M-eval: A multivariate evaluation function for opening positions in computer *go*

Jacques Basaldúa
Departamento de Estadística IO y Computación, Universidad de La Laguna, 38271 La Laguna, Santa Cruz de Tenerife, Spain
jacques@dybot.com

Tai-Ning Yang
Department of Computer Science, Chinese Culture University, Taipei, Taiwan
tnyang@faculty.pccu.edu.tw

J. Marcos Moreno Vega
Escuela Técnica Superior de Ingeniería Informática Universidad de La Laguna, 38271 La Laguna, Santa Cruz de Tenerife, Spain
jmmoreno@ull.es

Abstract— **Recently, computer *go* has experienced great advance with the introduction of Monte-Carlo Tree Search (MCTS). Although MCTS programs are overall stronger than previous programs, their strength manifests mostly as the game advances. Strong human players applying established opening principles overtake current MCTS programs in the early moves of non-handicap 19x19 games. In this paper, the authors propose a method, M-eval, for implementing these opening principles in a program. M-eval is a multivariate evaluation function that provides a priori information for MCTS-based *go* programs. An initial vector made of *k* positive qualities both players wish to maximize is computed on a board position for each player. The initial vector is dimensionally reduced using non-negative matrix factorization to minimize informational loss. This reduction is done using the multivariate structure learned offline from a set of over 110,000 board positions from a database of master human games. The resulting R^2 encoding is converted by regression to a real value used as *a priori* information in MCTS. Games were played using the MCTS GoKnot/QYZ framework with M-eval enabled against the same program without M-eval. Results show valuable improvement both in territorial gain and number of wins.**

Keywords: Machine learning, Multivariate analysis, Computer go, Monte-Carlo tree search, Positional judgment, Non-negative matrix factorization

I. Background

A. MCTS in computer go

Go is an ancient Chinese board game, also known as *weiqi*. This paper assumes knowledge of the fundamental *go* notions. Otherwise, the reader can find web resources including *go* rules and tutorials listed in [1]. The content of this paper applies to non-handicap games played on a board of 19x19 intersections. The scoring method can be either Chinese or Japanese counting since it makes no difference for opening positions.

Since 2006, computer *go* has experienced great advance with the introduction of Monte-Carlo Tree Search (MCTS). Although the idea of Monte-Carlo evaluation was proposed before [2], it did not produce strong programs until it was implemented on a tree. Implementation of Monte-Carlo evaluation on a tree not only focuses the computational effort towards the most promising moves, but relates these with their most expected answers. The balance between exploration and exploitation in terms of the bound of a confidence interval was developed by Kocsis and Szepesvári [3]. Plain "vanilla" MCTS produces simple and rather strong *go* programs. MCTS combines well with many previous ideas [4], [5], [6], [7], [8] and has also triggered new research areas [9], [10], [11], [12], [13], [14], [15] producing much stronger *go* programs than any previous approach.

B. Positional judgment in go

Evaluating a board position in *go* is extremely difficult even for professional players. All evaluation functions used in computer *go* have limitations and produce erroneous results under certain circumstances.

Most of the strength of MCTS comes from the fact that it does not require an explicit evaluation function. The Monte-Carlo evaluation is the result of a stochastic process, i.e., the complete playout of a random game. Assuming the program understands particular circumstances like seki, the evaluation of the final position is error free. As the end of the game approaches, given enough simulations, the tree reaches final or near final positions. *Yose* played by strong MCTS engines is equivalent to that of *dan* players as long as the outcome of the game depends on finding the correct move sequence. In the middle game, MCTS engines show strength due to full board evaluation and often overcome losing positions making excellent use of *tenuki* and *furikawari*.

Despite their strength, MCTS programs cannot evaluate opening positions. The length of the play-out games (mean = 528, IQR (501, 552) moves) does not allow the tree to reach near endgame positions. Since the whole evaluation is based on self play, MCTS programs are essentially blind to deviation from opening principles as both sides do the same.

Full board opening books in 19x19 *go* have been used by many authors and we have also implemented a book for time saving reasons in tournament games. Only exceptionally more than eight moves (four per side) are played from the book. This represents a very small part of the opening. Partial (*joseki*) pattern matchers have much longer application, but deciding which *joseki* should be applied is a very complicated problem where bad choices can be really bad.

Many books describe *go* opening principles. Cho Chikun [16] describes a method for counting territory, estimate thickness and *moyo*. Also, classical *go* programs have developed positional judgment functions [17], but it is hard to tell which of the resulting moves are going to be "understood" and successfully exploited by MCTS. The strength of a *go* program depends on entire sequences, not just in finding good moves frequently. Combining different algorithms can produce worse results than those of each algorithm by itself.

II. Introduction

A. M-eval rationale

To explain the motivation underlying our algorithm, the following aims must be kept in mind:

Aim 1. — *Go* is a game of balance, e.g., balance between speed and connectivity, between territory and influence, etc. Too much connectivity (known as slow play) gives the opponent the opportunity to consolidate a territorial advantage, too much speed (known as overplay) leaves exploitable weakness behind. Strong play in *go* is about achieving goals in many different directions always considering the achievements of the opponent. There is a natural equivalence between obtaining a gain +g and destroying an equivalent gain -g from the opponent's achievements. Sometimes the former is easier, sometimes the latter. Positional judgment is multivariate in nature.

Aim 2. — The relation between the move and the next moves is fundamental. Sometimes the best reason for a move is not letting the opponent play that same move. To take the interaction with other moves into consideration, positional evaluation requires search. Search finds strategic *go* principles like *miai* naturally. When there is an alternative way to reach the same target, urgency decreases automatically. Without search, such concepts are hard to define and subject to misevaluation.

Aim 3. — Perfect positional judgment (minimax wise) may be useless for a program. The positional judgment evaluation proposed in this paper is designed to interact smoothly with MCTS. Rather than determining what the "best" move is, it aims to guide the search towards directions that are compatible with human *go* opening principles. It is, just like MCTS, of stochastic nature and learns from MCTS.

Aim 4. — Whatever mathematically definable evaluation function implemented in a program will be far from flawless. Even worse, if the function is strictly deterministic, it will misevaluate positions systematically.

Aim 5. — Accepting the limitations of our individual functions, we need more than one way to measure each abstract strategic concept. Because of the excess in degrees of freedom, dimensionality reduction is mandatory. Furthermore, the first step in this reduction has to be based on the knowledge contained in a massive dataset.

Aim 6. — Since, without any loss of generality, functions can be defined as positive for each player, named *qualities* (e.g. enclosed territory, influence, thickness, more liberties in a semeai race, etc.), it would be very hard to justify that a dimensionality reduction technique translates a positive *quality* by multiplying it times a negative coefficient. That could reduce the squared residuals or some other metric, but it does not make sense as an evaluation function. In short, we have to preserve the non-negative nature of our *qualities*.

B. Non-negative matrix factorization

Originally introduced by Paatero et al. [18], [19] NMF has extended its use to many fields in data analysis, including image analysis. The property of NMF to decompose objects into parts was described by Lee and Seung [20] and further studied by Donoho et al. [21].

Given n board positions for which p qualities have been measured for each player, these non-negative values can be written as a matrix $X_{n\times p}$ which can be decomposed as:

$$X_{n\times p} = W_{n\times d} \cdot H_{d\times p} + U_{n\times p} \qquad (1)$$

Subject to: $W_{n\times d} \geq 0,\ H_{d\times p} \geq 0$

When the residue $U_{n\times p}$ is minimized, $V_{n\times p} = W_{n\times d} \cdot H_{d\times p}$ is an approximation of $X_{n\times p}$ called the NMF of $X_{n\times p}$ and $d < p$ is the dimension to which we have reduced the dataset $X_{n\times p}$.

$W_{n\times d}$ is a matrix whose rows (named encodings) are compressed images of the rows of $X_{n\times p}$.

Since our intention is using $V_{n\times p}$ to represent the entire dataset, we want to minimize the informational loss resulting of using $V_{n\times p}$ instead of $X_{n\times p}$. That loss is proportional (it would be equal if $\Sigma_{i,j}\ x_{ij} = \Sigma_{i,j}\ v_{ij} = 1$) to the Kullback-Leibler [22] divergence $D_{KL}(X \mid V)$, i.e., the extra message-length per datum that must be communicated if a code that is optimal for a wrong distribution V is used instead of a code based on the true distribution X.

We computed the NMF of our dataset using the Lee and Seung algorithm [23] version that uses $D_{KL}(X \mid V)$ as the cost function. Furthermore, $H^{+}_{p\times d}$, a pseudo-inverse of $H_{d\times p}$, was computed for the online conversion of individual points from the initial R^{p} space to its encoding in R^{d}, using:

$$W_{1\times d} = X_{1\times p} \cdot H^{+}_{p\times d} \qquad (2)$$

Finally, the H^{+}_{pxd} matrix is scaled to standardize the encoding over the dataset of 110,542 cases described below. I.e., when H^{+}_{pxd} is multiplied times the entire dataset measured for the differences between both players, it returns a dataset of d variables and 110,542 cases, each variable having mean ≈ 0 and SD = 1.

III. M-eval Description

We have experimented implementing M-eval in two different versions. In all cases we use similar solutions to approach the aims described above. The two implementations are described below under B and C.

A. M-eval main body

Following aims 1, 5 and 6, board positions are evaluated by computing a set of *p qualities* $X_{1\times p}$ for each player. Using NMF, that vector is dimensionally reduced to d components by equation (2). Then, $W_{1\times d}$ is reduced to a real value by:

$$v = W_{1\times d} \cdot L'_{d\times 1} \qquad (3)$$

Where $L'_{d\times 1}$ is a direction learned online by a version dependent greedy algorithm detailed below.

The value of the position is the difference between that values measured for both players.

Following aims 2 and 4, the M-eval value used for each candidate move m_i is the evaluation resulting of a short search starting from that position, rather than the evaluation of just the position after playing m_i. The intention of this search is double: blurring deterministic evaluation errors by inserting a brief stochastic component (a random sample of positions following the root position) and trying to find out the importance of moves including considerations such as *miai*. The search is interrupted after a small number of moves (4 and 8 moves have been tried) and the final node is evaluated as explained. Deeper search depths are expected to be less dependent of the evaluated position and, therefore, less interesting. Too shallow depths may have insufficient variation, since the number of candidate moves explored is not very high. Note that, otherwise, the search algorithm is deterministic and, in absence of that stochastic component, the evaluation would repeat the same path each time. We have not yet done any experimental setup to determine the optimal value for this depth.

Following aim 3, the result of M-eval is not used directly to choose the move played, but integrated inside an MCTS search. The specific details are version dependent and detailed below.

M-eval is disconnected after the opening. Since it is integrated as a source of a priori information for an MCTS program, the precise moment of disconnection should not be critical. In version 1 M-eval was applied during the first 64 moves and in version 2 it is applied until the partial *joseki* pattern matcher does not find recognizable shapes, which is typically somewhat earlier, but it depends on the more or less orthodox style of play.

B. Version 1 M-eval

1) Initial set of variables

Terms used by human players, such as: strength, potential territory, influence, connectivity, etc. are abstract and do not have a mathematical definition. We define *qualities* which are mathematical functions both users want to maximize. Since these *qualities* are different attempts to measure the abstract underlying variables, they are highly correlated. We did parameter tuning analyzing the distribution of each proposed *quality* using a dataset of high level play positions. We also eliminated candidate *qualities* analyzing their redundancy and their use as regressors on a dataset of intentionally unbalanced positions to measure their capacity to predict positional advantage. This work was mainly done by trial and error trying to combine simple definitions with the maximum available information, including the information available from urgency patterns and from ownership maps. Therefore, it has to be considered exploratory and finding the "ideal" set is beyond the scope of the present study.

Tables I and II describe the set of 19 *qualities*. All variables are non-negative when measured for one player. If the variable is a negative measure such as bad shape, it is measured for the opponent. The non-negative measures are used for offline learning using NMF. When computed online, the variables are computed for both players, reduced to a real value by (2) and (3) and subtracted. *Qualities* are called deterministic when they are a function of just the board position and stochastic when they combine the board position with ownership information obtained from previous Monte-Carlo simulations known as ownership maps.

TABLE I. DETERMINISTIC QUALITIES USED IN VERSION 1

Name	***Related go concept***	***Description***
terrCorner	territory	Potential or enclosed territory in the corners for each. Potential territory keeps an additive count of simple shapes. Enclosed territory counts points when the corner is owned by a player.
terrSide234	territory	Potential or enclosed territory along the sides in rows 2, 3, 4. Potential territory counts correct extensions in absence of direct threats. Enclosed territory counts points if groups are simple enough to be analyzed.
speed234	influence	Number of empty cells in rows 2, 3, 4 within the reach (Manhattan distance ≤ 4) of own stones.
speedCentr	influence	Number of empty cells above row 4 within the reach (Manhattan distance ≤ 4) of own stones
strenStones	captures (strength)	Number of stones for each player.
strenLibs	strength	Number of liberties for each player.
strenPincer	strength	Pincered opponent stones along the sides in rows 3, 4
strenUrg	shape (strength)	Weighted sum of urgent points for each player. Bad (= urgent) shape is not about having very urgent cells, but about having many. Urgency is read from a database of patterns learned from dan/pro level play.
AdStrLibs	semeai	Compensated difference (own - opponent) in liberties for each adjacency. An adjacency is a set of 2 groups of different color with stones in contact. All different adjacencies are considered.

TABLE II. STOCHASTIC QUALITIES USED IN VERSION 1

Name	*Related go concept*	*Description*
StrrCorner	territory	Same as terrCorner with each point times p_i.
StrrSid234	territory	Same as terrSide234 with each point times p_i.
StrrStrngT	territory	Number of empty cells with p_i above some threshold.
StrrStrngC	territory	Same as StrrStrngTt but only for points above row 4.
SspeeAlive	influence	Number of empty cells within the reach (Manhattan distance ≤ 4) of own stones surely alive (p_i > some threshold)
SspTimesP	influence	Number of empty cells within the reach (Manhattan distance ≤ 4) of each own stone times p_i.
SstStones	strength	Sum of p_i at each stone for each player. (p_i is the number of times each intersection i belongs to the player divided by the number of simulations.)
SstLibs	strength	Sum of liberties times p_i for each player.
SstMayDie	strength	Number of opponent stones with p_i below some threshold.
SStSurviv	semeai	Compensated difference (own - opponent) in probability of survival for each adjacency. All different adjacencies are considered.

For using dimensionality reduction techniques, it is necessary that the complete dataset can be approximated with fewer variables.

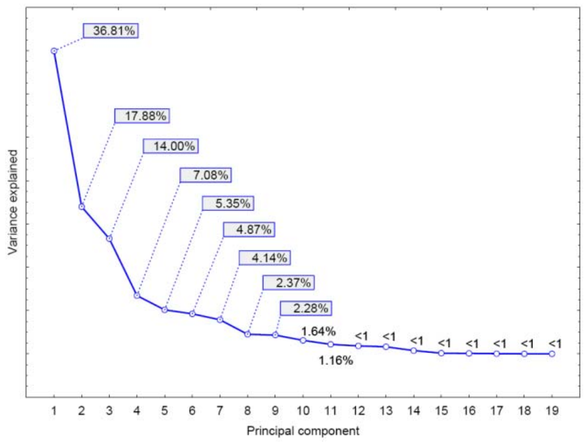

Figure 1. Scree plot of principal components of a dataset of the 19 variables measured on 110,542 board positions of amateur dan and professional games.

We analyzed the multivariate structure of a dataset of 19 variables and 110,542 cases (2 board positions between move 16 and move 62 from a database of 55,271 games played by dan-level and professional go players, included in [1]) using principal component analysis (PCA).

The set of the first 4 principal components explains 75.8% of the total variance of the dataset, while the other 15 principal components explain only the remaining 24.2%. Our set of 19 qualities is therefore a good candidate for dimensionality reduction techniques. We used PCA only to analyze the multivariate structure in order to decide the rank of the encodings matrix, not for reducing the dimensionality. Only NMF was considered for the latter. We considered that the best rank for the intermediate encoding is four, considering: The variance explained by 4 principal components, the number of degrees of freedom for the subsequent online learning and the fact that it is a good number for candidate searches since the number of CPU threads available will usually be a multiple of four. The risk of choosing a high intermediate dimension is the excessive number of parameters for the online learned direction $L'_{d \times 1}$. The offline learned matrix $H^{+}_{p \times d}$ is learned over a massive dataset, it is therefore less probably overfitted to particular conditions that could result in misevaluation. This of course, depends on the online learning algorithm. In our implementation we chose few parameters and simplest possible greedy algorithms for online learning.

This set of variables identifies the difference between KGS dan level and pro level using MANOVA analysis. The database includes games from the KGS archives filtered by ranks of both players, non-handicap and non-blitz. Also, games played by professional players in tournaments are included from two different sources: *go* software collections and revised by professional *go* teachers. Redundant and too short games were filtered to the make the final database of 55,271 games [1]. Results in figure 3 are not surprising: Professionals achieve simultaneously more territory, more influence and more strength than KGS amateurs.

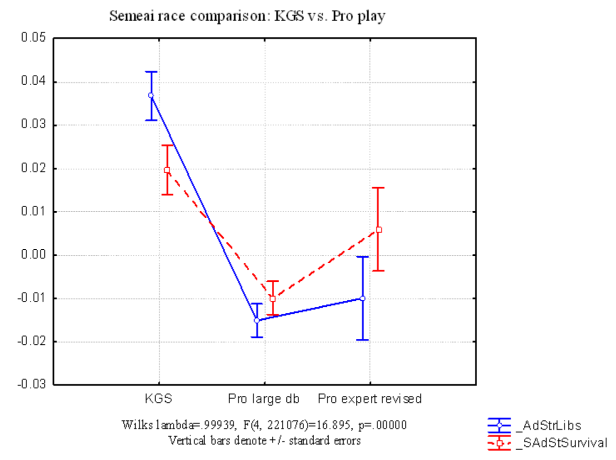

Figure 2. Semeai race comparison between KGS dan amateur players and professional players.

To our knowledge, statistical analysis of human *go* games using functions computed from board positions has never been done before. It is also interesting to note that the trend is inverted in *semeai* races (see figure 2). Professional players fight *semeai* races by a smaller margin than KGS amateur *dan* players. This is consistent with their superior accuracy in judgment.

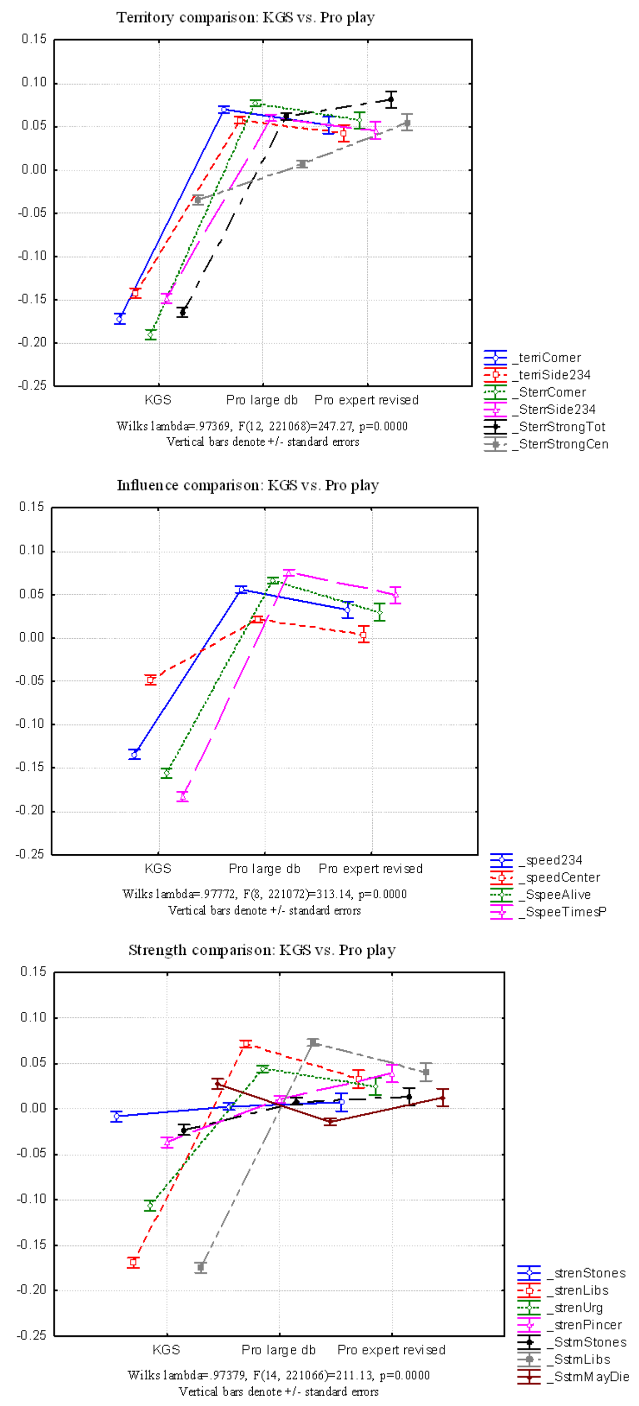

Figure 3. Territory and influence comparison between KGS dan amateur players and professional players.

2) *Implementation in search*

The evaluation function was applied after a short eight move long simulation. In our implementation M-eval is just another MCTS. Each search iteration i is a walk from the root node, down the tree, choosing at each node k the child j that maximizes (for the player who moves at that node) $V_j + UCT_j$, where:

$$V_j = \left(\textstyle\sum_{i\in I(j)} b_i\right)/n_j \quad \left| \begin{array}{l} I(j) = \text{set of all iterations that visited node } j \\ b_i = X_i \times H^{+}_{19\times 4} \times L'_{4\times 1} \\ X_i = \text{the 19 } \textit{qualities} \text{ at the end of iteration } i \\ n_j = |I(j)| \end{array}\right.$$

and:

$$UCT_j = K_{UCT} \times \sqrt{\ln(n_k)/n_j} \quad \left| \begin{array}{l} n_k = |I(k)| \ (k \text{ is the parent of } j) \\ K_{UCT} = \text{constant, defines selectivity} \end{array}\right.$$

The result was propagated upwards just as in a usual MCTS with the only difference that it is a real value instead of a win/loss. Four different $L'_{d\times 1}$ values proposed by the greedy algorithm detailed below were tested. The search included a fixed number of simulations for each $L'_{d\times 1}$ candidate. The list of the top moves found by four M-eval searches and some locally tactical moves were the only moves considered by a subsequent MCTS search. The move finally played by the engine is the best candidate found by MCTS. We used M-eval for guiding MCTS up to move 64 which is more than the usual definition of *fuseki*. After that move, MCTS has reached enough efficiency and the position is usually too complex to be evaluated correctly.

3) *Online learning algorithm*

Online learning of L_{1x4} is based on a simple greedy algorithm. The non-informative unary vector (½, ½, ½, ½) is used as the initial L_{1x4}. At each move four searches $i \in \{1, .. 4\}$ are scheduled using:

$$L_i = L_{1\times 4} + D_i \quad \left| \begin{array}{l} D_i = (\varepsilon \times \delta_{i1}, \varepsilon \times \delta_{i2}, \varepsilon \times \delta_{i3}, \varepsilon \times \delta_{i4}) \\ \delta_{ij} = \text{Kronecker's delta} \\ \varepsilon = \text{small constant} \end{array}\right.$$

The top moves for each of the searches are noted as T_i. The set $M = \bigcup_i T_i \cup L$, where L = *{local tactical moves}* are the only moves considered for the root node of an MCTS that uses about 3/4 of the total computing time for the move. The move played by the program is the best move for that MCTS. The top moves *T(M)* rated by the MCTS are used to evaluate the M-eval searches. After the move is played, a greedy direction is created:

$$D = \textstyle\sum_{i\in\{1,..4\}} w_i \times D_i \quad \left| \, w_i = |T_i \cap T(M)| \right.$$

(I.e., D is the sum of exploration directions times the number of moves being top moves for both M-eval of that direction and MCTS.)

For the next move, L_{1x4} is updated as:

$$\begin{array}{l} L_{1\times 4} \leftarrow L_{1\times 4} + \beta \times D \\ L_{1\times 4} \leftarrow \frac{L_{1\times 4}}{\|L_{1\times 4}\|} \end{array} \quad \left| \, \beta = \text{small constant} \right.$$

C. *Version 2 M-eval*

Since the implementation and testing of version 1 in 2009 described in this paper, M-eval's implementation has been modified in many aspects maintaining the fundamental ideas. This has been done while implementing other improvements to the engine. Testing has been done using both computer vs. computer and computer vs. human play. An extensive empirical test for version 2 similar to that described below for version 1 will be included in the supplementary materials [1].

1) *Version 2 implementation in search*

To implement a consistent interface combining M-eval with other sources of expert knowledge, M-eval is now applied as a short search with a limited number of simulations to provide *a priori* evaluation for MCTS search. This way, the program runs only one algorithm, MCTS. Each time a node is deployed at a depth ≤ 4 during the opening, all candidates moves are evaluated by a short M-eval search. Results are used as *a priori* knowledge to guide the initial simulations of the main MCTS. Also, M-eval is disconnected when the *joseki* pattern matcher no longer recognizes local shapes rather than at a fixed move number.

2) *Version 2 online learning algorithm*

The intermediate rank of the $W_{n\times d}$ matrix is now 2 instead of 4. All searches use the same online learned L_{1x2} which is constant during a complete move search. At the end of the search a new L_{1x2} is computed with the parameters that best fit a linear regression model between the observed (by MCTS) values and the intermediate encodings. Only nodes above a certain number of visits are considered.

3) *Version 2 initial qualities*

The set of *qualities* has been reduced from 19 to 8. Stochastic variables have been removed to avoid the need of previous ownership maps which are complicated to manage and update and their advantage has not been proved empirically. The benefits of speed increase and simplicity has been the main concern. A new method for estimating corner and side territory *terrJosRegr* has been implemented: Regression over the corner and side patterns of the joseki matcher into the territory computed from the expert database games from which the joseki patterns have been learnt. The eight qualities are: Three measures of territory: *terrCorner*, *terrSide234* implemented as in version 1 but applied only when the corner or side pattern is not found in the joseki library, otherwise, *terrJosRegr* is used instead. Two measures of speed: *speed234*, *speedCentr* as in version 1. One measure of stones/liberties which combines the previous *strenStones*, *strenLibs* in one quality. And the version 1 qualities *strenUrg* and *AdStrLibs*.

IV. TESTING RESULTS

Version 1 of M-eval was tested with 220 non-handicap 19x19 games (110 with M-eval as black, 110 as white) with *komi* = 7.5 were played up to move 50 using the GoKnot/QYZ engine. The games used single core search based on fixed number of iterations rather than time settings. The M-eval program scheduled 4x2500 iterations for M-Eval. In case of an M-eval search, one iteration is a simulated game from root until, 8 moves after a leave is found, the position is evaluated. The resulting top move lists were used, as described in III.B.2, to create the root moves for a global MCTS evaluation of 15,000 iterations. Then, for MCTS, one iteration is a simulated game from root until two consecutive passes are produced by the lack of legal non-1-point-eye filling moves. The resulting position is evaluated considering all remaining stones alive.

The non-M-eval opponent was a single 25,000 simulations MCTS. Of course, the 1:1 equivalence between 1 M-eval simulation and 1 MCTS simulation has to be taken with caution. In terms of computing resources, 1 M-eval is faster because the number of moves (8 vs. $\approx$ 500) simulated compensates for the higher complexity of the evaluation. That, of course, is implementation dependent. In our experiments the non-M-eval version used a little more computing resources.

The set of 220 games can be downloaded [1]. The final positions were evaluated using the public *MoGo* binary (see [1]). *MoGo* is a top MCTS engine and the first program to defeat a professional player in 9x9 *go*. *MoGo* evaluated each position five times, with the initial *komi* and with a (4, 8, 12, 16) extra point advantage for the non-M-eval version.

Rationale: It is difficult to measure the influence of an algorithm that is only applied in the beginning of the game. We hypothesized that evaluation at move 50 by the strongest program available at the moment of the study is a more accurate measure of the performance of M-eval than playing the game to the end. The latter would probably introduce more statistical noise in the remaining (about 200) moves. Additionally, this evaluation can be repeated with different *komi* values to draw a naive but consistent picture of the territorial advantage. This experimental setup is exploratory, like other aspects of M-eval, but is statistically sound and aims to exploit available information to the maximum.

TABLE III. RESULTS EVALUATED BY MOGO

Komi	Result		
	Number of wins (form 220)	***%***	***95% CI***
7.5 + 0	176	80.0%	74.21% — 84.74%
7.5 + 4(1)	164	74.5%	68.39% — 79.84%
7.5 + 8(1)	144	65.5%	58.94% — 71.42%
7.5 + 12(1)	128	58.2%	51.57% — 64.51%
7.5 + 16(1)	112	50.9%	44.34% — 57.45%

(1) Additional *komi* advantage for the non M-eval version.

The 80% winning percentage at even *komi* corresponds with a difference of 240 Elo points. The territorial advantage (*komi* difference against M-eval that reestablishes a 50% winning percentage) is about 16 points in 50 moves.

V. Conclusions

A. *Benefits of M-eval*

MCTS is only four years old and has already produced a quantum leap in games research. It is also applicable in other domains like classical operations research problems or econometrics (sometimes using the original name UCT). The fact that MCTS does not require an implicit evaluation function is one of its strong points, but it can also become one of its limitations when the stochastic process is too long or has too much variance to produce low noise evaluation with a reasonable number of simulations. M-eval taken simply as early evaluation can both, replace the noisy final evaluation or guide the stochastic process.

Also, the idea of decomposing dimensional reduction in two steps: The first step, learned offline, minimizing the informational loss and the second, modified during the program's execution, adapting to particular conditions, is applicable in many fields, particularly when underlying not-well-defined abstract concepts are replaced by a set of oversimplified mathematical functions. This form of evaluation function is also applicable in many other fields and can be combined with other types of search, like minimax or depth first proof number search.

B. *Limitations of this study*

This study is exploratory. To our knowledge, it the first work implementing a multivariate evaluation function for opening positions. Therefore, not many alternative methods have been studied. Initial evaluation functions have been designed mainly based on intuition and no alternative dimensionality reduction techniques or learning algorithms have been considered yet.

Also, our research focus (and the utility of M-eval) is mainly for 19x19 computer *go*, and it was implemented on the first version of our MCTS engine which was not among the strong engines in 19x19 at the time experiments were done. It relied too much on offline learned patterns and hard pruning. It is objectionable that an improvement on a weak engine does not necessarily translate into an improvement on a stronger engine. Since then, the engine has improved with the implementation of RAVE [12], [24], [25] and progressive widening [6], [11], [26] and M-eval has also improved and been simplified in version 2. While an empirical study of version 2 is still pending, we consider M-eval an essential component of our program for both computer vs. computer and computer vs. human games and a remarkable improvement in both style and strength.

References

[1] http://www.dybot.com/meval

[2] Bruegmann, B. “Monte-Carlo Go”, ftp://www.joy.ne.jp/welcome/igs/Go/computer/mcgo.tex.Z, 1993

[3] L. Kocsis and C. Szepesvari. “Bandit Based Monte-Carlo Planning.” Lecture Notes in Computer Science, 4212:282, 2006.

[4] T. Cazenave. “Automatic acquisition of tactical Go rules.” In Proceedings of the 3rd Game Programming Workshop, Hakone, Japan, 1996.

[5] T. Kojima. “Automatic Acquisition of Go Knowledge from Game Records: Deductive and Evolutionary Approaches.”, Citeseer, 1998.

[6] T. Cazenave. “Iterative Widening.” In International Joint Conference on Artificial Intelligence, volume 17, pages 523-528. Lawrence Erlbaum Associates Ltd, 2001.

[7] T.G.H. Houeland. “Reuse of Past Games for Move Generation in Computer Go.”, 2008

[8] M. Helvensteijn. “Applying Data Mining to the Study of Joseki.”, Artificial Intelligence in Theory and Practice II, pages 87-96, Springer, 2008.

[9] B. Bouzy and G. Chaslot. “Monte-Carlo Go Reinforcement Learning Experiments.” In IEEE 2006 Symposium on Computational Intelligence in Games, Reno, USA, pages 187-194, 2006.

[10] G. Chaslot, J.T. Saito, B. Bouzy, J. Uiterwijk, and H.J. van den Herik. “Monte-Carlo Strategies for Computer Go.” In Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, pages 83-90, 2006.

[11] R. Coulom. “Computing elo ratings of move patterns in the game of go.” In Computer Games Workshop, Amsterdam, The Netherlands, pages 113-124, 2007.

[12] S. Gelly and D. Silver. “Combining online and offline knowledge in UCT.” In Proceedings of the 24th international conference on Machine learning, pages 273-280. ACM Press New York, NY, USA, 2007.

[13] Y. Wang and S. Gelly. “Modifications of UCT and sequence-like simulations for Monte-Carlo Go.” In Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on, pages 175-182, 2007.

[14] P. Drake and S. Uurtamo. “Move Ordering vs Heavy Playouts: Where Should Heuristics Be Applied in Monte Carlo Go.” In Proceedings of the 3rd North American Game-On Conference, 2007.

[15] J.P. Hoock, A. Rimmel, and O. Teytaud. “Combining expert, offline, transient and online knowledge in Monte-Carlo exploration.”

[16] Cho Chikun. “Positional judgment high-speed game analysis.” Ishi Press, Inc., 1989.

[17] S.J. Yen and S.C. Hsu. “A positional Judgment System for Computer Go.” Advances in Computer Games, 9:313-326, 2001.

[18] P. Paatero and U. Tapper. “Positive Matrix Factorization: A Non-negative Factor Model with Optimal Utilization of Error Estimates of Data Values.” Environmetrics, 5(2):111-126, 1994.

[19] P. Paatero. Least squares formulation of robust non-negative factor analysis. Chemometrics and Intelligent Laboratory Systems, 37(1):23-35, 1997.

[20] D.D. Lee and H.S. Seung. “Learning the parts of objects by non-negative matrix factorization.” Nature, 401(6755):788-791, 1999.

[21] D.L. Donoho and V. Stodden. “When Does Non-negative Matrix Factorization Give a Correct Decomposition Into Parts?” Department of Statistics, Stanford University, 2003.

[22] S. Kullback and R.A. Leibler. “On information and sufficiency.” Annals of Mathematical Statistics, 22(1):79-86, 1951.

[23] D.D. Lee and H.S. Seung. “Algorithms for Non-negative Matrix Factorization.” Advances in Neural Information Processing Systems, pages 556-562, 2001.

[24] D. Helmbold and A. Parker-Wood. “All-Moves-As-First Heuristics in Monte-Carlo Go”

[25] G. Chaslot, C. Fiter, JB. Hook, A. Rimmel and O. Teytaud. “Adding expert knowledge and exploration in Monte-Carlo Tree Search”

[26] G. Chaslot, MHM Winands H.J. van den Herik. “Progressive strategies for Monte-Carlo tree search”, New mathematics and Natural Computation, 2008.