Bayesian Learning of Generalized Board Positions for Improved Move Prediction in Computer Go

Martin Michalowski and **Mark Boddy** and **Mike Neilsen**
Adventium Labs, 111 3rd Ave S, Suite 100, Minneapolis MN 55401 USA
{firstname.lastname@adventiumlabs.org}

Abstract

Computer Go presents a challenging problem for machine learning agents. With the number of possible board states estimated to be larger than the number of hydrogen atoms in the universe, learning effective policies or board evaluation functions is extremely difficult. In this paper we describe Cortigo, a system that efficiently and autonomously learns useful generalizations for large state-space classification problems such as Go. Cortigo uses a hierarchical generative model loosely related to the human visual cortex to recognize Go board positions well enough to suggest promising next moves. We begin by briefly describing and providing motivation for research in the computer Go domain. We describe Cortigo's ability to learn predictive models based on large subsets of the Go board and demonstrate how using Cortigo's learned models as additive knowledge in a state-of-the-art computer Go player (Fuego) significantly improves its playing strength.

1 Introduction

This paper describes our work on *Cortigo*, a system that uses learning in hierarchical Bayesian networks for move prediction in the game of Go. The extremely large number of possible states on a Go board means that learning a policy based on exactly matching specific board positions is unlikely to be helpful. An *exhaustive* policy is infeasible: the number of possible states is estimated to be greater than the number of hydrogen atoms in the universe (Hsu 2007). A *partial* policy can be much smaller, but empirical evidence demonstrates that even very large databases of Go positions are unlikely to provide exact matches past the first few moves of the game. Cortigo learns *approximate* board positions. Unlike previous work in the area (Wolf 2000; Silver et al. 2007; Dabney and McGovern 2007) that uses explicit game features such as liberty counts, the presence or absence of walls or ladders, Cortigo has no explicit model of board features beyond which points are occupied by stones of what color.

Our hypothesis was that using hierarchical Bayesian networks would permit the system to learn *implicit* features, which would then result in good move prediction for board positions that had not appeared in the training set. This has in fact been the case. Presented with a training set of partial board positions (rectangles of varying sizes, oriented around one of the corners of the board), Cortigo learns a set of distributions which can be shown to provide effective move prediction for both human and computer gameplay.

It is important to note that by *prediction*, we do not mean that there is a single best move that Cortigo will select for a given board position. Even the very best human players will disagree on the "best" move for a given position. However, it turns out that effective move prediction in the sense that the *top few* moves predicted by the system include the move that was taken in the actual game results in a system that can provide effective guidance, in the form of a move-ranking heuristic for a Upper Confidence Intervals for Trees (UCT)-based Go player.

In this paper, we present results demonstrating that adding Cortigo to the set of heuristics already used by the computer Go player Fuego results in a significant improvement in performance. We also show that learning these distributions is a process that converges based on small numbers of samples, though both predictive accuracy and convergence suffer as the number of stones on the board increases.

2 Computer Go

The game of Go has a very simple set of rules, but is nonetheless very challenging for both human and computer players. Go differs from most other games for which computer players have been implemented in that the best human players are still significantly better than the best computer players. In contrast, for games such as chess, backgammon, and poker, computer players are acknowledged to be as strong as or stronger than the best human players.

Go is played on a 19-by-19 board, with players taking turns placing either a black or a white stone on an unoccupied point.[1] Consequently, the number of possible moves is very large for the first few moves of the game. The number of "good" moves in the opening game is presumed to be significantly smaller, but is not known. While it is possible to construct an opening book for Go, there is no guarantee that other good opening moves are not waiting to be discovered. Games can theoretically proceed until nearly the entire the board is covered with stones, but in practice tend to run roughly 100 moves, with a wide variance. Considered as a game-tree search, then, the game of Go is a huge problem:

[1]For full rules see: http://en.wikipedia.org/wiki/Go_(game)

the tree depth is on the order of 100 and the branching factor is on the order of the size of the board for at least the first few moves. The number of states is immense and there are no obvious means to factor or partition them, beyond the obvious rotational and reflective symmetries.

Recent approaches to computer Go have either treated it as a stochastic search problem, using a sampling-based heuristic search algorithm known as Upper Confidence Intervals for Trees (UCT), or as a learning problem, or as some combination of the two (Bouzy and Cazenave 2001). Learning for computer Go has included attempting to learn policies via reinforcement learning (Silver et al. 2007), or learning classifiers for board positions, such that each board comes with a suggested next move (Stern, Herbrich, and Graepel 2006). At this point, the strongest players are all variants on or refinements of UCT. One of the strengths of the UCT approach, and one of the reasons for the wide variety of competitive UCT-based Computer Go players (e.g., Fuego, Crazy Stone, MoGo, Zen, etc.) is that the UCT algorithm presents several opportunities for heuristic approaches and enhancements, including but not limited to the selection of nodes for expansion and biasing the Monte Carlo search used to evaluate those nodes.

3 Cortigo

Cortigo provides a ranking on possible next moves for a given board position. Cortigo does this using *move prediction*: mapping from a given board position to a prediction of the next move that would be made by an expert Go player. The moves are ordered in decreasing likelihood that that move would be taken given the current board position, given the distributions that have been learned in a previously-trained hierarchical Bayes net, described below. In many ways, this is an easy domain for this kind of classification problem. The description of the current board position is simple in structure and small in size, consisting only of the current state of each point on the board. For a board position appearing in any given game, the output of the classifier is unambiguous: the next move taken. Finally, this is a domain where very extensive training and testing data is available. Go has been played for hundreds of years in very close to its current form, and many of those games have been recorded in detail.

If the problem were that straightforward, the easy solution would be to buy a large game database,[2] and match the current board position to those appearing in saved games, resulting in a selection of suggested moves based on a long history of expert play. However, the size of the state space makes *finding* a match for the current position very unlikely, even if we limit ourselves to considering quadrants of the board, looking for matches under all of the obvious rotational and reflective symmetries. One might object that what matters is not the absolute size of the state space, but the number of states that are likely to be visited in the course of competent play. This does reduce the state space, but not enough: fewer than a dozen moves into the game, the odds

[2]For example, see http://bigo.baduk.org/index.html

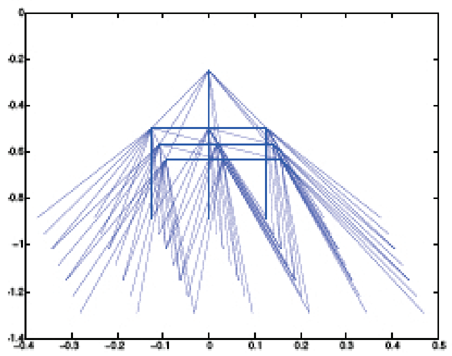


Figure 1: Multilayer pyramidal Bayes net, from (Dean 2006)

of an exact match on board position in even a large database are very low.

Consequently, we treat this as a *generalization* problem: given a limited set of training samples consisting of a quadrant board position and the next move in that quadrant, learn a function such that the move(s) suggested based on a position in a game not previously seen will include the move that was actually taken in that game. This problem is made more difficult by the fact that it is not obvious how to characterize two board positions as being "close" to one another. Removing a single stone from the board, or moving a stone on the board a single point in any direction, can have drastic effects on the strength of the resulting positions for each player.

As mentioned earlier, previous work on computer Go players has sometimes relied on defining specific features thought to be relevant to this classification problem. In contrast, Cortigo starts with no assumptions about specific relevant features above the level of individual points on the board. Cortigo deals with the problem of approximate matches on the board position and multiple possible moves for any given position by learning *distributions*, encoded as a set of weights within a hierarchical Bayesian network. As shown in Figure 1, this network has a very specific structure, known as a Pyramidal Bayes Net (PBN), and is part of a more general approach known as *Hierarchical Generative Models* (Dean 2005; 2006). Intended to be loosely analogous to the visual cortex, these structures have been shown by Dean to provide some of the same benefits in terms of preserving recognition of objects and structures through translation and scaling changes, and in the presence of noisy or missing data. We specifically chose PBNs for their use of discrete-valued (non-binary) nodes and their explicit hierarchy of levels to aid in the required generalization.

In application to recognition (i.e., classification) problems involving visual images, at each but the lowest level the nodes in a PBN take as input the output of the nodes below them in a fixed topology. The "visual field" for nodes at a given level overlap: they share some input nodes in the next level down, thus increasing the degree to which larger features can be captured in correlations between inputs to neighboring nodes at the higher level. Due to the nature of probabilistic inference, nodes corresponding to different parts of the visual field are correlated by virtue of having common children at the next level up, thus biasing the model towards more likely combinations of features.

Mapping from Go board positions to suggested moves fits well with this approach. The "image" is small, with both a limited number of pixels and a very limited pixel depth (3 possible values being white/black stone or empty space). Learning an approximate mapping from board positions to likely next moves is crucial, due to the huge number of possible board positions after the first few moves, and to there being multiple possible next moves for any given position. Finally, there are clearly large-scale features of strategic significance in the game, even if it is difficult to systematically characterize or prioritize them.

4 Experimentation

Ultimately, the relevant metric for evaluating progress in this area is improved performance: if our aim is improved heuristics for UCT computer Go players, how much better does a given player do with Cortigo than without it? This question is addressed in Section 5. In this section, we evaluate *predictive accuracy*: presented with a novel board position, how well does Cortigo rank the move that was actually taken in the game from which that position is drawn? We also address *convergence*: the number of samples required to converge to a reasonably stable set of distributions.

The data used in these experiments comes from games stored in the KGS Go Server game database (Görtz and Shubert 2008). These games involve the top expert skill-level players participating in non-professional games on KGS, in which either one player is at least 7 dan, or both players are at least 6 dan.[3]

The Models

In order to reduce training and evaluation time and to increase the likelihood of finding at least an approximate match to a given board position, all the models we learned and subsequently evaluated covered only subsets of the board. These models came in two forms: 9x9 quadrants around a single corner of the board, and 5x19 fields along each side. These shapes are chosen to cover regions that frequently contain features of interest in Go. The corners and edges of the board are the main focus of play, especially early in the game.

The topology of the PBN has a pronounced effect on how fast the model converges, and on the quality of the resulting performance. All of the results reported here were obtained using 4-level PBNs, with either 81 or 95 nodes in the bottom layer, each with three possible values (white, black, empty). Nodes in the second and third layers have 3x3 visual fields in the next level down. These visual fields have a 1-pixel overlap, such that adjacent nodes at one level share observable nodes at the next level down. Finally, the top-level node has as observable nodes all the nodes in the third layer. The domain sizes for second and third-level nodes are 12 and 7, respectively, and the single top-level node has a domain of either 81 or 95 values, corresponding to the number of possible moves in the region. Choices for topology and domain size are empirical, optimizing over the configurations considered.

[3]*Dan* grades indicate a player's skill in the game of Go

Sample generation

Both training and test sets were generated from the KGS data described above. Each sample consists of a partial board position (the state of the points contained in the 9x9 or 5x19 subset of the board), plus the next move in the game. If that move is in the region in question, the sample is retained. Otherwise (i.e., if the next move in the game was in some other part of the board), the sample is discarded.

Training

Each of the models is trained using *expectation maximization* over the set of training samples, each consisting of a board position (fixing the values of the bottom-layer nodes) and a next move (fixing the value of the top-level node). The overall PBN is subdivided into *subnets* each corresponding to a single node at each level above the lowest and the nodes in that node's "visual field" at the next level down. Expectation maximization is used to estimate the parameters of each subnet. At the first level, training samples correspond to the inputs (board positions) and at all other levels training data is acquired by sampling a posterior belief function over the observable nodes for that subnet. We trained models over varying sizes of training sets, including 30, 700, 5000, and 40000 samples. Training times were prohibitive for significantly more than 40000 samples.

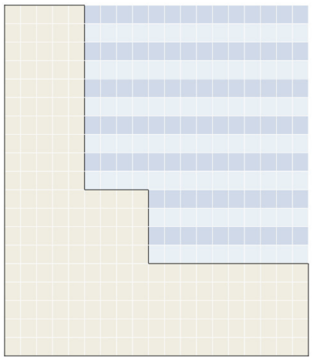
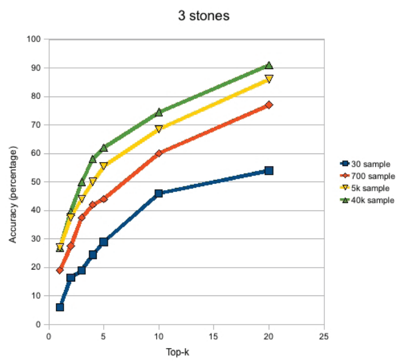


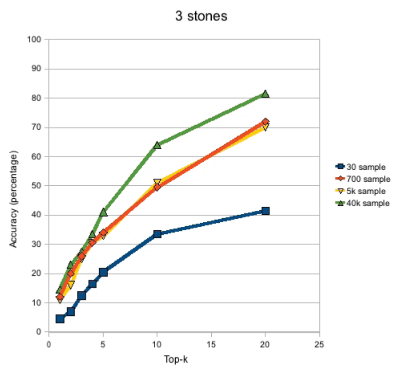
Figure 2: Combined model coverage on a 19x19 board.

Evaluation

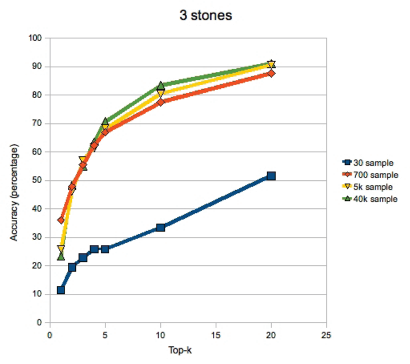
In our experiments, the models exhibited significantly improved predictive accuracy (and a more strongly positive effect on UCT game play) when we trained models for a fixed number of stones within the relevant part of the board. Consequently, we trained individual models for stone counts in the range $\{0 \ldots 20\}$. Those models are then used to evaluate board positions with the corresponding number of stones in that part of the board. We also observed that both predictive accuracy and convergence are better for positions involving fewer stones.



(a) 9x9 model



(b) 5x19 model



(c) Combined model

Figure 3: Top-k results for models trained on 3-stone positions

We evaluated move prediction accuracy for both 9x9 and 5x19 models. Specifically, we looked at whether the specific move taken next in the game from which the given evaluation sample was drawn appears in the top k moves, ranked according to a posterior distribution over the possible values of the top-level node in the model, given the current board position as observed values for the nodes in the lowest level.

We also evaluated the predictive accuracy of *combined* models, making use of both 9x9 and 5x19 trained models. The intuition behind this experiment is that each type of model has an extent that will encompass different types of board features. In the case of combined models, the process by which test samples are generated is modified slightly. For each quadrant, we combine results from the corresponding 9x9 quadrant, and each of the two 5x19 strips including that corner. The total area covered is the shaded region of a 19x19 Go board shown in Figure 2.

The results for training and evaluation on board positions including 3, 5 and 10 stones are shown in Figures 3, 4, and 5, respectively. Each figure plots the predictive accuracy as a percentage (the y-axis) as it changes over the range $\{1 \ldots 20\}$ of k (the x-axis) for our four different training sample sizes. The individual models show good predictive accuracy and convergence for small numbers of stones (so, early in the game). Combining the two models results in significant improvements in both predictive accuracy and convergence. As we discuss in Section 6, these results exhibit a higher predictive accuracy using a fraction of the training examples when compared to existing Bayesian-based approaches.

As this is a generalization problem addressed using a hierarchical model, one possibility to investigate is the presence of "grandma" nodes: specific nodes that have been trained to recognize particular features. In this evaluation, we found no obvious correspondence between intermediate node values and specific configurations of stones, but our investigation was not extensive enough to be conclusive in this regard.

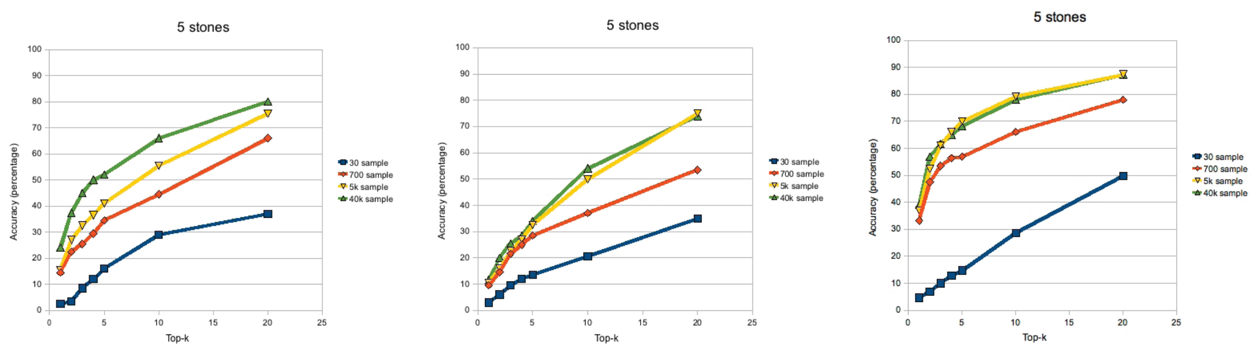
5 Integration With a Computer Go Player

Monte Carlo approaches for Go evaluate positions by playing many games with randomized moves and taking the average result. Although each game is of relatively low quality by itself, the statistics over many random games show a higher evaluation result for better positions. The UCT algorithm adds focused search to this, using partial statistics at repeatedly visited nodes in the tree to select the most promising candidates for subsequent playouts (Kocsis and Szepesvári 2006a). In addition, the top Monte Carlo/UCT Go programs include small amounts of external knowledge to focus search on the highest-priority candidates and to bias Monte Carlo random move selections towards better moves (Gelly and Silver 2007; Coulom 2007).

Fuego (Enzenberger and Müller 2009) is a Monte Carlo/UCT Go program that was developed to provide an effective platform for investigating the integration of machine learning methods. Having a strong program ensures that methods being tested are being evaluated by their effect on playing strength that is near state-of-the-art. Currently, at tournament settings and running on a standard workstation, Fuego (v. 0.4.1) defeats the classical Go program GnuGo (v. 3.6 level 10) over 63% of the time on the 19x19 board.

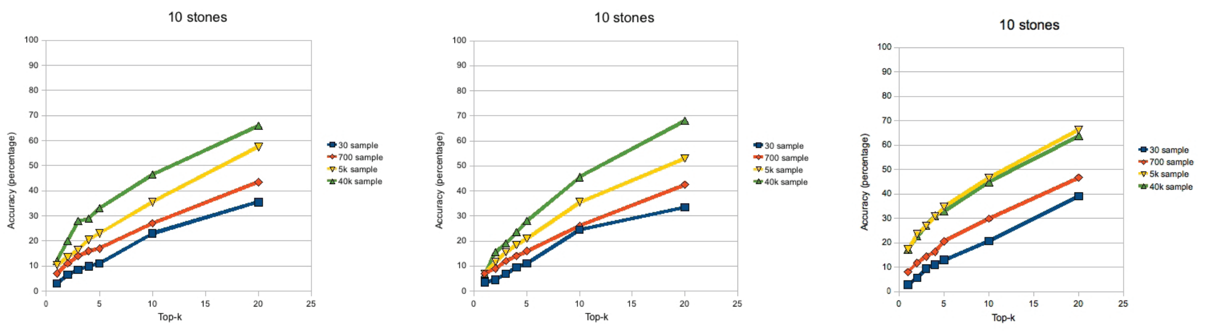
As discussed above, the type of model being learned by Cortigo is well-suited to incorporation into a UCT Go player such as Fuego. Providing a ranked set of move choices fits perfectly with the UCT algorithm's stochastic choice of nodes to expand next. Cortigo's predictive knowledge is incorporated as an additive modification to the UCT formula (Kocsis and Szepesvari 2006b) used by Fuego, the details of which are described in (Rosin 2010). Due to Cortigo's normalized probability distribution it can be used as an additive term to boost and inhibit moves on the board. As Cortigo outputs a ranked list of moves for a single quadrant, we apply the model in all four quadrants for a single board position.

To test the impact Cortigo has on Fuego's performance, Fuego played GnuGo (v. 3.6 level 10) both with and without Cortigo's additive knowledge for 500 games. Cortigo's current implementation is comparatively slow, introducing roughly a 3x slowdown in Fuego. To ensure a fair evaluation, we ran Fuego with and without Cortigo both using 10,000 playouts (starting from the root node), with no time



(a) 9x9 model (b) 5x19 model (c) Combined model

Figure 4: Top-k results for models trained on 5-stone positions



(a) 9x9 model (b) 5x19 model (c) Combined model

Figure 5: Top-k results for models trained on 10-stone positions

limits. This setup ensures each player searched the same number of lines of play. For the version of Fuego using Cortigo, we provided 9x9 models trained on stone counts $S = \{0 \ldots 20\}$ and PBNs as described in Section 4. Cortigo's influence was thus only in the first 20 moves in any given quadrant. Note as well that this means that Fuego was not using the models for which we report the best results, which are the combined models.

	Fuego + Cortigo	Fuego w\o Cortigo
Games won	350	319
Games played	500	500
Winning Percentage	**70.0**	63.8

Table 1: Fuego winning percentage with and without Cortigo additive knowledge.

Table 1 shows that even with influence only in the opening game, Cortigo improves the playing strength of Fuego by a statistically significant 6.2%. As future work we will explore the influence our combined models have on Fuego's playing strength. Integrating the combined models into a UCT-based player is non-trivial as their coverage is non-uniform and the returned move probabilities interact with the UCT formula in ways requiring further study.

6 Related Work

Applying Bayesian methods, including hierarchical generative models, to feature recognition and image classification is an active area of research. A wide range of approaches have been employed for feature detection and extraction, image classification, and either incorporating or inferring annotations for both features and images. All of these are potentially of use in Go, or in other applications involving planning over a two-dimensional field, such as maneuver planning for military operations (Li, Socher, and Fei-Fei 2009; Du et al. 2009). Research in this area has made significant progress over the past few years, resulting in new capabilities that we may wish to incorporate into Cortigo. The work on either inferring or employing annotations seems to be particularly promising.

Work by David Stern and colleagues (Stern, Herbrich, and Graepel 2006) also applies Bayesian inference to the problem of move prediction for computer Go players. This approach employs a two-phase learning process. First, local patterns are extracted from expert games. Each pattern is an exact arrangement of stones on the board, centered around an empty location where a move could be made. Each pattern is then augmented with 8 binary features selected by the authors and Bayesian models are learned and used to select the largest matched pattern whose empty location represents the next move. The authors use roughly 45,000,000 moves for training and learn roughly 12,000,000 patterns, presenting a top-k accuracy of 86% for $k = 20$. Our Cortigo models achieve better predictive accuracy for moves early in the game, while training on a far smaller number of samples. Additionally our proposed method has the significant advantage that it operates simply on the raw board configuration of a Go position, with no need for complex feature engineering. However, it is possible that the more complex features in Stern's work will provide better performance with more stones on the board than Cortigo is capable of. Finally, the work of Tidwell et al. (Tidwell, Hellman, and McGovern 2011) also considers the problem of expert move prediction, in their case using spatial probability models called Spatial Probability Trees (SPTs). We consider this work complementary to ours, as it provides additional additive knowledge that can be used by the UCT-based Go player.

7 Discussion and Future Work

The work described in this paper clearly indicates a promising avenue for further research. There are several directions in which this work could be extended.

Using the current training code implemented in Matlab, training runs of 40,000 samples required up to several days to complete. Speeding up training times would allow us to use larger training sets, which the shape of the graphs presented in this paper suggest would have a positive effect on predictive accuracy, though the size of that effect is difficult to estimate.

The drop-off in predictive accuracy with more stones on the board also raises a more fundamental question: is there a point at which this approach will simply cease to be useful, as the number of stones on the board increases? Intuitively, the answer would seem to be yes. The more stones are on the board, the more likely they are to be connected (where "connected" on a Go board has a complex meaning, depending on context). This makes it more likely that details of the board position significantly separated from one another and from any potential move should have a large effect on move ranking. It is possible that trying alternative topologies in the PBN would mitigate this effect, for example by facilitating the learning of more complex features of the board position.

Finally, modern UCT Go players incorporate multiple heuristics at different points in the algorithm, all tunable by adjusting different parameters. It would be very interesting to see what would result from a systematic exploration of this space, specifically including the use of Cortigo's move rankings. What we have demonstrated here is the improvement in play resulting from adding Cortigo's models while changing no other parameters affecting Fuego's search. That leaves a very large space of alternative configurations, some of which might result in still better performance.

8 Acknowledgements

This work was supported in part by the DARPA GALE project, Contract No. HR0011-08-C-0110. The authors would also like to thank three anonymous reviewers for a thorough reading and comments both extensive and helpful.

References

Bouzy, B., and Cazenave, T. 2001. Computer go: an ai oriented survey. *Artificial Intelligence* 132:39–103.

Coulom, R. 2007. Computing Elo ratings of move patterns in the game of Go. In *Computer Games Workshop 2007*. MICC Technical Report Series, number 07-06.

Dabney, W., and McGovern, A. 2007. Utile distinctions for relational reinforcement learning. In *IJCAI'07*, 738–743.

Dean, T. 2005. A computational model of the cerebral cortex. In *Proceedings of AAAI 2005*, 938–943.

Dean, T. 2006. Scalable inference in hierarchical generative models. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*.

Du, L.; Ren, L.; Dunson, D. B.; and Carin, L. 2009. A bayesian model for simultaneous image clustering, annotation, and object segmentation. In *Neural and Information Processing Systems (NIPS)*.

Enzenberger, M., and Müller, M. 2009. Fuego - an open-source framework for board games and go engine based on monte-carlo tree search. Technical report, Dept. of Computing Science, University of Alberta.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *ICML'07*.

Görtz, U., and Shubert, W. 2008. Game records in SGF format. http://www.u-go.net/gamerecords.

Hsu, F.-H. 2007. Cracking go. *IEEE Spectrum* 10:50–55.

Kocsis, L., and Szepesvári, C. 2006a. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, 282–293.

Kocsis, L., and Szepesvari, C. 2006b. Bandit based monte-carlo planning. In *ECML-06*, 282–293.

Li, L.-J.; Socher, R.; and Fei-Fei, L. 2009. Towards total scene understanding:classification, annotation and segmentation in an automatic framework. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*.

Rosin, C. D. 2010. Multi-armed bandits with episode context. In *Proceedings of the Eleventh International Symposium on Artificial Intelligence and Mathematics (ISAIM '10)*.

Silver, D.; Sutton, R.; ; and Müller, M. 2007. Reinforcement learning of local shape in the game of go. In *IJCAI'07*, 1053–1058.

Stern, D.; Herbrich, R.; and Graepel, T. 2006. Bayesian pattern ranking for move prediction in the game of go. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, 873–880.

Tidwell, Z.; Hellman, S.; and McGovern, A. 2011. Expert move prediction for computer go using spatial probability trees. Technical Report OU-CS-2011-100, University of Oklahoma.

Wolf, T. 2000. Forward pruning and other heuristic search techniques in tsume go. *Information Sciences* 1:59–76.