On Semeai Detection in Monte-Carlo Go

Tobias Graf, Lars Schaefers, and Marco Platzner

University of Paderborn, Germany,
tobiasg@mail.uni-paderborn.de, {slars,platzner}@uni-paderborn.de

Abstract. A frequently mentioned limitation of Monte-Carlo Tree Search (MCTS) based Go programs is their inability to recognize and adequately handle capturing races, also known as *semeai*, especially when many of them appear simultaneously. This essentially stems from the fact that certain group status evaluations require deep lines of correct tactical play that somewhat oppose to the exploratory nature of MCTS. In this paper we provide a technique for heuristically detecting and analyzing semeai during the search process of a state-of-the-art MCTS implementation. We evaluate the strength of our approach on game positions that are known to be difficult to handle even by the strongest Go programs to date. Our results show a clear identification of semeai and thereby advocate our approach as a promising heuristic for the design of future MCTS simulation policies.

Keywords: MCTS, Criticality, Computer Go, Semeai Detection, Heuristic

1 Introduction

Monte-Carlo Tree Search (MCTS) is a class of simulation-based search algorithms that brought about great success in the past few years regarding the evaluation of stochastic and deterministic two-player games such as the Asian board game Go. MCTS is a simulation based algorithm that learns a value function for game states by consecutive simulation of complete games of self-play using randomized policies to select moves for either player. MCTS may be classified as a sequential best-first search algorithm [12] that uses statistics about former simulation results to guide future simulations along the search space's most promising paths in a best-first manner. A crucial part of this class of search algorithms is the playout policy used for move decisions in game states where statistics are not yet available. The playout policy drives most of the move decisions during each simulation. Moreover, the simulations' final positions are the main source of data for all remaining computations. Accordingly, for the game of Go there exists a large number of publications about the design of such policies, e.g. [7][3][4]. One of the objectives playout designers pursue focuses on simulation balancing to prevent biased evaluations [7][13][8]. Simulation balancing targets at ensuring that the policy generates moves of equal quality for both players in any situation. Hence, adding domain knowledge to the playout policy for attacking also necessitates adding domain knowledge for defending. One

of the greatest early improvements in Monte-Carlo Go were sequence-like playout policies [7] that highly concentrate on local answer moves. They lead to a very selective search. Further concentration on local attack and corresponding defense moves ameliorated the handling of some types of semeai and hereby contributed to additional strength improvement of MCTS programs. However, adding more and more specific domain knowledge with the result of increasingly selective playouts, we open the door for more imbalance. This in turn allows for severe false estimates of position values. Accordingly, the correct evaluation of semeai is still considered to be extremely challenging for MCTS based Go programs [11]. This holds true, especially when they require long sequences of correct play by either player. In order to face this issue, we search for a way to make MCTS searchers become aware of probably biased evaluations due to the existence of semeai or groups with uncertain status. In this paper we present our results about the analysis of score histograms to infer information about the presence of semeai. We heuristically locate the fights on the Go board and estimated their corresponding relevance for winning the game. The developed heuristic is not yet used by the MCTS search. Accordingly, we cannot definitely specify and empirically prove the benefit of the proposed heuristic in terms of playing strength. All experiments are performed with our MCTS Computer Go engine Gomorra. The main contributions of this paper are:

- Analysis of score histograms towards the presence of semeai during an MCTS search process.
- Stochastic mapping of score clusters derived from histograms to board sites of individual semeai.
- Experimental evaluation of our approach on a variety of test positions that are known to be difficult to handle by modern MCTS-based Go programs.

In Section 2 we give a short summary of related work and some background about clustering and mode finding in empirical density functions and the meaning and computation of the Monte-Carlo (MC) criticality measure. Section 3 details our concrete method for the detection of semeai by the analysis of score histograms. The method was used for our experiments which we present in Section 4. Finally, in Section 5 we draw conclusions and give some directions for future work.

2 Background and Related Work

A central role in our proposed method for identifying the presence and concrete location of semeai in game positions, takes into account the clustering of MC simulations into groups related to the score they achieve in their corresponding terminal positions. We decided to use a mean shift algorithm on score histograms for this task. Mean shift is a simple mode-seeking algorithm that was initially presented in [6] and more deeply analyzed in [2]. It is derived from a gradient ascent procedure on an estimated density function that can be generated by Kernel Density Estimation on sampled data. Modes are computed by iteratively shifting the sample points to the weighted mean of their local neighboring points

until convergence. Weighting and locality are hereby controlled by a kernel and a bandwidth parameter. We obtain a clustering by assigning each point in the sample space the nearest mode computed by the mean shift algorithm. We refer to Section 3.2 for explicit formulae and detailed information about our implementation.

The stochastic mapping of clusters to relevant regions on the Go board is realized by the application of a slightly modified MC-criticality measure. Starting in 2009 several researchers came up with the idea of using an intuitive covariance measure between controlling a certain point of a Go board and winning the game. The covariance measure is derived from the observation of MC simulation's terminal positions[5][1][10][1]. Such kind of measures are most often called *(MC-)Criticality* when used in the context of Computer Go. The particular publications around this topic differ slightly in the exact formula used but foremost in the integration with the general MCTS framework. Coulom [5] proposed the use of MC-Criticality as part of the feature space of a move prediction system that in turn is used for playout guidance in sparsely sampled game states, while Pellegrine et al. [10] used the criticality measure with an additional term in the UCT formula. Baudis and Gailly [1] argued for a correlation of MC-Criticality and RAVE and consequently integrated both.

3 MC-criticality Based Semeai Detection

In this section, we present our approach for detecting and localizing capturing races (jap.: semeai) in positions of the game of Go by the clustering of MC simulations according to their respective score and the computation of cluster-wise MC-criticality. When performing an MCTS search on a game position, a number of randomized game continuations called simulations are generated from the position under consideration. Each of these simulations ends in some terminal game position that can be scored according to the game rules. In case of Go, the achievable score[2] per simulation ranges from about -361 to +361. A common first step for obtaining information about the score distribution is the construction of a score histogram that can be interpreted as an empirical density function by appropriate scaling. Assuming that the presence of semeai likely results in more than one cluster of simulation scores (depending on whether the one or the other player wins) we are interested in identifying such clusters and the corresponding regions on the Go board that are responsible for each particular cluster. Accordingly, semeai detection with our approach is limited to cases in which different semeai outcomes lead to distinguishable final game scores.

In the following, we first introduce some notations and afterwards step through our method starting with clustering.

[1] Available online: `http://remi.coulom.free.fr/Criticality/`.

[2] The player that makes the second move in the game is typically awarded some fixed bonus points, called komi, to compensate for the advantage the other player has by making the first move. Typical komi values are 6.5, 7 and 7.5, depending on the board size. Accordingly, the score range might become asymmetric.

3.1 Notations

Let $S \subseteq \mathbb{Z}$ be the discrete set of achievable scores. Having built a histogram H of a total of n simulation outcomes, we write $H(s)$ for the number of simulations that achieved a final score of $s \in S$, hence $\sum_{s \in S} H(s) = n$. We denote the average score of n simulations by $\overline{s} = \sum_{s \in S} H(s)/n$. Each element c of the set of score clusters C is itself an interval of scores, hence $c \subseteq S$. All clusters are disjunct in respect to the score sets they represent. We write $c(s)$ for the single cluster to which a score s is assigned to.

3.2 Clustering

As mentioned in Section 2 we use a mean-shift algorithm for mode seeking and clustering in score histograms that depends on the choice of a kernel and a parameter h that is called bandwidth. For our implementation we use the triweight kernel K defined as

$$K(x) = \frac{35}{32}\left(1 - x^2\right)^3 \mathbb{I}(|x| \leq 1) \ ,$$

where $\mathbb{I}$ denotes the indicator function that equals to 1 if the condition in the argument is true and to 0 otherwise. Fig. 1 shows a plot of the kernel. We use *silverman's rule of thumb* [14] (p. 48) to determine the bandwidth parameter h based on the number of simulations n and their variance:

$$h = 3.15 \cdot \hat{\sigma} n^{-\frac{1}{5}} \quad \text{with } \hat{\sigma}^2 = \frac{\sum_{s \in S}(s - \overline{s})^2 H(s)}{n - 1} \ .$$

When perceiving $f(s) = H(s)/n$ as an empiric density function, we can use Kernel Density Estimation with the triweight kernel and the computed bandwidth parameter to obtain a smooth estimated density function $\hat{f}$:

$$\hat{f}(y) = \frac{1}{nh} \sum_{s \in S} H(s) K\left(\frac{y - s}{h}\right) \ .$$

A plot of the term $K\left(\frac{x-s}{h}\right)$ is shown in Fig. 2. An example for a normalized histogram and the corresponding estimated density function $\hat{f}$ can be found, e.g., in Fig. 3b, where $\hat{f}$ is represented by the black curve.

To find the modes of $\hat{f}$, we initialize a mode variable m_s for each score $s \in S$ to the respective score itself, i.e. $m_s = s$. The mean shift algorithm now iteratively updates this mode variables by

$$m_s = \frac{\sum_{s \in S} H(s) K\left(\frac{m_s - s}{h}\right) s}{\sum_{s \in S} H(s) K\left(\frac{m_s - s}{h}\right)}$$

until convergence. The different conversion points of the mode variables are the positions of the modes of $\hat{f}$. We build one score cluster for each mode and write m_c for the position of the mode that corresponds to cluster c. To account for estimation errors and sample variances we only consider clusters corresponding to mode positions m with $\hat{f}(m) \geq T$ for some appropriate threshold T. Each score s is then assigned to the cluster $c(s) = \text{argmin}_{c \in C} |m_s - m_c|$.

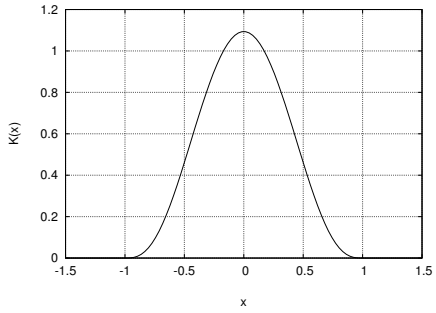


Fig. 1: The Triweight kernel $K(x)$

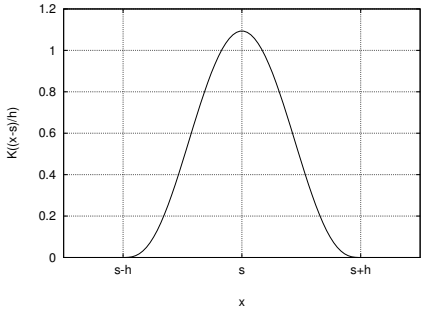


Fig. 2: $K\left(\frac{x-s}{h}\right)$ as used in $\hat{f}(y)$

3.3 Player-, Intersection- and Cluster-wise MC-criticality

MC-criticality was introduced as an intuitive covariance measure between controlling a certain intersection of a Go board and winning the game. Here, controlling an intersection from the viewpoint of a specific player means that in the game's terminal position the intersection is either occupied by a stone of the player, or the intersection is otherwise counted as the players territory (e.g. builds an eye of the player). We propose a slightly modified measure to compute the correlation of controlling a point on the Go board and achieving a score that falls into a given interval. Let $P = \{\text{black}, \text{white}\}$ be the set of players, I the set of all board intersections and C a set of score clusters determined as presented in Section 3.2. Given a number of terminal game positions generated by MCTS simulations, we define the following random variables:

$$X_{p,i} = \begin{cases} 1, & \text{if player } p \text{ controls intersection } i \\ 0, & \text{else} \end{cases}$$

$$X_c = \begin{cases} 1, & \text{if the score falls into cluster } c \\ 0, & \text{else} \end{cases}$$

We define the player-, intersection- and cluster-wise MC-criticality measure $g : P \times I \times C \to (-1, 1)$ by the correlation between the two random variables $X_{p,i}$ and X_c:

$$g(p, i, c) := \text{Corr}(X_{p,i}, X_c) = \frac{\text{Cov}(X_{p,i}, X_c)}{\sqrt{\text{Var}(X_{p,i})}\sqrt{\text{Var}(X_c)}}$$

which gives

$$g(p, i, c) = \frac{\mu_{p,i,c} - \mu_{p,i}\mu_c}{Z} \quad \text{with } Z = \sqrt{\mu_{p,i} - \mu_{p,i}^2}\sqrt{\mu_c - \mu_c^2} \ ,$$

for $\mu_{p,i,c} = E[X_{p,i}X_c]$, $\mu_{p,i} = E[X_{p,i}]$ and $\mu_c = E[X_c]$ with $E[\cdot]$ denoting the expectation. Accordingly, $\boldsymbol{\mu_{p,i,c}}$ denotes the ratio of all n simulations' terminal positions in which player p controlls intersection i and the score falls into cluster

c, $\boldsymbol{\mu_{p,i}}$ represents the ratio of the simulations' terminal positions in which player p controls intersection i regardless of the score and $\boldsymbol{\mu_c}$ is the ratio of simulations with a final score that falls into cluster c.

The measure $g(p, i, c)$ gives the criticality for player p to control intersection i by the end of the game in order to achieve some final score $s \in c$. The lowest possible value that indicates a highly negative correlation is -1. Here, negative correlation means that it is highly unlikely to end up in the desired score cluster if player p finally controls intersection i.

Our measure becomes most similar to the former published intersection-wise criticality measures when choosing the cluster $C_{\text{black}} = \{s \in S | s > 0\}$ representing a black win, and $C_{\text{white}} = \{s \in S | s < 0\}$ representing a white win. This clustering then resembles the criticality by

$$g_{\text{former}}(i) \approx g(\text{black}, i, C_{\text{black}}) + g(\text{white}, i, C_{\text{white}})$$

with the difference that $g(p, i, c)$ uses the correlation instead of the covariance.

3.4 Detecting and Localizing Semeai

Putting the clustering procedure from Section 3.2 and the criticality measure of Section 3.3 together, we obtain a method for analyzing complete board positions with respect to a possible presence of semeai and in case they exist, and that even allows for approximately localizing them on the board.

To analyze a given position, we perform standard MCTS and collect data about the simulations' terminal positions which is necessary to later on derive the score histogram H and the values for $\mu_{p,i,c}$, μ_c and $\mu_{p,i}$. All we need for this purpose is a three dimensional array *control* with a number of $|P| \cdot |I| \cdot |S|$ elements of a sufficiently large integer data type, initialize all elements to zero and increment them appropriately at the end of each MC simulation. Here, for each terminal position the value of element control(p, i, s) is incremented in case player p controls intersection i and the position's score equals s. Given this, we derive the histogram function H by

$$H(s) = \sum_{p \in P} \text{control}(p, i, s) \quad \text{for some fixed } i \in I \ .$$

Having the score histogram of $n = \sum_{s \in S} H(s)$ simulations, we apply the clustering procedure as described in Section 3.2 to obtain the set of score clusters C. As mentioned in Sections 3.1 and 3.2, each cluster $c \in C$ is constructed around a mode of $\hat{f}$ and we denote the corresponding mode's position by m_c. Given this, we can derive the values for $\mu_{p,i,c}$, μ_c and $\mu_{p,i}$:

$$\mu_{p,i,c} = \frac{1}{n} \sum_{s \in c} \text{control}(p, i, s) \ ,$$

$$\mu_c = \frac{1}{n} \sum_{s \in c} H(s) \ ,$$

$$\mu_{p,i} = \frac{1}{n} \sum_{s \in S} \text{control}(p, i, s) \ .$$

This in turn allows for the cluster-wise criticality computation as described in Section 3.3, that, for each player determines the criticality of controlling the corresponding intersection in order to make the game end with a score belonging to the respective cluster. In case more than one cluster was found, the resulting distribution of criticality values for a given player and cluster, typically shows high valued regions that are object to a semeai. Thereby, the criticality values represent a stochastic mapping of each cluster to board regions with critical intersections that have to be controlled by one player in order to achieve a score that corresponds to the cluster. By further comparison of the critical board regions of the varying clusters and under consideration of the clusters mode positions m_c, it might even be possible to estimate the value of a single semeai in terms of scoring points.

In the next section, we present results achieved with our approach on a number of example positions.

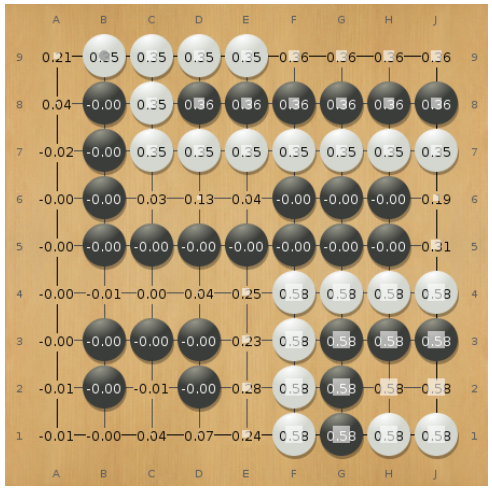
4 Experimental Results

Based on our Go program Gomorra, we implemented our approach and made a series of experiments on different Go positions that contain multiple semeai. For our experiments, we concentrated on a number of two-safe-groups test cases out of a regression test suite created by Shih-Chieh (Aja) Huang (6d) and Martin Müller [9]. The collection of problems in this test suite was especially created to reveal the weaknesses of current MCTS Go engines and is part of a larger regression test suite of the FUEGO Open Source project[3].

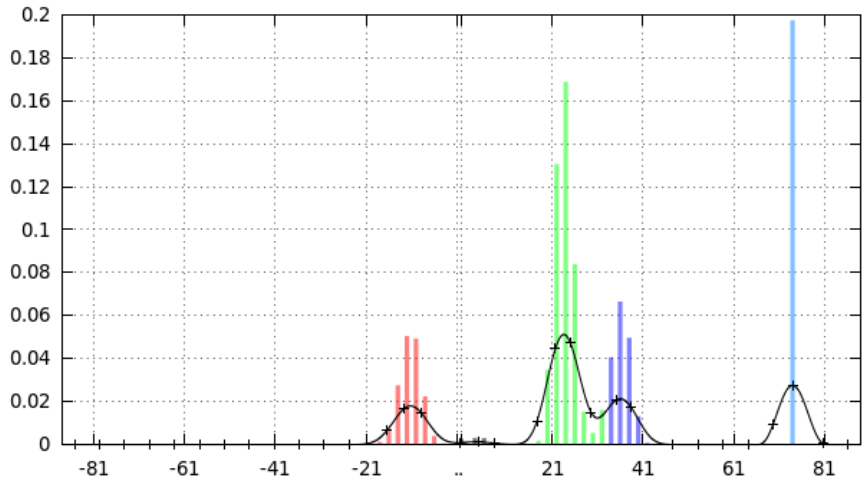
Fig. 3 shows one of the test positions that contains two semeai, one on the upper right, the other on the lower right of the board. Black is to play and the result should be a clear win for white, hence a negative final score, because both semeai can be won by the white player. Fig. 3b shows the corresponding score histogram of 128.000 MC simulations. The colors indicate the clustering computed by the method described in Section 3.2. Figs. 3a, 3c, 3d and 3e show the respective criticality values for the white player to end up in cluster 1, 2, 3 and 4, counted from left to right. Positive correlations are illustrated by white squares of a size corresponding to the degree of correlation. Each intersection is additionally labeled with the criticality value. One can clearly see how the different clusters map to the different possible outcomes of the two semeai.

In the same manner, Figs. 4a to 4d show the results for test cases 1, 2, 3 and 5 of the above mentioned test suite (test case 4 is already shown in Fig. 3). Due to space limitations, we restrict the result presentation to the score histograms and the criticality for player white to achieve a score assigned to the leftmost cluster. The histograms and criticality values always show the correct identification of two separate semeai. We took the leftmost cluster, as for the given test cases this

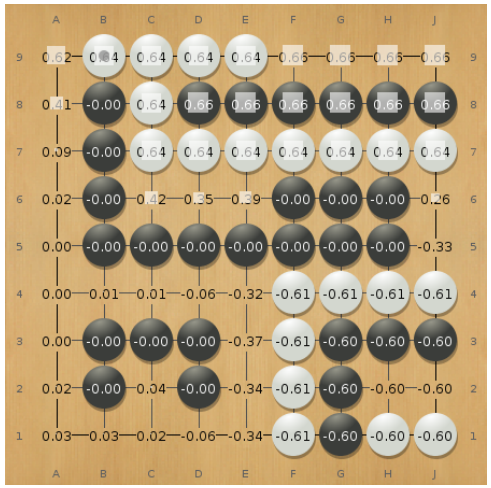
[3] Available online at `http://fuego.svn.sourceforge.net/viewvc/fuego/trunk/regression/`



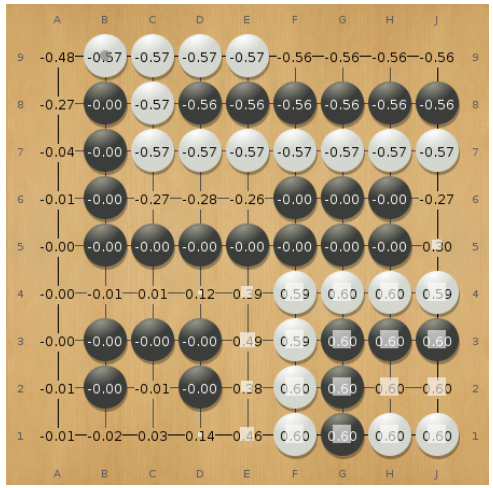
(a) Criticality for white to reach a score in cluster 1 (counted from left to right).



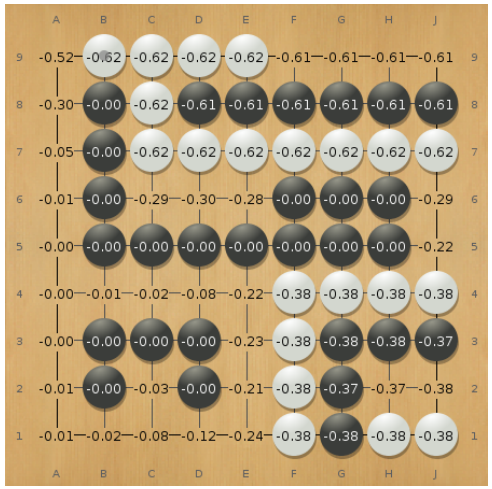
(b) Score histogram after 128,000 simulations.



(c) White's criticality for cluster 2.



(d) White's criticality for cluster 3.



(e) White's criticality for cluster 4.

Fig. 3: The player-wise criticality values reveal the critical intersections on the board for making the game end with a score associated to a specific cluster. The locations of the two capturing races are clearly identified.

is the one containing the correct evaluation, i.e., a win for the white player. As can be derived from the shown histograms, in all cases, Gomorra gets distracted by the other possible semeai outcomes and wrongly estimates the position as a win for black as it is typical for MCTS programs that only work with the mean outcome of simulations.

Fig. 5 shows results for a 13x13 game that was lost by the Go program Pachi playing black against Alex Ketelaars (1d). The game was played at the European Go Congress in Bonn in 2012 and was one of only two games lost by Pachi. As can be seen in the histogram, Alex managed to open up a number of semeai. Again, the board shows the criticality values for the leftmost cluster and reveals Gomorra's difficulties in realizing that the lower left is clearly white's territory. It is one example of a number of games Ingo Althoefer collected on his website[4]. The site presents peak-rich histograms plotted by the Go program Crazy Stone. He came up calling them *Crazy Shadows*. Rémi Coulom, the author of Crazy Stone, kindly generated a *Crazy Analysis* for the 13x13 game discussed above[5].

5 Conclusions and Future Directions

We presented a method to detect and localize capturing races and explained how to integrate the detection into existing MCTS implementations. By doing so in practice, we were able to present a number of examples that demonstrate the power of our approach. However, the detection and localization of semeai alone is only a first step towards improving the semeai handling capabilities of modern MCTS based Go programs. We must develop and evaluate methods to use the gathered knowledge in form of criticality values in the simulation policies to finally turn it into increased playing strength. Specifically, we are highly convinced that remarkable improvements can only be achieved when using the gathered information even in the playout policies.

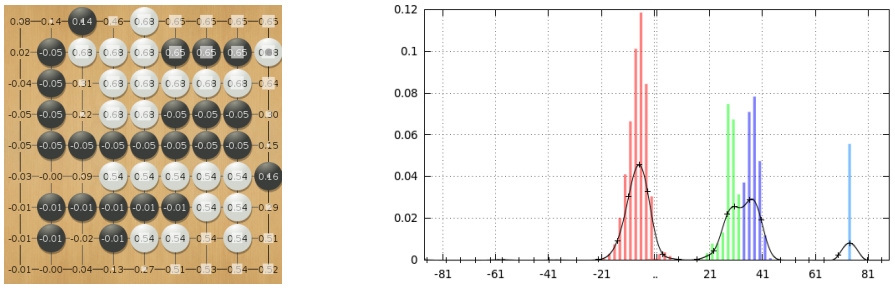
Accordingly, in future work, we plan to investigate the use of some kind of large asymmetric shape patterns that dynamically adapt their size and shape to the critical regions as they are determined by the presented method. Integrating those patterns into existing move prediction systems as they are widely used in Computer Go in addition with training their parameters during the search process builds the next interesting challenge [15]. Already now, the results might be of interest for human Go players using Go programs to analyze game positions[6].

For the sake of correctness, we must admit that the term *semeai* might not be completely appropriately used throughout this paper. A score cluster does not always need to be caused by the presence of a capturing race. In any case, it represents an evaluation singularity that is likely caused by uncertainty in the evaluation of the life and death state of one or more groups of pieces. Also in this case our approach will help to localize the respective groups on the board.

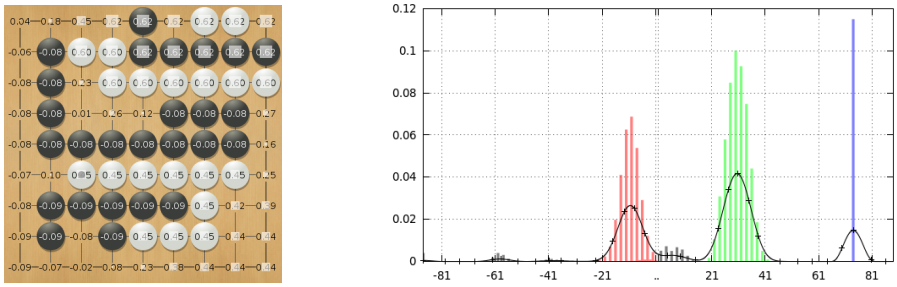
[4] `http://www.althofer.de/crazy-shadows.html`

[5] `http://www.grappa.univ-lille3.fr/~coulom/CrazyStone/pachi13/index.html`

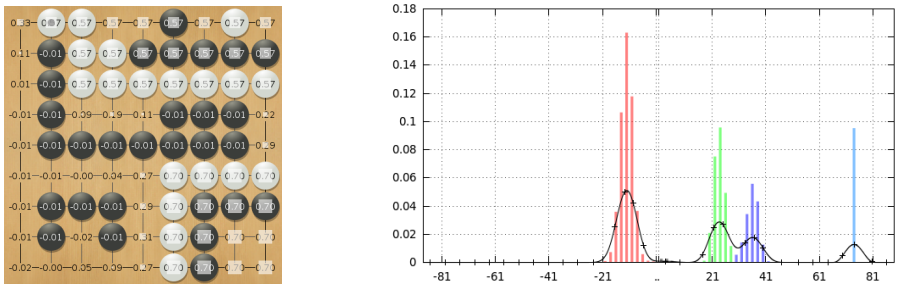
[6] Some more context to existing visualizations for computer aided position analysis can be found online at `http://www.althoefer.de/k-best-visualisations.html`.



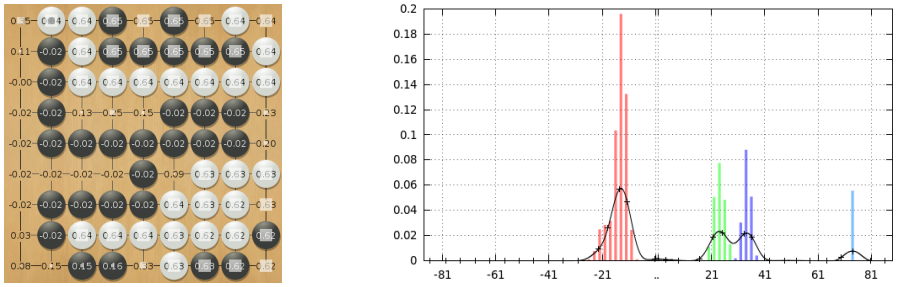
(a) White's criticality and score histogram for two-safe-groups test case 1.



(b) White's criticality and score histogram for two-safe-groups test case 2.

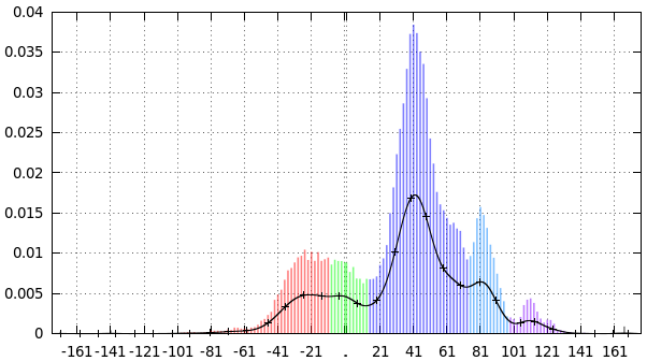


(c) White's criticality and score histogram for two-safe-groups test case 3.

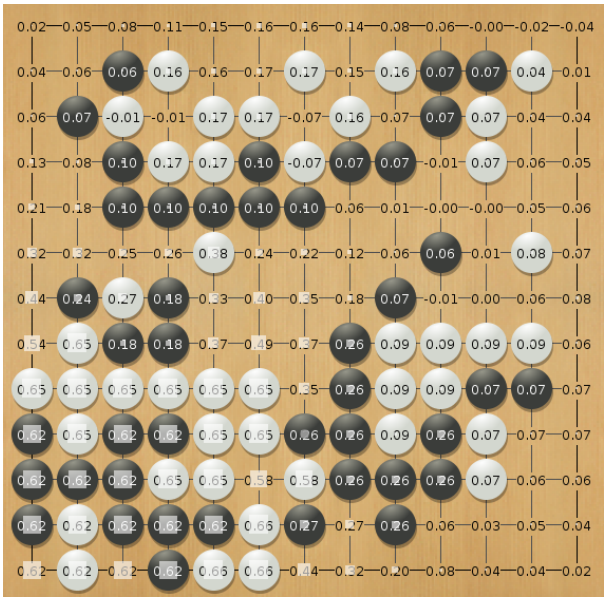


(d) White's criticality and score histogram for two-safe-groups test case 5.

Fig. 4: The player-wise criticality values for another 4 test cases of the two-safe-groups regression test suite. The boards always show the criticality values for player white to end up with a score associated with the leftmost cluster of the corresponding histogram shown to the right. The numbering of test cases is as given in the test suite.



(a) Score histogram after 128,000 simulations.



(b) Criticality for player white to reach a score of the leftmost cluster.

Fig. 5: Analysis of a position occurred in the game between Go program Pachi (black) vs. Alex Ketelaars at the European Go Congress 2012 in Bonn (move 90).

Acknowledgements We like to thank Ingo Althoefer for pointing the community to the potential of score histograms and thereby encouraging our work, as well as for kindly commenting on a preliminary version of this paper. We also thank Rémi Coulom for comments on an early version of this paper as well as for kindly providing an automated analysis by his Go program Crazy Stone of one of the discussed example positions. We further thank Shih-Chieh (Aja) Huang and Martin Müller for the creation and sharing of their regression test suite.

References

1. Petr Baudis and Jean-Loup Gailly. PACHI: State of the Art Open Source Go Program. In H. Jaap van den Herik and Aske Plaat, editors, *Advances in Computer Games 13*, volume 7168 of *LNCS*, pages 24–38. Springer, 2011.
2. Yizong Cheng. Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, August 1995.
3. Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Int. Conf. on Computers and Games*, volume 4630 of *LNCS*, pages 72–83. Springer, 2006.
4. Rémi Coulom. Computing Elo Ratings of Move Patterns in the Game of Go. In *ICGA Journal*, volume 30, pages 198–208, 2007.
5. Rémi Coulom. Criticality: a Monte-Carlo Heuristic for Go Programs. Invited talk at the University of Electro-Communications, Tokyo, Japan, January 2009.
6. Keinosuke Fukunaga and Larry D. Hostetler. The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition . *IEEE Transactions on Information Theory*, 21(1):32–40, January 1975.
7. Sylvain Gelly, Yizao Wang, Remi Munos, and Olivier Teytaud. Modifications of UCT with Patterns in Monte-Carlo Go. Technical Report 6062, INRIA, 2006.
8. Shih-Chieh Huang, Rémi Coulom, and Shun-Shii Lin. Monte-Carlo Simulation Balancing in Practice. In *Int. Conf. on Computers and Games*, pages 81–92, 2010.
9. Shih-Chieh Huang and Martin Müller. Investigating the Limits of Monte Carlo Tree Search Methods in Computer Go. In *International Conference on Computers and Games*, LNCS. Springer, 2013.
10. Seth Pellegrino, Andrew Hubbard, Jason Galbraith, Peter D. Drake, and Yung-Pin Chen. Localizing Search in Monte-Carlo Go Using Statistical Covariance. *ICGA Journal*, 32(3):154–160, 2009.
11. Arpad Rimmel, Olivier Teytaud, Chang-Shing Lee, Shi-Jim Yen, Mei-Hui Wang, and Shang-Rong Tsai. Current Frontiers in Computer Go. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 2, pages 229–238, December 2010.
12. David Silver. *Reinforcement Learning and Simulation-Based Search in Computer Go*. PhD thesis, University of Alberta, 2009.
13. David Silver and Gerald Tesauro. Monte-Carlo Simulation Balancing. In *International Conference on Machine Learning*, pages 945–952, 2009.
14. Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26 of *Monographs on Statistics and Applied Probability*. Chapman & Hall/CRC, April 1986.
15. Martin Wistuba, Lars Schaefers, and Marco Platzner. Comparison of Bayesian Move Prediction Systems for Computer Go. In *Proc. of the IEEE Conf. on Computational Intelligence and Games (CIG)*, pages 91–99, September 2012.