Expert Move Prediction for Computer Go using Spatial Probability Trees

Zack Tidwell, Scott Hellman, Amy McGovern
School of Computer Science
University of Oklahoma
Norman, OK 73019
ztidwell@ou.edu, shellman@ou.edu, amcgovern@ou.edu

Technical report number OU-CS-2011-100

Abstract

We introduce Spatial Probability Trees (SPTs), which provide a method of learning spatial probability distributions using a decision-tree type of structure. SPTs split the data using statistics calculated with respect to a single point in space and produce a local probability distribution centered on this point. By aggregating these regional probabilities across the entire space, SPTs can be used to generate global probability density functions. We apply SPTs to computer Go to predict expert moves, and show that this predictive power can be used to significantly improve the playing strength of state-of-the-art computer Go players that utilize Upper Confidence Bounds for Trees. In addition, we can perfectly predict the best expert move 35% of the time.

1 Introduction

Go is a two-player board game that is typically played on a 19 x 19 board. Players take turns placing stones on the board; one player placing black stones and the other white stones. Go differs from games such as chess in that, other than color, all stones are functionally identical. The goal of Go is to control the most territory at the end of the game.

The rules for Go are simple, but the resulting game is incredibly complex. The 19×19 board means that there are 361 points on the board. Each point on the board can either be empty, have a black stone, or have a white stone. This yields 3^{361} ($\approx 10^{172}$) possible board positions Hsu (2007), of which approximately 1.2% ($\approx 10^{170}$) are legal Müller (2002). The complexity of Go results in a difficult yet interesting problem for learning and search. Given the size of the search space and the difficulty of designing a static evaluation function, conventional game tree search is ill-suited to playing Go. The current best Go players utilize Upper Confidence Bounds for Trees (UCT)Kocsis and Szepesvári (2006). Players such as Fuego and MoGo utilize UCT

Parameter	Description
l	Leaf node size
r	Distinction sampling rate
m	Minimum node size
d	Maximum tree depth

Table 1: SPT Parameters

search to play Go remarkably well, although not at the skill level of top expert human players Enzenberger and Müller (2009); Lee et al. (2009). Given the complexity of the game, much of the research has been dedicated to solving smaller problems. These include evaluating the safety of groups (the problem of life and death), tactical search, and identification of useful patterns of stones Wolf (2000); Silver, Sutton, and Müller (2007); Cazenave and Helmstetter (2005); Dabney and McGovern (2007). Success in these smaller problems can be used to enhance UCT search by providing knowledge that helps to guide the search. Although these approaches are promising, Go remains an open problem.

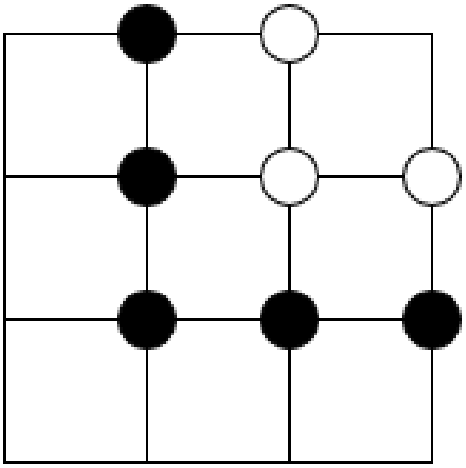
We look at the idea of providing advice to UCT search from the perspective of a spatial prediction task. In Go, we can ask the question, "Where would an expert play given a specific board layout?" The answer to this question can be used to directly guide the UCT search Werf et al. (2003); Stern, Herbrich, and Graepel (2006); Araki et al. (2007); Coulom (2007); Sutskever and Nair (2008); Gelly and Silver (2008). We introduce a novel spatial prediction method, the Spatial Probability Tree (SPT) and we demonstrate that SPTs enable a top computer Go player to significantly improve its playing strength and that they are powerful predictors of expert moves.

2 Spatial Probability Trees

The core functionality of an SPT is to provide the probability of an event occurring at every point in a given space. This is done with a learned, decision tree-like structure. SPTs currently operate within discrete, gridded domains, although they can be easily modified to work in continuous domains. An example of a small Go board represented as a grid for an SPT is shown in Figure 1.

The motivation for SPT comes from observing that, in Go, the shape (patterns) of stones in a small area of the board can be very important in that local region of the board. We developed a model that explicitly works with this idea that local features can provide predictive information not just for the cell where the pattern is centered, but for a small spatial area around the pattern. This local information is not necessarily useful by itself. However, when taken in aggregate over an entire space, meaningful probabilities for an event occurring in that space can be generated.

We first describe the SPT itself and then the learning algorithm. The parameters to the SPT are outlined in Table 1 and described in detail below. An SPT consists of distinction nodes and leaf nodes, all of which are grounded to a single point in space, which we call the focus point. Distinction nodes ask questions about the focus point



(a) Raw Go Board

4	4	4	4	4	4
4	8	1	2	8	4
4	8	1	2	2	4
4	8	1	1	1	4
4	8	8	8	8	4
4	4	4	4	4	4

(b) Gridded Representation

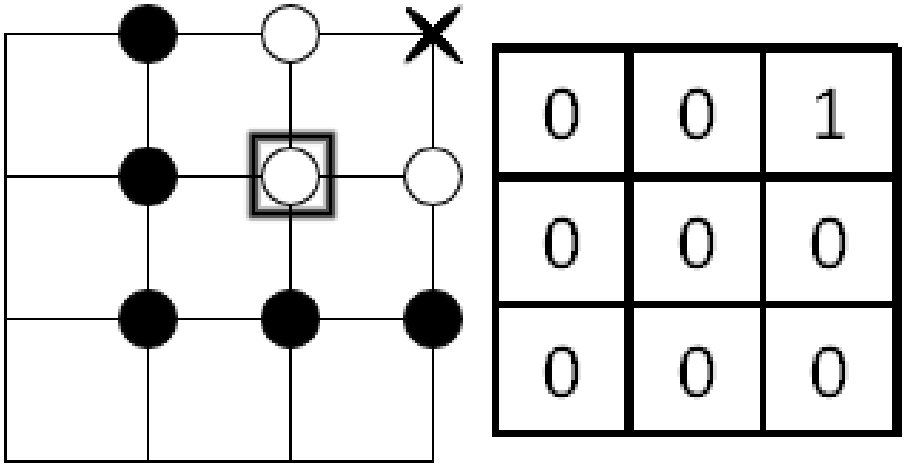
Figure 1: b) shows the gridded representation of a). 1 Represents a black stone, 2 a white stone, 4 an edge, and 8 an empty spot.

and the area surrounding it. Leaf nodes provide the probability of an event happening at the focus point or the surrounding area.

The most basic question that an SPT can ask is, "Is the frequency of a specific value in a window of size n around the focus point $\leq m$?" For example: "Is the number of black stones in a 3×3 window around the focus point less than 3?" "Is the number of empty spaces in a 5×5 window around the focus point less than 7?" "Is the number of edge points in a 3×3 window around the focus point less than 1?" Although these types of questions seem very simple, they are very powerful when chained together in a tree structure.

For our Go experiments, we also provide domain specific distinctions. These include, "How many liberties (vacant places vertically or horizontally adjacent to the stones) will a group (a chain of stones of the same color that are horizontally or vertically adjacent) connected to the focus point have after a stone is placed at the focus point?" "How many stones would be captured by playing at the focus point?" "Was the last move played in a window of size n around the focus point?"

As in a standard binary decision tree, observations that meet the criteria of a distinction are passed down to the subtree on the yes branch, and all other observations are sent down to the subtree on the no branch. At the bottom of each branch is a leaf node. A leaf node is an $l \times l$ probability density function (pdf), where l is the leaf size. The leaf node also contains the probability that the next move is not within the window.



(a) Go Board (b) Label

Figure 2: a) An observation on the Go board, with the focus set to the boxed white stone and the next expert move at the X. b) The corresponding label. In this case, the label size is 3x3.

2.1 Creating an SPT

In most cases, training data must be preprocessed before it can be used to generate an SPT. We obtain our training data from the KGS Go Server[1]. To process the data, each grid with a labeled event must be transformed into a set of observations. On a 19x19 Go board, there will be one point with the next expert move labeled. Every point on the board is considered to be a separate observation and the label for each observation depends on the leaf pdf size l. Unlike traditional decision trees, the label is an $l \times l$ window of 0's with at most one 1. If the next expert move falls within this window, then the corresponding point in the window is set to 1 and the observation is considered to be a positive observation. Figure 2 shows an example for a 3x3 section of a Go board. Given the size of the board and the fact that there is only one next expert move, the training data is heavily skewed towards negative observations. To counteract this, we undersample the negative observations until the ratio is approximately 50/50.

Once the data has been preprocessed, a new SPT can be constructed as described in Algorithm 1. Our approach follows the standard greedy decision-tree learning strategies. In order to create the tree, we must identify a distinction that splits the data in a useful manner. This process is described in Algorithm 2. Due to the large number of possible distinctions, simply trying all of them is not feasible. Instead, we randomly sample according to the sampling rate r. By evaluating how well each sampled distinction splits the data (explained below), we can find a suitable distinction. This splitting is performed recursively. Tree growth stops if a node has too few training observations (m) or if the maximum depth d has been reached.

The pdf of a leaf node is calculated by summing each cell across all of the ob-

[1] http://www.gokgs.com/

Algorithm 1: GrowSPT

```
Input: D = Processed data, c = current depth, d = maximum depth, m =
       minimum node size, r = distinction sampling rate
Output: An SPT
if c < d then
    tree ← findBestSplit(D,m,r);
    if tree ≠ ∅ then
        ydata ← examples in D that meet split criteria;
        ndata ← examples in D that fail split criteria;
        tree.addYes(growSPT(ydata, c+1, d, m, r));
        tree.addNo(growSPT(ndata, c+1, d, m, r));
        return tree
    end
end
return Leaf node
```

servations' labels, effectively adding all of the labels together and then normalizing. The probability that the next move is not within the window is also computed from the frequency of negative observations at the leaf node.

Given a leaf node, we can calculate the likelihood that it explains an observation. For a single observation, the likelihood that it is explained by a leaf node is read from the node's pdf. For a set of observations, the likelihood is the product of the likelihood of each individual observation. Because this product quickly goes to zero as the number of observations increases, we instead calculate the log likelihood. To evaluate a distinction d at a leaf node, we first calculate the likelihood that the given data is explained by the node, L_{null}, as well as the likelihood that the data is explained by leaf nodes that would be created if the distinction was selected, L_d. We then calculate the log likelihood ratio of these two models, $\ln(L_d) - \ln(L_{null})$. We choose the distinction that has the highest likelihood ratio.

2.2 Evaluating an SPT

Once an SPT has been trained, it can be used to predict an event over an entire gridded spatial area. Each point is evaluated independently and the results are combined. When combining local pdfs into a global pdf, overlapping values from nearby local pdfs are added together. Figure 3 shows an example of this process for a very small SPT. The resulting global distribution is then normalized to create a true pdf. This is described in Algorithm 3

2.3 SPT Forests

Although a single SPT can be a powerful spatial predictor, our past work and much of the recent work in machine learning, demonstrate that an ensemble of models is more powerful than any single model. For this work, we ensemble a set of SPTs into an SPT forest. If there was insufficient training data, we could ensure diversity in

Algorithm 2: findBestSplit

Input: D = Processed data, m = minimum node size, r = distinction sampling rate
Output: Best distinction found given parameters, otherwise $\emptyset$
best $\leftarrow \emptyset$;
for $i = 1 \ldots r$ **do**
  split $\leftarrow$ generate random split;
  value $\leftarrow$ likelihood ratio test value for split;
  rdata $\leftarrow$ examples in D that meet split criteria;
  ldata $\leftarrow$ examples in D that fail split criteria;
  if *value* $\geq$ *best and size of rdata and ldata* $\geq m$ **then**
    best $\leftarrow$ split;
  end
end
return *best*

Algorithm 3: GeneratePDF

Input: B = Gridded spatial domain, T = An SPT
Output: Returns a pdf over B
PDF $\leftarrow$ [B.height][B.width];
for *Every point p in B* **do**
  leafPDF $\leftarrow$ T.dropDownTree(p,B);
  add values of leafPDF to PDF centered at p;
end
normalize PDF;
return *PDF*

the forest by following the bootstrap randomization approach used by Random Forests Breiman (2001). However, given the number of training games available online for computer Go, we chose to train each tree in the forest on a different subset of games. Given the diversity across players, this yielded a sufficiently diverse forest for effective prediction. Another approach we have explored was to train each tree on games from an individual player but it was not more powerful than simply training on different sets of games.

To evaluate an SPT forest, the board is evaluated by each tree in the forest. These pdfs are added together and the result is normalized to provide the forest's pdf.

3 Empirical Results and Analysis

We empirically evaluated SPTs in computer Go using two measures. The first measured the kth-move performance and the second assessed the performance of Fuego[2]

[2]http://fuego.sourceforge.net/

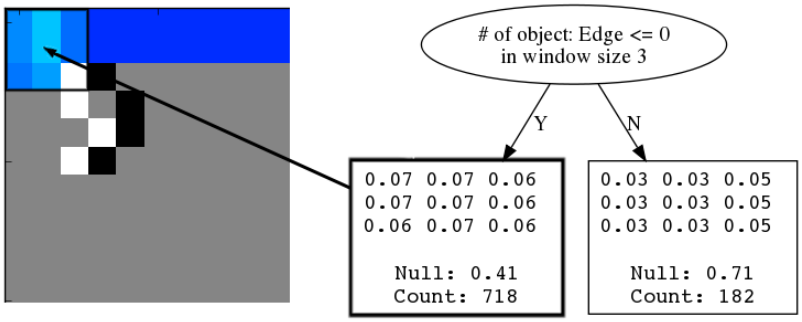


Figure 3: A snapshot of the pdf creation process. The leaf pdf values are added onto the board at the focus point. Lighter blue is higher probability, gray indicates no probability assigned yet.

Enzenberger and Müller (2009), a top computer Go player, both with and without the advice provided to UCT search by the SPTs. We used Fuego version 0.4.1.

Kth-move measures the ability to predict expert moves. An SPT produces a ranked list of potential moves and we identify where in this ranked list of moves the actual expert move falls. For example, if the expert move was the fifth move, then it was within the top 5 moves. From this, we can ask a more general question: "Given multiple test examples, how often is the expert's move within the top k moves?" By varying k, and computing the percentage of the time the expert move is within the top-k moves across all test examples, we can analyze results and compare to published results from state-of-the-art methods that predict expert moves Stern, Herbrich, and Graepel (2006). The motivation behind the kth-move measure is that, in Go, there is not necessarily only one correct move in a given situation. When integrating the advice from SPT into a UCT based player, how often the expert move is correctly labeled as the top move is less important than the fuzzier concept of ranking the expert move within some top number of moves, which enables the UCT player to efficiently reduce the search space. Kth move data for each forest was gathered over 30 full Go games, selected randomly from 671 testing games.

In the Fuego integration experiments, we use a forest of SPTs to give advice to the Fuego UCT player. Fuego uses this advice to bias its UCT rollouts towards better moves. This is done using a theoretically-motivated additive modification to the UCT formula Rosin (2010). To test the performance of the player with the SPT advice, we play 1000 games against GnuGo version 3.6 level 10[3]. Each game uses 10,000 rollouts per move.

Our training data consists of 19×19 games drawn from the KGS Go Server[4] where both players are ranked at least 6 dan and the game lasts at least 70 moves. Dan

[3]http://www.gnu.org/software/gnugo/
[4]http://www.gokgs.com/

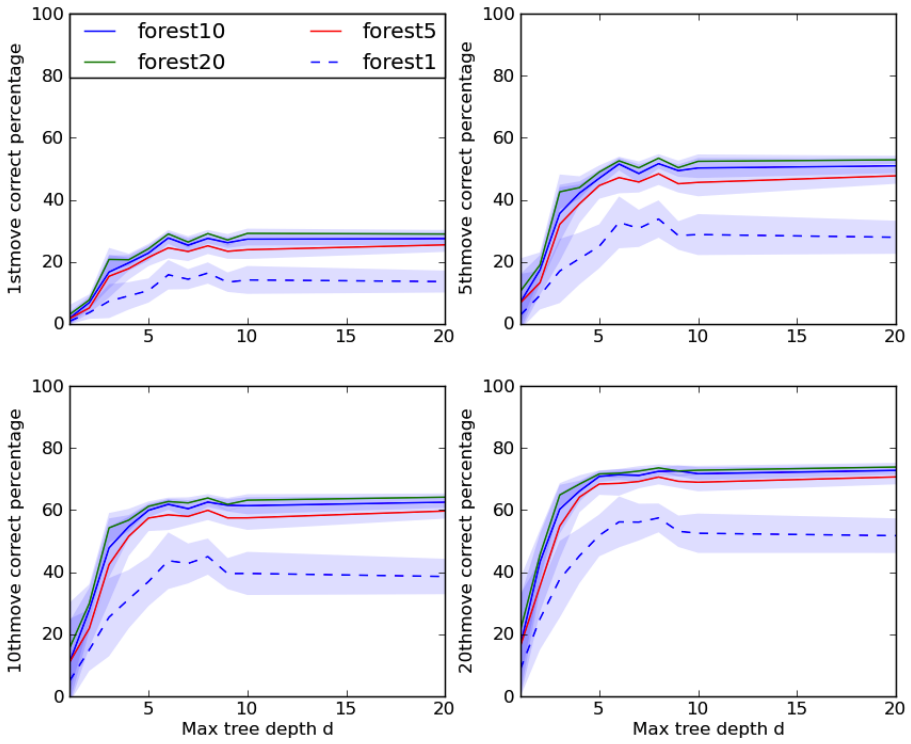


Figure 4: Percent of the time that the expert move is correctly predicted by the top 1, 5, 10 and 20th ranked moves as a function of the maximum tree depth d. Shaded area represents standard deviation.

indicates the level of skill possessed by the Go player. Expert players are ranked into nine dan grades, with players of higher dan possessing greater skill. By focusing on only the top players, we expect the moves to be more consistent. By including only games that last 70 moves or longer, we ensure that our data consists of full games.

3.1 Parameter sensitivity

Our first experiments analyzed the impact of the parameter choices on the performance of the SPT. We measured the SPT's ability to correctly predict the expert move within the first move, the top 5, the top 10, and the top 20 moves. Figures 4, 5, 6, and 7 show the results of this measure as a function of the four main parameters of the SPT and Figure 8 shows the results of varying the number of training observations. All of the experiments were run with the same parameters, except for the one being varied. For the parameter not being varied, the choices were: l = 3, m = 30, d = 10, r = 300, and 300 observations. These observations were drawn randomly from 781 training games.

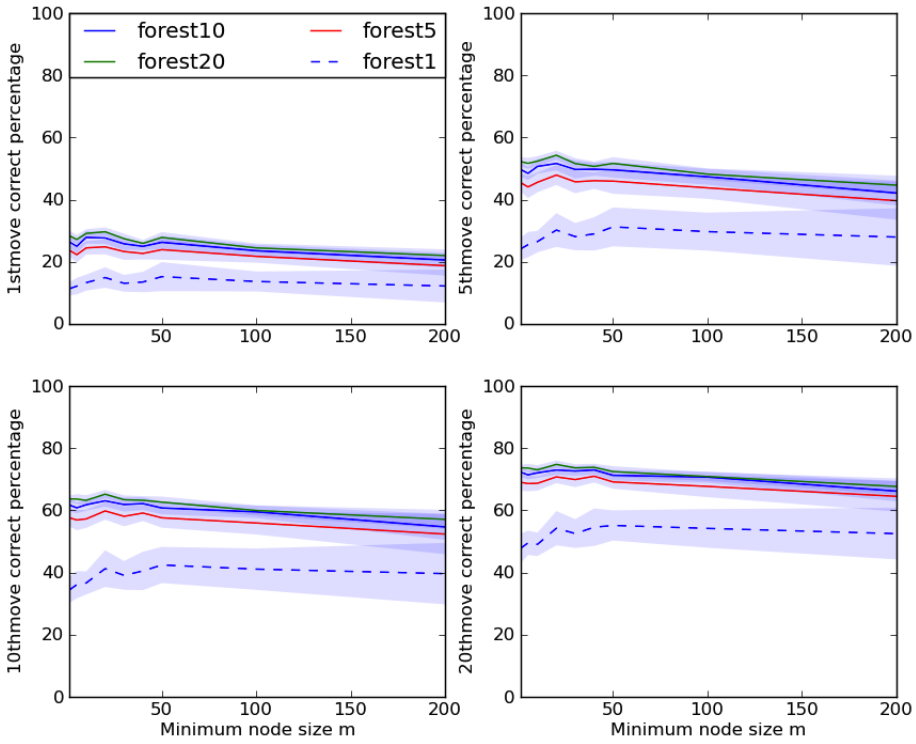


Figure 5: Percent of the time that the expert move is correctly predicted by the top 1, 5, 10 and 20th ranked moves as a function of the minimum number of instances required at a leaf m.

We examined the performance of forests of sizes 1, 5, 10, and 20.

As the tree depth is increased (Figure 4), the SPT is better able to predict the expert moves. The performance quickly levels out around depth 10, most likely in an interaction with the minimum node size m and the number of training observations available. Figure 5 shows the same results as a function of the minimum number of instances required in each leaf node. Here we see a clear decrease in performance as the minimum size is increased because the tree is unable to fully grow. As expected, the sampling rate improves the performance of the tree (Figure 6). Performance asymptotes around 200 samples. The performance as a function of the size of the leaf node (Figure 7) peaks with leaf nodes of 3-5 and then shows a clear decrease as the size of the pdf increases. Finally, the performance as a function of the number of training examples quickly increases but levels off very quickly at around 1000 examples. These results show a clear robustness to parameter selection.

We compare our kth-move results to those provided by Stern, Herbrich, and Graepel (2006). Because we are using different sets of expert Go moves, the results are not directly comparable but they are illuminating. His system was able to correctly predict

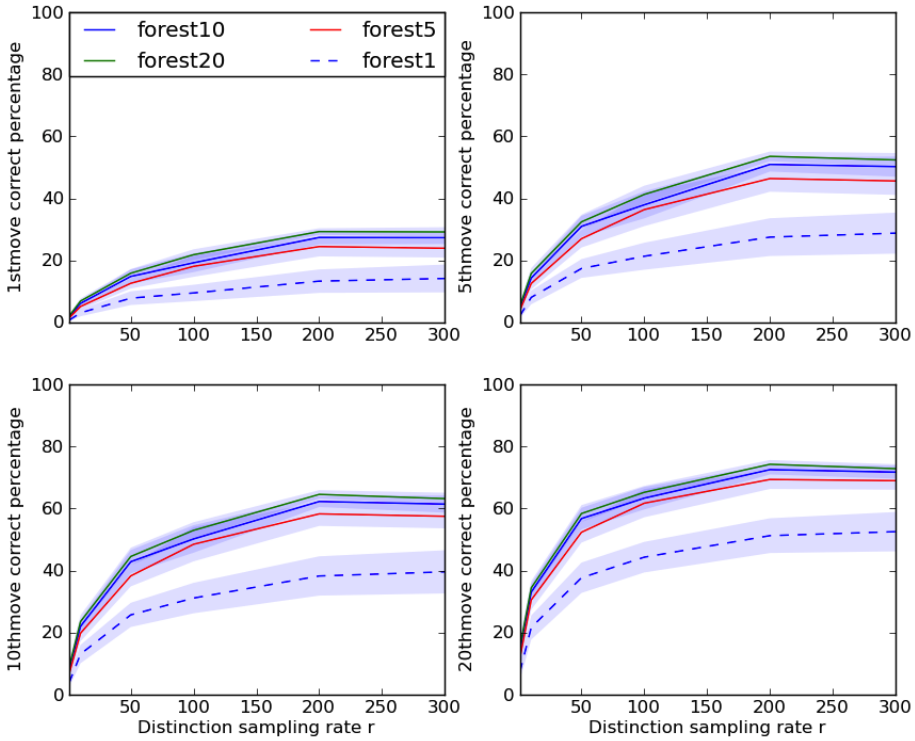


Figure 6: Percent of the time that the expert move is correctly predicted by the top 1, 5, 10 and 20th ranked moves as a function of the sampling rate r.

the expert move 34% of the time, within the top 5 moves 66% of the time, within the top 10 moves 76% of the time, and within the top 20 moves 86% of the time. We are able to achieve improved performance and with a fraction of the necessary training data. In particular, we trained on only 300 board positions while they train on approximately 45,000,000 positions.

3.2 Fuego integration and testing

Given the relative insensitivity to the SPT parameters, we used the same parameter set described above for our comparison against Fuego. We trained 30 SPTs and the forests of varying sizes were created by randomly selecting trees from these 30.

We compared our model to the collection of heuristic Go players provided by the Fuego framework. These models use hard-coded domain knowledge to perform static evaluation of the board. Because the SPT is also providing a fast static evaluation approach, we chose to compare to methods used by Fuego to provide knowledge. For this task, we cannot compare to a search-based player because it cannot provide quick guidance to a UCT player.

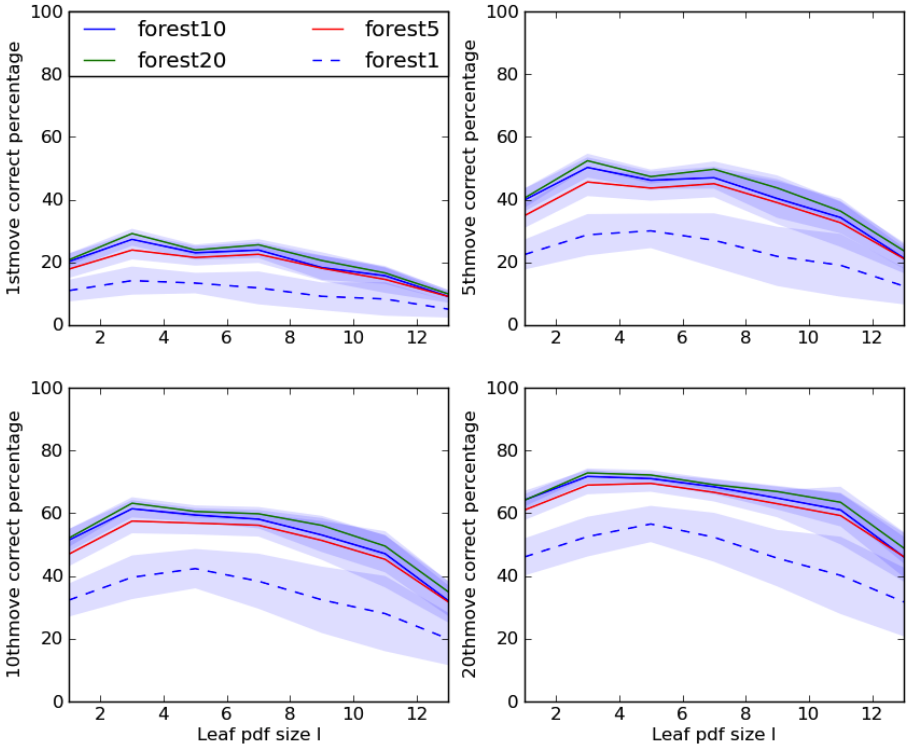


Figure 7: Percent of the time that the expert move is correctly predicted by the top 1, 5, 10 and 20th ranked moves as a function of the size of the leaf l.

Figure 9 shows the kth-move curves for SPT forests of size 1, 5, and 10, as well as random and the best two Fuego heuristic players. From this, we see that even a single SPT outperforms the best heuristic players. Furthermore, we see that a forest of 5 trees performs noticeably better than a single tree. Increasing the forest size beyond 5 does not appear to greatly affect the performance.

The kth-move graphs indicate that the SPTs can predict expert moves well. To assess whether SPT can be used to effectively guide UCT search, we integrated the forests into Fuego and compared the results against GnuGo level 10. Table 10 summarizes the results. Unmodified Fuego won approximately 68.5% of games, Fuego using a single SPT won 73.6% of games, and Fuego using a forest of 5 SPTs won 74.2% of games. These results are all significant at a p-value of 0.01, which means that SPTs were able to significantly improve the performance of Fuego.

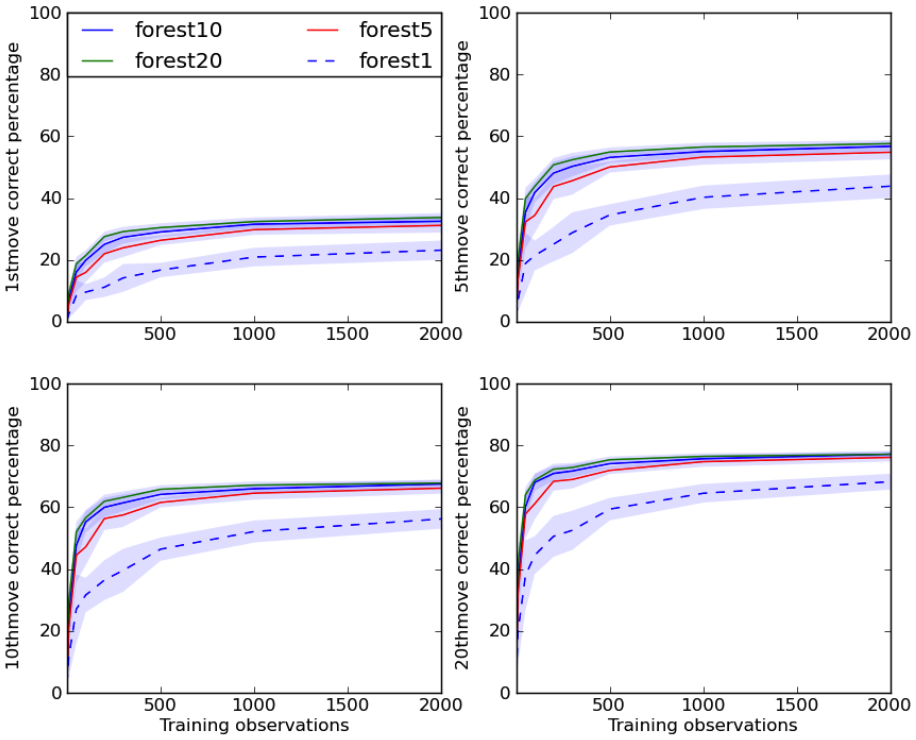


Figure 8: Percent of the time that the expert move is correctly predicted by the top 1, 5, 10 and 20th ranked moves as a function of the number of training observations.

4 Discussion and conclusion

We have introduced a new model for predicting events in spatial domains, the Spatial Probability Tree. We have applied SPT forests to the problem of predicting expert moves in Go. We demonstrated that SPTs achieve improved expert move prediction compared to other state-of-the-art algorithms Stern, Herbrich, and Graepel (2006) and further demonstrated that SPTs enable Fuego, one of the top computer Go players, to significantly improve its performance.

One of the downsides of integrating knowledge into a UCT player is that it slows down the search process. Although the evaluation of the SPTs is very efficient, evaluating hundreds of thousands of moves within a game adds to the complexity of any system. We are investigating how to best add a limited set of knowledge for timed competitions.

The problem of spatial prediction is a common one. Although we applied the SPT only to computer Go, we are investigating the application to several very different domains. As part of this work, we have added distinctions based on landscape metrics

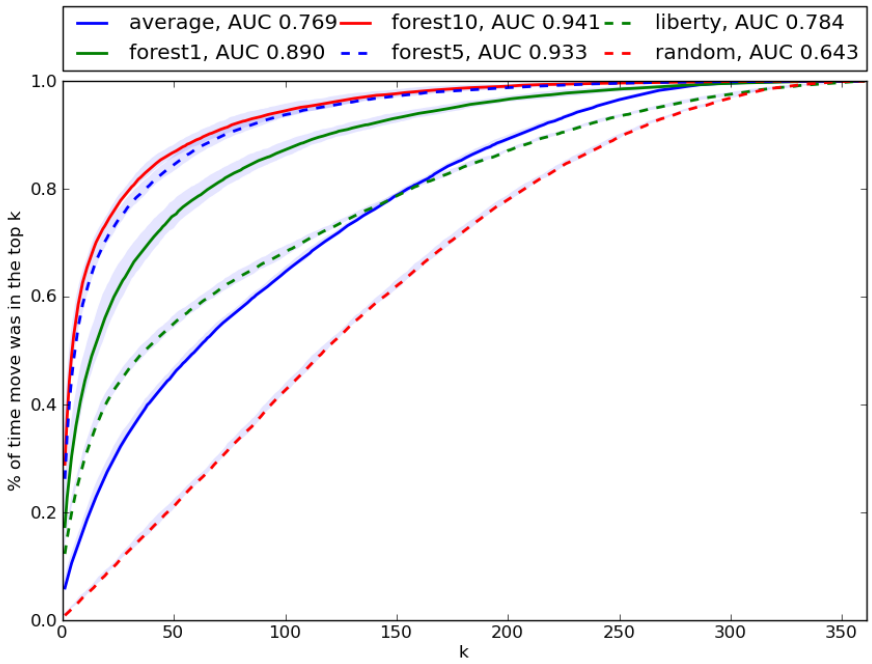


Figure 9: Kth move comparison against Fuego heuristics.

Turner (2005). Although these distinctions did not improve the performance of SPT on computer Go, we believe they will enable SPT to be more widely applicable.

References

Araki, N.; Yoshida, K.; Tsuruoka, Y.; and Tsujii, J. 2007. Move prediction in Go with the maximum entropy method. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, 189–195.

Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

Go Player	Win Rate
Unmodified Fuego	68.5
Fuego w single SPT	73.6
Fuego w 5 SPT forest	74.2

Figure 10: Win rate vs GnuGo version 3.6, level 10

Cazenave, T., and Helmstetter, B. 2005. B.: Combining tactical search and monte-carlo in the game of go. In *In: CIG05*, 171–175.

Coulom, R. 2007. Computing Elo ratings of move patterns in the game of Go. In *Proceedings of the Computer Games Workshop*.

Dabney, W., and McGovern, A. 2007. Utile distinctions for relational reinforcement learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, 738–743. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Enzenberger, M., and Müller, M. 2009. Fuego an open-source framework for board games and Go engine based on Monte-Carlo tree search.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9 x 9 computer Go. In *Proceedings of the 23rd Conference on Artificial Intelligence, Nectar Track*.

Hsu, F.-H. 2007. Cracking Go. *IEEE Spectrum* 44(10):50–55.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 282–293. 10.1007/11871842_29.

Lee, C.-S.; Wang, M.-H.; Chaslot, G.; Hoock, J.-B.; Rimmel, A.; Teytaud, O.; Tsai, S.-R.; Hsu, S.-C.; and Hong, T.-P. 2009. The computational intelligence of MoGo revealed in Taiwan's computer Go tournaments. *Computational Intelligence and AI in Games, IEEE Transactions on* 1(1):73 –89.

Müller, M. 2002. Computer Go. *Artificial Intelligence* 134:145–179.

Rosin, C. 2010. Multi-armed bandits with episode context. In *ISAIM*.

Silver, D.; Sutton, R.; and Müller, M. 2007. Reinforcement learning of local shape in the game of go. In *Proceedings of the 20th international joint conference on Artifical intelligence*, 1053–1058. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Stern, D.; Herbrich, R.; and Graepel, T. 2006. Bayesian pattern ranking for move prediction in the game of Go. In *Proceedings of the 23rd International Conference on Machine Learning*, 873–880.

Sutskever, I., and Nair, V. 2008. Mimicking Go experts with convolutional neural networks. In Kurkov, V.; Neruda, R.; and Koutnk, J., eds., *Artificial Neural Networks - ICANN 2008*, volume 5164 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 101–110.

Turner, M. G. 2005. Landscape ecology: What is the state of the science? *Annual Review of Ecology, Evolution, and Systematics* 36(1):319–344.

Werf, E. V. D.; Uiterwijk, J.; Postma, E.; and Herik, J. V. D. 2003. Local move prediction in Go.

Wolf, T. 2000. Forward pruning and other heuristic search techniques in tsume go. *Information Sciences* 122(1):59 – 76.