramon, Tom Francis, and Hendrik Blockeel. Learning a tsume-go heuristic with TILDE. In *Proc. of the 2nd Int. Conf. Computers and Games*, pages 151–169, 2001.

[Ramsdell, 2002] John D. Ramsdell, November 25, 2002. Personal communication.

[Tesauro, 1995] Gerald Tesauro. Temporal–difference learning and td–gammon. *Communications of the ACM*, 38(3):58–68, 1995.