Evaluating the Effectiveness of Exploration and Accumulated Experience in Automatic Case Elicitation

Jay H. Powell[1], Brandon M. Hauff[2], and John D. Hastings[1]

[1] University of Nebraska at Kearney,
Dept. of Computer Science & Information Systems,
Kearney NE 68849, U.S.A
hueljh@hotmail.com, hastingsjd@unk.edu
[2] University of Nebraska at Lincoln,
Dept. of Computer Science & Engineering,
Lincoln NE 68588, U.S.A
brandon@genxian.com

Abstract. Non-learning problem solvers have been applied to many interesting and complex domains. Experience-based learning techniques have been developed to augment the capabilities of certain non-learning problem solvers in order to improve overall performance. An alternative approach to enhancing pre-existing systems is automatic case elicitation, a learning technique in which a case-based reasoning system with no prior domain knowledge acquires knowledge automatically through real-time exploration and interaction with its environment. In empirical testing in the domain of checkers, results suggest not only that experience can substitute for the inclusion of pre-coded model-based knowledge, but also that the ability to explore is crucial to the performance of automatic case elicitation.

1 Introduction

Non-learning problem-solving algorithms are commonly used to solve problems in a variety of complex and challenging domains including the application of alpha-beta search to checkers [1] and the use of the null-move heuristic in chess [2]. Such non-learning, non-adaptable algorithms are sufficient for domains that are either simple, static, or deterministic, but are incapable of adapting to changing environments. For domains that are sufficiently complex or dynamic, it has been argued that a system capable of learning and adapting to its environment is needed [3]. DeJong and Schultz [4] describe a technique for designing and implementing architectures for extending the capabilities of non-learning systems by automatically extending the knowledge bases of static problem solvers. Unfortunately, the process of enhancing a pre-existing problem solver is complicated by the fact that interfacing with the underlying problem solver can be difficult, and by the fact that overall problem-solving abilities can be hampered by the abilities of the underlying problem solver.

An alternative approach to augmenting a non-learning problem solver is for a system to acquire knowledge automatically without the need for predefined domain knowledge. One existing technique for the automatic capture of knowledge without a reliance upon prior domain knowledge is automatic case elicitation [5]. Automatic case elicitation is a case-based reasoning (CBR) technique that relies on the system's ability to explore its domain in real time through trial and error in order to acquire knowledge from scratch. Due to its exploratory capabilities and case-based knowledge acquisition techniques, automatic case elicitation is particularly well suited to learning in domains with observable outcomes (e.g. robot navigation or game playing).

This paper extends initial research on automatic case elicitation detailed in Powell et al. [5]. In contrast to this previous research, the methodologies described in this paper make use of a probabilistic approach to case selection to ascertain the value of exploration in an unknown environment in the context of automatic case elicitation. This paper compares automatic case elicitation against non-exploring experience based learning techniques to further determine the merit of free exploration within an automatic case elicitation system.

We detail automatic case elicitation in Section 2, and follow in Section 3 with a brief overview of extending non-learning systems through experience-based learning. We compare the two approaches in Section 4. Section 5 sets forth an empirical evaluation that demonstrates that experience can substitute for predefined knowledge, and that exploration is crucial to the performance of automatic case elicitation. We close with a discussion of related work in Section 6.

2 Automatic Case Elicitation

Automatic case elicitation (ACE) is a learning technique whereby a CBR system automatically acquires knowledge (in the form of cases) from scratch during real-time trial and error interaction with its environment without reliance on pre-coded domain knowledge (e.g. rules or cases). A probabilistic reinforcement learning approach is utilized to evaluate the effectiveness of each case (acquired or stored) after an interaction is complete, providing a means for an ACE system to learn and improve from experience. The use of reinforcement learning [6] allows an automatic case elicitation system to be used in environments which are capable of being explored as well as allow for the system to learn from its experiences (e.g. autonomous robot navigation or game playing). For implementation purposes, a case contains the following:

1. an observation or snapshot of the environment,
2. the action taken in response to the observation, and
3. a rating of the success of the applied action in meeting a goal.

Figure 1 illustrates the procedure *Ace*, the primary reasoning module within a system using automatic case elicitation. *Ace* operates on the sequence of observations (O_1 through O_n) made during interaction with the environment and completes at the point at which the effectiveness of the interaction can be determined (e.g. in chess, the effectiveness of an interaction will be determined at the completion of a game). For each observation of the environment (O_i), the system selects and applies actions (A) suggested by its case library until a change in the environment is observed. The process of selecting and applying an action is as follows. First, the system finds and loads the set

```
Procedure Ace( )
    $C$ := case base
    $AC$ := $\phi$        ; applied cases
    While success of interaction unknown Do
        $O_i$ := ObserveEnvironment( )
        $M$ := MatchingCases($C$, $O_i$)
        Repeat
            $A$ := Decision($M$)
            ApplyAction($A$)
            $O_j$ := ObserveEnvironment( )
        Until $O_i \neq O_j$
        $AC$ := $AC \cup$ Case($O_i$, $A$)
    End While
    $AC$ := Evaluate($AC$)
    Store($C$, $AC$)
End Ace
```

Fig. 1. ACE Algorithm for Interacting within an Environment

```
Function Decision(var $M$ : matching cases) : Action
    If $M = \phi$
        $A$ := NewAction( )
    Else If Rating($M_0$) $\geq$ Random(0..1)
        $A$ := ExtractAction($M_0$)
    Else
        $M$ = $M$ - $M_0$
        $A$ := Decision($M$)
    End If
    Return $A$        ; action to take
End Decision
```

Fig. 2. ACE Algorithm for Determining the Appropriate Action

of all cases whose observation closely[1] matches the current observation. If the current situation is novel or sufficiently distant from prior experience, the set of matching cases returned will be empty. If the current scenario has previously been encountered, the system makes use of an indexing scheme which counts the distinct elements in the case to quickly retrieve all cases which most closely correspond.

Once an ACE system has determined the set of matching cases (possibly empty), it calls the function *Decision*, illustrated in Figure 2. *Decision* selects and returns an action to apply. *Decision* may need to be invoked multiple times in order for the agent to formulate a legal action. In automatic case elicitation, the legality of an action is not determined by the system itself, but by the environment in which the system is interfacing (e.g. a chess engine). In other words, an ACE system attempts an action and observes whether changes to the environment occur in response. If so, the system

[1] The current implementation of ACE handles only exact matches, but will in the future support "close" matches in a domain-independent fashion.

has entered a new situation, and will react accordingly. Otherwise, it would attempt a different action.[2]

The action returned by *Decision* depends on M, the set of matching cases given by *Ace*. When M is empty (i.e., the system has encountered a novel situation), *Decision* generates a new random action. When applying random actions to a new situation, *Ace* repeatedly calls *Decision* until a valid random action (i.e., one which affects the environment) is found. This technique for the generation of new actions through random exploration is utilized because an ACE system does not rely on any pre-coded domain knowledge. Without the dependence on pre-coded knowledge or exterior problem solving techniques, random exploration is necessary for the acquisition of the minimum knowledge to operate in a given domain.

If cases are found in the case-base which correspond to the current state of a system's environment, the *Decision* module determines which of the returned cases to apply, if any. A case from the set M (arranged from M_0 to M_n in descending order based upon case ranking) is chosen on a pseudo-random basis to encourage exploration. The probability $P(M_0)$ of the most successful case in the set, M_0, being selected is equal to the case's rating, the derivation of which will be discussed later. The probability of the system iterating deeper through the list of matching cases is $1-P(M_0)$. The probability of case M_i being selected is

$$P(M_i) = (1 - P(M_0)) \times (1 - P(M_1)) \times ... \times (1 - P(M_{i-1})). \tag{1}$$

If the highest-rated case M_0 is not selected for reuse, then it is removed from the set M and $Decision$ is called again. If the entire set of matching cases is searched and no case has been chosen for reuse, a new random action is created using the process described above. This approach differs from previous implementations of automatic case elicitation which made use of a win/loss ratio, as compared to the current use of probability to select cases for reuse. The motivation for implementing this new case selection algorithm was to encourage exploration and the subsequent growth of the system's knowledge base.

Once an ACE system has created an action, it applies the action and observes the resulting consequences. If changes in the environment are observed, the ACE system remembers the action (along with the observation of the environment). For new situation/action pairs, a new case is created. Reused cases are simply remembered so that their success rating can be updated.

Upon the completion of the interaction, *Evaluate* is called to update the ratings of each applied case. Each case is rated according to its success in attaining a goal at the completion of the interaction using the formula

$$r_n = \begin{cases} \frac{1}{2}s_0 & \text{if n = 0,} \\ \frac{1}{2}s_n + \frac{1}{2}r_{n-1} = (\frac{1}{2})^1 s_n + (\frac{1}{2})^2 s_{n-1} + ... + (\frac{1}{2})^{n+1} s_0 & \text{if n> 0} \end{cases} \tag{2}$$

[2] Disregarding actions which do not immediately result in an observable change in the environment is less than desirable for domains in which a combination of actions are needed to affect a single change in the environment. For such domains, a case structure that encapsulates a sequence of actions is likely required.

In (2), r_i represents the rating (between 0.0 and 1.0 inclusive) and s_i represents the outcome (1 for success, 0 for failure) of the ith application of a case within the environment. The purpose of this formula is to provide a decaying memory representation of each case, where the consequences of applying a case early in the system's life (when cases are applied with little thought) are quickly forgotten. In contrast to work on forgetting complete cases [7,8], only the older applications of a case, not the cases themselves, are forgotten by mathematically diminishing their affect on the case rating. The rating of each case initially tends to fluctuate near 0.5 early in the system's life, while success or failure is equally probable and the system is simply attempting to learn valid domain behavior. As the system gains experience, the case's ranking can tend towards either 1 (highly successful) or 0 (completely ineffectual). Upon completion of *Evaluate*, the ratings for each of the applied cases, *AC*, have been updated and any new cases are committed to the case library using the procedure *Store*.

3 Extending Non-learning Systems Through Experience-Based Learning

DeJong and Schultz [4] describe the use of experience-based learning in improving the capabilities of non-learning systems by automatically extending the knowledge bases of static problem solvers. In their approach, actions in the knowledge base are initially suggested by the underlying problem solver. Over time, their system applies only those experiences proven to produce the best results. They illustrate that proper application of experience-based learning algorithms in combination with an underlying static problem solver can lead to the development of a system capable of quickly recalling and applying actions from the knowledge base.

To demonstrate their approach, they made use of the system GINA, an experience-based learning Othello game-playing agent. GINA relied upon a static minimax lookahead agent as the foundation of the system's experience base. When GINA encountered a scenario which did not exist in its knowledge base, it was able to consult its underlying problem solver for advice and commit the given advice to memory. At the conclusion of each game, a minimax algorithm was used to apportion credit to every move used during the game that could be found in the agent's experience base. In their paper, the authors suggested that their approach could be applied to other domains with success similar to theirs.

4 Comparison of Methodologies

The primary purpose of this paper is to compare and contrast automatic case elicitation against the technique by DeJong and Schultz [4] in order to demonstrate the power of exploration. For the purposes of testing, the performance of the two approaches is compared in the domain of checkers. A DeJong agent was created which can play checkers. Automatic case elicitation is demonstrated in the system CHEBR (CHeckers case-Based Reasoner), a system in which CBR agents utilize automatic case elicitation to learn and test their expertise in the game of checkers.

Several key differences exist between the DeJong approach and automatic case elicitation in the knowledge acquisition process. A DeJong system relies upon an underlying problem-solver and thus begins its life with pre-existing domain knowledge. When a situation is encountered that is novel or sufficiently distant from prior experience, such a system can query its problem-solver for pre-programmed guidance. When a scenario is encountered that is similar to previous experience, a system can refer back to its knowledge base for advice. At the end of each game played, a minimax algorithm is used to distribute credit to each individual move, based on how the move affected the rest of the game.

A CHEBR agent begins its life with no prior domain knowledge. Domain knowledge is acquired through a process of trial and error interaction with the checkers environment, rather than relying upon pre-programmed decision-making capabilities. In its infancy, a CHEBR agent will perform many incorrect actions until valid behavior is encountered, as dictated by the environment. Valid actions taken by the agent (in this case specific checkers moves) are stored as cases and committed to the agent's case-base, along with a rating which is used as a predictive measure of the case's future worth. In CHEBR, all experiences are stored as cases, instead of generalizations of experiences or environment states. When a CHEBR agent assigns credit to an action, it assigns credit based on the final outcome of the interaction with the environment, rather than apportioning credit based on how the move affected the rest of the game. As a CHEBR agent gains experience, the need to rely upon arbitrary move generation is greatly reduced as the requisite behavior for survival is stored in the agent's case-base.

The power and flexibility of a CHEBR agent is tied in part to its ability to acquire knowledge from scratch. With no pre-programmed domain rules, a CHEBR agent is given free reign to explore its environment. This is contrasted by the limited abilities of an agent designed around a pre-existing problem solver. Static underlying problem solvers can be inherently inflexible, due to the fact that their capabilities are hard-coded. Relying upon the decision-making skills of agents with limited flexibility in novel situations can hinder the ability of an agent to derive unique or "creative" solutions to new situations. An agent given the power to explore freely has the potential to generate inventive solutions to previously un-encountered situations.

5 Results

Automatic case elicitation (through the system CHEBR) was tested in repeated two hour training sessions against a static lookahead agent without any experience-based learning augmentations, as well as the DeJong agent with experience-based learning capabilities.

Figure 3 illustrates the winning percentages of CHEBR in competition with a standard four-ply lookahead agent utilizing a minimax algorithm and alpha beta pruning. Figure 4 illustrates the winning percentages of CHEBR in competition with a DeJong agent that makes use of the same four-ply lookahead agent as its core. The results shown were duplicated through repeated training sessions with a slight variability in results due to the use of random move generation on the part of CHEBR. As the lookahead and DeJong agent's shown made use of predefined domain knowledge, they were able

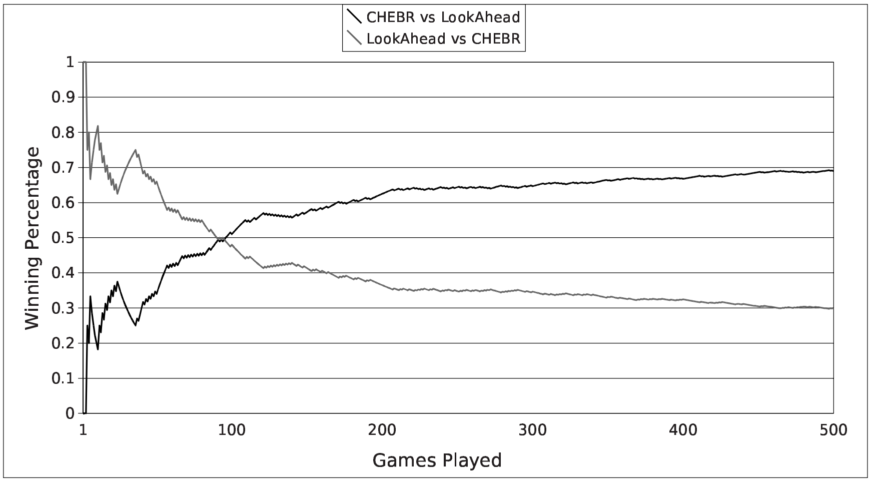

Fig. 3. Agent Win/Loss Ratio for CHEBR vs. Lookahead

to defeat CHEBR a significant portion of the time during the initial stages of game play while CHEBR was learning and exploring its environment. However, after approximately one hundred to two hundred games (about one-tenth of each training session), the winning rates of CHEBR and its opposition converged. For the remainder of each training session, CHEBR's acquired knowledge, through exploration, was sufficient to clearly defeat its opponents a majority of the time. The eventual slow growth rate of CHEBR's win ratio could be caused by overtraining against each particular opponent.[3]

As illustrated by Figures 3 and 4, it was slightly more difficult for CHEBR to adapt to the DeJong agent and defeat it as compared to the lookahead agent, helping to confirm the results of DeJong and Schultz, which state that augmenting an pre-existing problem solver using experience-based learning can create a more capable reasoner than the underlying system alone.

We believe that CHEBR's ability to defeat the DeJong approach lies in its ability to explore. To support this argument, a non-exploring version of CHEBR (Non-Explore CHEBR) was created. In Non-Explore CHEBR, the abilities to explore by applying random move selection as well as random move generation were removed and replaced with a four-ply lookahead. Figure 5 illustrates the winning percentages of CHEBR in competition with Non-Explore CHEBR. The results tentatively confirm that automatic case elicitation in CHEBR depends heavily on the ability to explore.

Although CHEBR began its life with no prior domain knowledge (cases), it proved capable of acquiring knowledge about its environment through repeated exploration and interaction with its environment. As CHEBR gained experience and acquired knowledge, it was able to learn the behavior required to succeed. Further training allowed CHEBR to refine its case-base sufficiently to defeat each of its opponents a majority of

[3] CHEBR is generally quick to adapt to new opponents. However, the speed with which CHEBR is able to conquer new opponents is diminished as the size of the case library becomes extremely large.

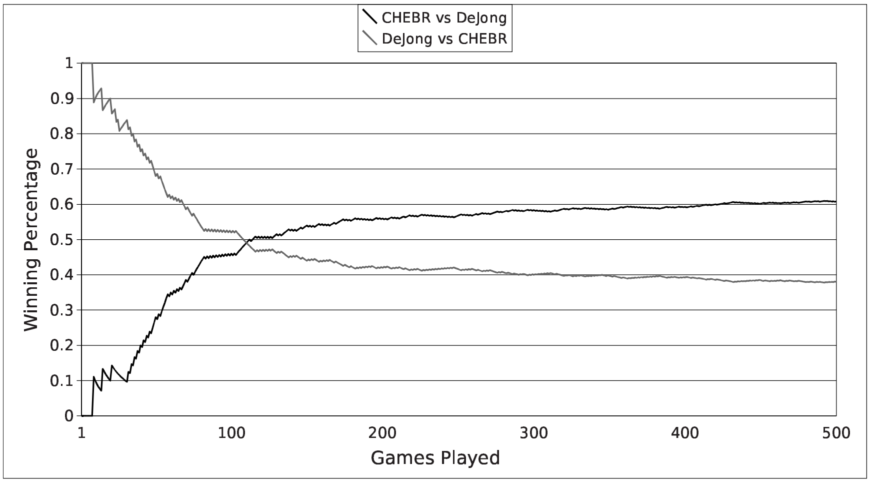

Fig. 4. Agent Win/Loss Ratio for CHEBR vs. DeJong

the time. CHEBR's ability to explore allowed it to locate and exploit the weaknesses of its opponents, and as a result created a challenging and adaptable game player. The inability of the lookahead and DeJong agents to explore due to a fundamental reliance on a static rule-based system prevented them from responding to new situations created by CHEBR. The results suggest that experience can substitute for the inclusion of pre-coded model-based knowledge as seen in the success of CHEBR against the DeJong agents (which use model-based knowledge). The results further suggest that the ability to explore is crucial to the performance of automatic case elicitation which relies primarily on its ability to acquire new experiences.

6 Related Work

Previous work has investigated the automatic generation of cases from predefined expert knowledge. For example, the planning system SHOP/CCBR [9] automatically acquires cases from manually entered project plans. A related approach has been seen in chess games [10,11] which use CBR for chess play by automatically generating case libraries from sets of pre-existing grandmaster games. Shih [12] integrates CBR and the idea of sequential dependency to learn bridge play from a set of existing games. In contrast, automatic case elicitation does not compile cases from manually entered or existing data, but instead acquires knowledge automatically through the experiences of the agents who learn completely from scratch.

CBR has also seen use in a real-time games. For example, Fagan and Cunningham [13] describe the use of case-based plan recognition to predict a player's actions in real time interaction with the game Space Invaders. Construction of the plan library is delayed until after the player has played the game three times, although it would seem possible that the system would not require such a delay. The authors suggest that their

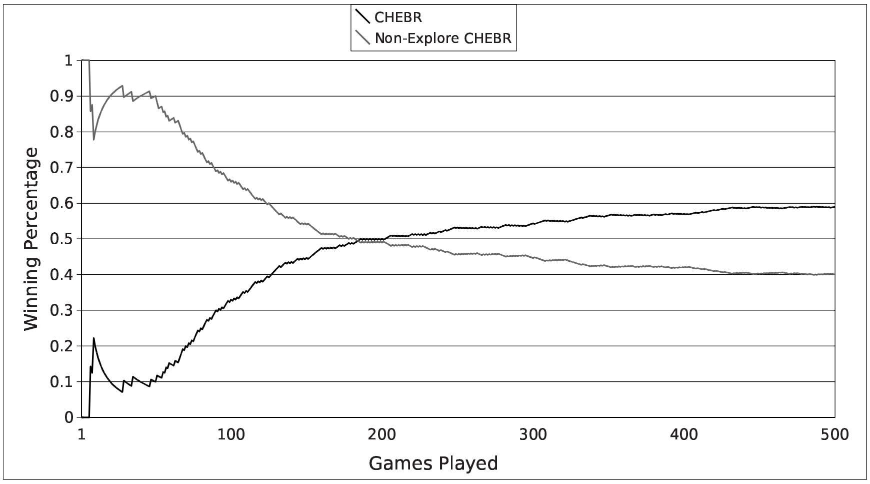

Fig. 5. Agent Win/Loss Ratio for CHEBR vs. Non-Exploring CHEBR

approach could be extended to adjust the behavior of non-player characters, although such an action selection mechanism is not present.

Wendler and Lenz [14] employ CBR in a real-time setting to appropriately position soccer agents based on previously collected cases. Their agents learn during the game and adapt their behavior accordingly. In contrast to our approach, Wendler and Lenz do not use CBR as their sole reasoning technique.

MAYOR [15] is a player of the simulation game SimCity and is based on a predefined understanding of an incomplete world model. A case-based planner complements the world by using a library of plans manually built prior to game play. In automatic case elicitation, cases are gathered in real time and are used as the sole reasoning mechanism.

Goodman [16,17] describes the use of off-line built decision-tree induction projectors to predict the outcome of various actions during game play in Bilestoad. Automatic case elicitation differs in that agents learn in real time and projection is not coded as a separate step but is instead encapsulated within individual case ratings.

Samuel [18,19] describes the use of rote-learning and argues that a program can learn to play a domain better than the creator. A lookahead of two or three plays is used to find moves to be scored for a checker game. Samuel's approach requires that the game must have at least one intermediate goal. Automatic case elicitation differs in that it does not require intermediate goals, instead utilizing only the final success rating of the interaction with its environment. In addition, automatic case elicitation does not use a pre-existing reasoner such as that described by Samuel.

Likhachev et al. [20] use CBR to tune the parameters used to guide a robot through obstacles. The cases provide a mapping from mathematical sensor input to sensor parameters that guide a robot. Over time, the results of applying a case are used to further fine tune the contained parameters. Similar to our approach, Likhachev et al. use randomness to encourage exploration. In an extension to this work, Kira and Arkin [21] describe the use of forgetting as a means to compensate for a limited case library size when moving the robot to different environments. Our approach in a sense makes use of a forgetting mechanism inherent to the case rating. The approach described by Likhachev et al. and Kira and Arkin is relatively domain specific. In contrast, we feel our approach is generally applicable in a wide variety of domains.

7 Conclusion

For domains that are sufficiently complex or dynamic, a system capable of learning and adapting to its environment is needed. One approach is to extend the capabilities of a non-learning system with a mechanism that automatically records and evaluates experiences. An alternative known as automatic case elicitation supports the automatic capture of knowledge from scratch in real time without a reliance upon prior domain knowledge. In testing in the domain of checkers, an agent using automatic case elicitation (CHEBR) was shown to successfully defeat opponents using a standard lookahead agent, and an agent using experience-based learning with an underlying lookahead agent. In addition, CHEBR minus the ability to explore was shown to perform at a lower level than when using full automatic case elicitation. The results suggest not only that experience can substitute for the inclusion of pre-coded model-based knowledge, but also that the ability to explore is crucial to the performance of automatic case elicitation.

References

1. Schaeffer, J.: One Jump Ahead: Challenging Human Supremacy in Checkers. Springer Verlag (1997)
2. Beal, D.F.: A generalised quiescence search algorithm. Artificial Intelligence **43** (1990) 85–98
3. Grefenstette, J.J., Ramsey, C.L.: An approach to anytime learning. In: Proceedings of the Ninth International Machine Learning Workshop, San Mateo, CA, Morgan Kaufmann (1992) 189–195
4. DeJong, K.A., Shultz, A.C.: Using experience-based learning in game-playing. In: Proceedings of the Fifth International Conference on Machine Learning, San Mateo, California, Morgan Kaufmann (1988) 284–290
5. Powell, J.H., Hauff, B.M., Hastings, J.D.: Utilizing case-based reasoning and automatic case elicitation to develop a self-taught knowledgeable agent. In Fu, D., Orkin, J., eds.: Challenges in Game Artificial Intelligence: Papers from the AAAI Workshop (Technical Report WS-04-04), AAAI Press (2004) 77–81
6. Kaelbling, L.P., Littman, M.L., Moore, A.P.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4** (1996) 237–285
7. Smyth, B., Keane, M.T.: Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In: Proceedings of the 14th International Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada (1995) 377–382
8. Watanabe, H., Okuda, K., Fukiwara, S.: A strategy for forgetting cases by restricting memory. IEICE Transactions on Information and Systems (1995) 1324–1326

9. Mukkamalla, S., Muñoz-Avila, H.: Case acquisition in a project planning environment. In: Proceedings of the Sixth European Conference on Case-based Reasoning (ECCBR-02), LNAI 2416, Springer-Verlag (2002) 264–277
10. Flinter, S., Keane, M.T.: On the automatic generation of case libraries by chunking chess games. In: Proceedings of the First International Conference on Case Based Reasoning (ICCBR-95), LNAI 1010, Springer Verlag (1995) 421–430
11. Sinclair, D.: Using example-based reasoning for selective move generation in two player adversarial games. In: Proceedings of the Fourth European Workshop on Case-Based Reasoning (EWCBR-98), LNAI 1488, Springer-Verlag (1998) 126–135
12. Shih, J.: Sequential instance-based learning for planning in the context of an imperfect information game. In: Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR-01), LNAI 2080, Springer-Verlag (2001) 483–501
13. Fagan, M., Cunningham, P.: Case-based plan recognition in computer games. In: Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCBR-03), LNAI 2689, Springer Verlag (2003) 161–170
14. Wendler, J., Lenz, M.: CBR for dynamic situation assessment in an agent-oriented setting. In Aha, D.W., Daniels, J.J., eds.: Case-Based Reasoning Integrations: Papers from the AAAI Workshop (Technical Report WS-98-15), Madison, WI, AAAI Press (1998)
15. Fasciano, M.J.: Real-time case-based reasoning in a complex world. Technical Report TR-96-05, Computer Science Department, University of Chicago (1996)
16. Goodman, M.: Projective visualization: Acting from experience. In: Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), Menlo Park, Calif., AAAI Press (1993) 54–59
17. Goodman, M.: Results on controlling action with projective visualization. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), Menlo Park, Calif., AAAI Press (1994) 1245–1250
18. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal on Reseach and Developement **3** (1959) 211–229
19. Samuel, A.L.: Some studies in machine learning using the game of checkers, ii – recent progress. IBM Journal on Reseach and Developement **11** (1967) 601–617
20. Likhachev, M., Kaess, M., Arkin, R.C.: Learning behavioral parameterization using spatio-temporal case-based reasoning. In: Proceedings of the 2002 IEEE International Conference on Robotics and Automation. Volume 2. (2002) 1282–1289
21. Kira, Z., Arkin, R.C.: Forgetting bad behavior: Memory management for case-based navigation. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (2004) 3145–3152