Reinforcement of Local Pattern Cases for Playing Tetris

Houcine Romdhane & Luc Lamontagne

Department of Computer Science and Software Engineering
Université Laval
Pavillon Adrien-Pouliot, Québec (Québec), Canada, G1K 7P4
{houcine.romhdane, luc.lamontagne}@ift.ulaval.ca

Abstract

In the paper, we investigate the use of reinforcement learning in CBR for estimating and managing a legacy case base for playing the game of Tetris. Each case corresponds to a local pattern describing the relative height of a subset of columns where pieces could be placed. We evaluate these patterns through reinforcement learning to determine if significant performance improvement can be observed. For estimating the values of the patterns, we compare Q-learning with a simpler temporal difference formulation. Our results indicate that training without discounting provides slightly better results than other evaluation schemes. We also explore how the reinforcement values of the patterns can help reduce the size of the case base. We report on experiments we conducted for forgetting cases.

1. Introduction

Game AI has been growing in the last few years as a topic of interest both for the industrial and academic communities. Most games have some kind of decision components, and AI techniques are increasingly used to implement them. Case based reasoning (CBR) is foreseen as a candidate approach as it would provide an efficient mean to memorize gaming experiences that could later be reused by some non-player characters (NPC).

However this introduces some challenges to the current state of the art of CBR. Cases usually represent individual episodes of problem solving instances. As most of the games are sequential in nature, a purely episodic representation would only depict partial situations. Also many tactical and reactive games impose some forms of time dependent limitations on a game AI component. Very few initiatives have been taken to introduce time constraints in CBR reasoning. Efficient management of the case base would be one of the main issues to impact on the performance of CBR systems due to its nearest-neighbor style of retrieval. Management policies would be required to tackle these problems.

In this paper, we investigate to what extent reinforcement learning (RL) can contribute to the management of a legacy case base containing local patterns for playing a sequential game. We use the game of Tetris as an application due to the dimensionality of the decision space which makes it a complex but tractable problem to solve. Also this game present the interesting property of being subject to time limitations as decisions must be taken before a dropping figure reaches the surface of the board.

Our interest for this work is on the evaluation of prior cases. We assume that cases are already provided to the CBR system either by an external program or by some human player. Through offline training, we make use of the reinforcement values to assess the quality of the cases and to guide the forgetting of cases to control the size of the case base. This differs from reinforcement learning efforts in CBR assuming online learning without prior knowledge (Sharma et al., 2007) (Gabel et al., 2005).

In the next sections of this paper, we propose a representation for structuring local patterns for playing Tetris with a CBR component. We then assess the contribution of reinforcement learning to evaluate the quality of the patterns. We compare Q-learning with a simple temporal difference formulation without discounting to estimate the value of each individual pattern. We finally perform some experiments for forgetting cases and estimating what level of degradation can be observed by reducing the size of the case base using reinforcement values as a pruning criterion.

2. Playing Tetris with Local Pattern Cases

Tetris is one of the oldest video games. It consists of placing a dropping piece onto a set of columns to complete rows and avoid accumulation of pieces. Seven different shapes of pieces exist in the game. Placing each of them involves various combinations of rotation and sliding.

From a computational point of view, Tetris is an interesting laboratory for experimenting with CBR as it is a complex game with a solution space of 10 columns and 20 lines. The problem space is estimated to 2^{200} and its solving through analytic methods is NP-complete (Breukalaar et al., 2004). Moreover, this game presents interesting time constraints as a decision must be made before the piece touches the upper row of cubes.

In our initial experiments, we cumulated examples using the Tieltris extension of the TIELT testbed (Molineaux et al., 2005) and tried to apply these in a CBR fashion by comparing the heights of all the 10 columns. As reported in section 6.1, this global approach reveals inefficient, mainly due to the large number of situations that might arise in this game. However we make use of these results as a baseline for our work.

An interesting observation is that when someone plays Tetris, he sweeps through the surface and tries to locate an adequate subset of columns where the dropping figure could be placed. This is representative of a local scheme to play this game. Local assessment approaches have been proposed for games, and more specifically for complex board games such as Chess (Campbell et al., 2002), Checkers (Schaeffer, 2005) and Go (Silver et al., 2007). They offer the advantage of reducing the complexity of the problem space while providing good approximation for potential solutions.

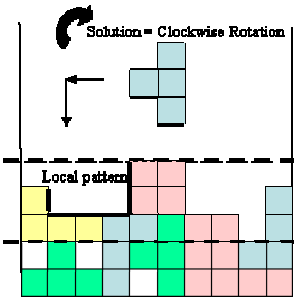

Figure 1 – Using local patterns for playing Tetris.

To reproduce such a local approach in CBR, we converted our legacy cases as a set of patterns indicating where blocks would be placed on a set of columns. These represent height patterns that can be used for deciding where to place blocks on a new configuration.

In this setting, a case describing a local pattern is represented as follows (Figure 1):

- *Problem*: it consists of the type of piece, its orientation and the *local pattern* where the piece was placed. The width of the patterns corresponds to those of the figures to be placed.
- *Solution*: the rotation that should be applied to the piece to fit it in the local pattern. For instance, a piece could be turned clockwise and moved 2 columns on the left.
- *Value*: the improvement to be expected by applying to the piece on the columns according to the solution. This can either correspond to the reward obtained from Tetris or be estimated by some other evaluation scheme.

In the paper, we try to get insights on the following three questions:

- What CBR performance can be expected from a set of local patterns?
- What level of play can be reached by estimating the value of the patterns through some reinforcement scheme?
- As Tetris is a real-time game, what degradation of performance can be expected by reducing the size of case base in order to meet time constraints imposed by the game?

To address these issues, we adopt a reinforcement learning approach to assign values to case and to make use of them during the case management phase.

3. General Approach

The problem solving CBR cycle we used for playing Tetris is described in Figure 2. This cycle contains the usual phases of CBR, i.e. retrieval, reuse and revision. The case maintenance phase is not elicited in the cycle as it is currently deployed as an offline process. This issue is further discussed in Section 5 of the paper.

```
choose-CBR-Move(P, O, CB) {
  inputs: P, a new piece presented with orientation O
          CB, the pattern case base used by the CBR cycle.
  local variables: Cols, the height of the columns of the game
          k, the number of nearest neighbors being considered
          Candidates, some similar patterns
          Pattern-Case, one specific local pattern
          New-solution, rotation and translation of the piece P.
   // Retrieval
  Cols ← current configuration of columns on the board
  Candidates ← find-Knn(k, P, O, Cols, CB)

  // Reuse
  Pattern-Case ← select-Best-Pattern(Candidates, P, O, Cols)
  New-solution ← adapt(Pattern-Case, P, O, Cols)
  R ← Reward obtained by applying New-solution to P on Cols

  // Revise
  Pattern-Case.Value ← update-Case-Value(Pattern-Case, R)
}
```

Figure 2 - CBR approach for the reuse of local patterns.

3.1 Representation of Local Patterns

We constructed an initial CBR system by playing multiple games with Tieltris, thereby cumulating over 60 000 moves. Following our decision to adopt a local representation, we converted each case into a local pattern. A pattern, depicting local information on where a figure is dropped, is represented by the following features:

- *Restricted pattern*: A pattern is represented as the heights of a sequence of *N* columns, *N* being the size of the figure being placed on these columns. The heights are all relative to the lowest column in the pattern (see Figure 3).
- *Local depth*: the depth of the restricted pattern, i.e. the difference between the lowest part of the pattern and the highest column adjacent to the pattern.
- *Global height*: the global height of the pattern with respect to the lowest column in the whole surface.
- *Degree of intrusion*: we memorize the number of empty squares on the row corresponding to the lowest level of the local pattern (Figure 4). It is an estimation of the possibility to complete a row if a piece is applied to this position. This is important as filling rows is the only way in Tetris to create additional room on the board to place figures.

This local representation scheme reduces the problem space to less than 70 millions possible states.

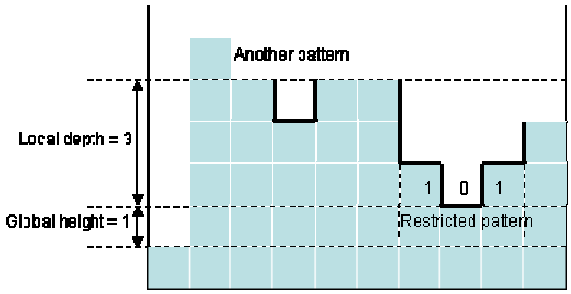

Figure 3 – Pattern features. A case contains information about the height of the columns where the figure is placed (restricted pattern) as well as the position if this pattern relative to the global surface of columns.

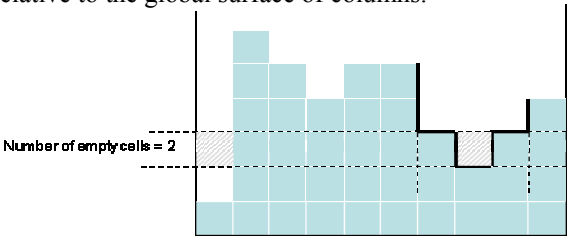

Figure 4 – Degree of intrusion. This corresponds to the number of empty cells at the bottom of the pattern.

3.1 Retrieval of cases

In order to estimate the similarity between a board configuration and a local case pattern, the function *find-knn* traverses the surface of columns and estimates the similarity between the case pattern and each sequence of N columns in the surface (N being the size of the pattern).

Similarity with a restricted pattern is obtained from the Manhattan distance between the cells of the board and the heights of the pattern. For the other features (local depth, global height, degree of intrusion), the distance corresponds to the absolute difference of value. Each distance is then converted into a similarity measure using the inverse function

$$similarity = \frac{1}{\text{distance} + 1} \cdot$$

Global similarity is estimated as the weighted sum of the similarity between these four features. The retrieval function returns the k nearest local patterns applying to the type of piece to be placed on the surface board. In our experiments, k was arbitrarily set to 5 cases.

3.2 Reuse of cases

For selecting a case to be reused (*select-Best-Pattern* function), we tried different linear combinations of similarity and case value. For small values of k, the best results were obtained with case value as the only selection criteria. And we noted that adding similarity as a selection criterion either brings degradation or no significant contribution. So the selection function consists of choosing the most valued case of the nearest neighbors.

Using a local representation, the adaptation of the selected case is rather simple and consists of determining two parts:

- the rotation of the piece to be placed;
- how many columns the piece has to be moved sideways (translation).

Each case solution already contains a rotation to be applied to the figure. So the resulting rotation is a composition of the initial orientation of the piece and the manipulation proposed in the case. For translation, a piece is always moved sideways to the left-most position of a pattern. Hence reuse corresponds to applying the solution of a retrieved case (a rotation) to the piece, followed by a shift to the appropriate position corresponding to the location of the pattern in the global surface.

3.3 Case Revision

When dropping a figure on the board, the system assigns a gain to this move. Revision consists of adjusting the value of a case based on the payoff of new playing episodes. This adjustment can be made from two different perspectives:

- the value of a case is only modified according to the new gain obtained for its application to a new board configuration;
- the value of a case is discounted according to the payoff obtained from the application of subsequent cases. This interpretation captures the sequential nature of the game as the value accounts for the expected payoff of the next moves.

These are discussed in section 4. It is important to note that at this stage, we do not consider adding new patterns to the case base or replace existing ones in this phase as our goal is primarily to study the evaluation of available cases.

4. Evaluating Cases through Reinforcement

The game of Tetris can be considered as a sequential decision making process under uncertainty. Moreover Tetris is a Markovian process as the next configuration of a board only depends on the current surface of columns and the move applied to the dropping figure.

Reinforcement learning (Sutton & Barto, 1998) is a practical approach to learn consistent evaluations from observed rewards for complex Markov decision processes (MDPs). Some authors (Carr, 2005) (Mairal et al., 2006) have applied approximate techniques of RL to Tetris, but not in a CBR setting.

RL can be used to estimate the value of a state through temporal difference techniques or to evaluate action-state pairs through Q-learning. For our application, we adopt a Q-learning approach.

The training procedure goes as follows. We start with some initial evaluations V corresponding to the rewards assigned by Tetris to each of the local patterns of the case base. Then we let the CBR component play games during which cases are selected using the general approach described in section 3. For each selected case C_t at time t, a

revision of its value is performed using the following update function

$$V(C_t) = V(C_t) + \alpha\left(R(C_t, surface_t) + \gamma \max_{C_{t+1}} V(C_{t+1}) - V(C_t) \right) \quad (1)$$

where R is the reward obtained by applying the move adapted from C_t to the new target surface at time t and γ is a discount factor assigning some importance to future moves.

In the update equation (1), C_{t+1} corresponds to the case selected by the CBR system at the iteration $t+1$. This captures the idea that an efficient CBR system should seek the maximum payoff expected from future moves. Hence the value of future moves should be backed up in the value of C_t. As the CBR cycle always chooses the most valued local pattern present in the case base, we assume that the next selected case C_{t+1} is a good approximation of the maximum solution to be applied to $surface_{t+1}$. From an implementation point of view, the value of C_t is updated during the CBR cycle at time $t+1$.

In order to prevent falling into local optima regions, the training process is allowed to explore the search space by selecting non maximal cases from the set of nearest neighbors. This is captured by a softmax rule (Sutton & Barto, 1998) where the probability of selecting one of the nearest neighbors is given by

$$P(case) = \frac{e^{V(case)/\tau}}{\sum_{case_i \in knn} e^{V(case_i)/\tau}} \quad (2)$$

where τ is an exploration factor (or temperature) and V is the value of a local pattern case. This factor is reduced progressively with time to bring the training algorithm to adopt a greedy exploitation behavior (i.e. select the most valued pattern).

To compare Q-learning to a baseline, we also implemented a simplified version of temporal difference, referred to as every-visit Monte Carlo method by Sutton and Barto, to train retrieved cases. This approach can be interpreted as follows for our application:

$$V(C_t) = V(C_t) + \alpha\left(R(C_t, surface_t) - V(C_t) \right) \quad (3)$$

In fact, from an implementation point of view, this is equivalent to applying Q-learning without discounting. But for simplicity, we refer to this update function as TD(0) later in this work.

5. Controlling the Size of the Case Base

Tetris is a time-constrained game as one must take an action before the dropping figure reaches the surface of the board. The problem gets also more complex as the available time to place a figure reduces with the level of the game. In our general CBR approach, most of the reasoning time is dedicated to retrieval, i.e. matching patterns with a surface. As execution time of a CBR cycle is linearly proportional to the size of the case base, one should learn how to manage this parameter to efficiently build a system.

We have made some trials to evaluate the relevance of reinforcement values for deleting cases from the case base. For comparison purposes, we have introduced a usage degree U to the specification of a case indicating the contribution of the case during the training process. The usage degree is an increasing factor defined as

$$U = 1 - \tau^n$$

where n is the number of times the case was invoked during reinforcement training. We selected a value of τ = 0.6 for our experiments. The description of a case becomes:

C = <*Piece, Orientation, Pattern, Solution, V, U*>.

6. Experiments

For conducting our experiments, we adopted the following procedure for each of our trials. We first generated approximately 60 000 cases using Tieltris and 5000 of these cases were picked at random to form a case base. Then a number of games were played either for training purpose (> 500 games) or performance assessment (~100 games). Results presented in the tables are the average values obtained during performances assessment games.

6.1 Performance with a case base of local patterns

Figure 5 presents our initial performance estimation of the general CBR approach described in Section 3. We compared three different CBR configurations:

a. Global: Case similarity is based on the distance between two global surfaces and the results shown here are for a Manhattan distance[1]. The solution of the case with the most similar global surface is applied without any modification.
b. Local with k = 1: Case problems are represented as local patterns. The selected move always corresponds to the most similar case. Hence the value of a case does not intervene in the selection process. The solution is adapted in rotation and translation.
c. Local with k = 5: The same as previous but the selected case is the most valued among 5 nearest neighbors. Case value corresponds to the gain attributed by Tieltris for each move.

	# of pieces played	Game Score	Game level	# of lines removed
a. Global	38.2	375.8	1.0	1.8
b. Local (k=1)	70.8	702.0	1.0	13.7
c. Local (k=5)	97.9	973.6	1.08	23.8

Figure 5 – Global vs. local similarity.

[1] We also estimated similarity from a Euclidian distance but we observed no significant difference in the results.

Our results clearly indicate that the global approach (config. a) performs poorly. As it fails to remove lines from the board due to its incapability to adapt to slightly varying situations, it has little potential for improvement. Using patterns based on similarity (config. b) offers a better capability to target where a figure should be placed. This translates in a significant increase in the number of lines being removed. Finally, using reward values assigned by Tieltris (config. c) further improves the performance of system by providing better discrimination among competing patterns. However some additional experiments (not shown here) indicate that behavior degrades when we increase the number of nearest neighbors considered for case selection.

6.2 Value estimation using reinforcement training

To determine if reinforcement learning can contribute to the evaluation of cases, we performed training on a case base of approximately 5275 cases using a local pattern configuration with k=5 (config. c). We arbitrarily chose the following parameters to update the value of a retrieved case:

- The learning rate $\alpha = 0.6$.
- The exploration factor $\tau = 0.7$.

Learning	# of pieces played	Game score	Game level	# of lines removed
TD(0)	116.5	1159.4	1.42	32.0
$\gamma = 0.1$	104.7	1041.6	1.16	26.2
$\gamma = 0.5$	98.6	979.5	1.06	23.8
$\gamma = 0.9$	98.9	982.7	1.13	24.2

Figure 6 – CBR performance after estimation of case value using reinforcement learning without exploration.

	# of pieces played	Game score	Game level	# of lines removed
TD(0)	147.8	1466.8	1.90	44.1
$\gamma = 0.1$	91.7	911.0	1.08	21.7
$\gamma = 0.5$	100.3	996.5	1.14	24.7
$\gamma = 0.9$	94.7	1055.9	1.18	27.2

Figure 7 – CBR performance after estimation of case value using reinforcement learning with exploration.

Results obtained by playing 100 games with the resulting case base are presented in Figure 6. These were generated without exploration, i.e. by always selecting the most valued pattern. We generated results with different value of γ to evaluate the impact of discounting future moves. From Figure 6, we note that a significant improvement is made when training is performed without discounting. However this progressively vanishes when we assign more importance to the expected payoff from future moves. We repeated this experiment by retraining the system and allowing the exploration of non maximal pattern (Figure 7). As expected, letting the system explore brings further benefits when no discounting is applied. For the TD(0) formulation, we get an increase of 26.5% in game score and 37.5% more lines being removed per game.

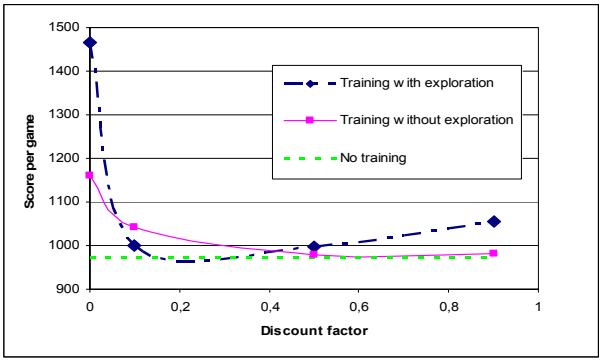

Figure 8 – Score per game with respect to future discount.

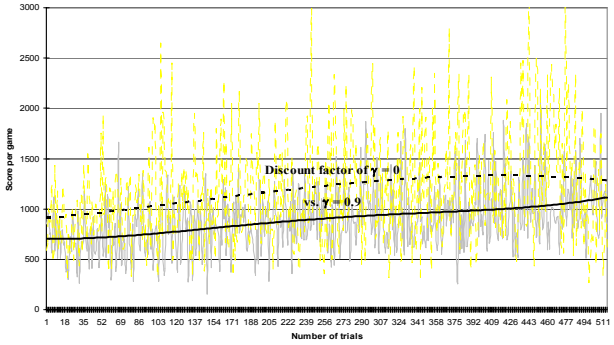

Figure 9 – Tendencies of score during training. The upper line corresponds to a discount factor of γ=0, the lower one to γ=0.9.

However, as we increase γ, Q-learning reveals unstable and it becomes difficult to ensure that the benefits of exploration are always obtained. Figure 8 clearly shows there is no improvement made by using discounting values. As mentioned in (Carr, 2005), this could be attributed to the stochastic nature of Tetris which limits the accuracy of the estimates. By increasing the look ahead window, we might introduce noise in the estimation of the patterns. Another possible explanation, as illustrated by the tendency lines in Figure 9, is that while TD(0) seems to have converged within 500 trials, higher discounting would require longer training to efficiently provide useful values. Finally discounted updates could be applied only when successive pieces are placed beside or above each other as moves being far apart can be considered independent.

6.3 Forgetting Patterns from the Case Base

For this experiment, we try to determine if case values obtained through reinforcement training are a good indicator for reducing the size of the case base. We compared 2 case reduction criteria: usage degree and case

value. For both approaches, we start with a case base of 5000 patterns trained with TD(0) and softmax exploration. We progressively removed cases from the case base in increasing order to evaluate what performance variation can be observed. As for previous experiments, 100 games were played for each trial. Comparative results are presented in Figure 10.

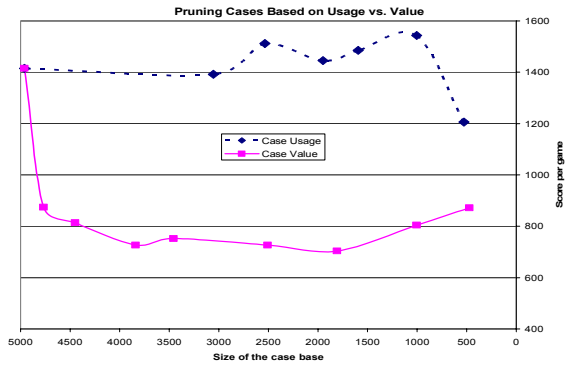

Figure 10 – Reducing case base size. The graphics presents game score obtained by removing cases with increasing reinforcement value or usage degree.

Our results indicate that removing cases based on their reinforcement value prematurely yield an important degradation of performance (see *Case Value* curve). By deleting the least valued 200 cases from the initial case base, the CBR component looses 38% in game score and removes on average 50% fewer lines during a game. We conjecture that these cases are those applied to difficult moves offering little payoff. By removing them from the case base, the CBR system is left with no guidance on how to manage these situations. However if we progressively remove additional cases, performance stagnates and even slightly improves around 1000 cases.

Removal based on the case usage results in a far better behavior (see *Case Usage* curve). Removing the 2000 lesser used cases (40% of the initial case base) has no significant effect on performance (a diminution of 1.7%). And it even increases further as we reached an optimum for a case base size of approximately 1000 cases. This is quite unexpected as we did not anticipate that such a small number of cases would maintain the initial level of play.

Hence, from these results, we conclude than forgetting unused cases is a better tactic than removing bad valued cases. Reinforcement value is not by itself an informative criterion for reducing the size of a case base. And our results also suggest that it is possible to significantly reduce the size of the case base without too much impact on CBR performance.

7. Conclusions and Future work

Our goal was to explore how cases can be valued with reinforcement learning and determine if the learnt values would impact on the performance the CBR component. We made use of the game of Tetris to conduct our study. Our experiments indicate that performance of a CBR component can significantly be improved through reinforcement training of the cases. However discounting for future moves in our Tetris CBR approach does not provide any improvements. We also established that case value is not an efficient criterion for downsizing a case base.

Many aspects remain to be investigated for this work. We are currently comparing tactics for removing patterns from the case base combining criteria such as usage, value and density. Modifying existing patterns or adding new ones should also be considered. Our goal was not to build a fully-optimized version of CBR Tetris but we should devote efforts to improve the performance beyond the level of an intermediate human player. Finally constraining the size of the case base with respect to the game level or the width of the board would provide an opportunity for studying real-time control in CBR.

References

Breukelaar, R., Demaine, E.D., Hohenberger, S., Hoogeboom, H.J., Kosters, W.A., Liben-Nowell, D. (2004). Tetris is Hard, Even to Approximate. *International Journal of Computational Geometry and Applications, vol* 14, pp. 41-68.

Campbell, M., Hoane, A., Hsu, F. (2002). Deep Blue, *Artificial Intelligence*, Vol. 134, pp. 57-83.

Carr, D. (2005). Applying reinforcement learning to Tetris; Technical report, Rhodes University, 15 pages.

Gabel, T., Riedmiller. M. (2005). CBR for State Value Function Approximation in Reinforcement Learning. *Proceedings of ICCBR'05*, Springer, pp. 206-220.

Molineaux, M., Aha, D. (2005). TIELT: A testbed for gaming environments. *Proceedings of AAAI'05,* AAAI Press, pp. 1690-1691.

Mairal, J., Jacob, L. (2006). Apprentissage par renforcement: Intelligence artificielle pour un jeu de Tetris. Technical report, University of Cachan, 22 page.

Sutton, R., Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge,MA.

Schaeffer, J., Bjornsson, Y., Burch, N., Kishimoto, A., Muller, M., Lake, R., Lu, P., Sutphen, S. (2005). Solving Checkers, *Proceedings of IJCAI'05*, pp. 292-297.

Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C., Ram, A. (2007). Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL; *Proceedings of IJCAI'07*, pp. 1041-1046.

Silver, D., Sutton, R., Mueller, M. (2007). Reinforcement Learning of Local Shape in the Game of Go, *Proceedings of IJCAI'07*, pp. 1053-1058.