On Move Pattern Trends in Large Go Games Corpus

Petr Baudiš, Josef Moudřík

***Abstract*—We process a large corpus of game records of the board game of Go and propose a way to extract per-player summary information on played moves. We then apply several basic data-mining methods on the summary information to identify the most differentiating features within the summary information, and discuss their correspondence with traditional Go knowledge. We show mappings of the features to player attributes like playing strength or informally perceived “playing style” (such as territoriality or aggressivity), and propose applications including seeding real-work ranks of internet players, aiding in Go study, or contribution to discussion within Go theory on the scope of “playing style”.**

***Index Terms*—board games, go, data mining, pattern recongition, player strength, playing style**

I. Introduction

THE field of Computer Go usually focuses on the problem of creating a program to play the game, finding the best move from a given board position. We will make use of one method developed in the course of such research and apply it to the analysis of existing game records with the aim of helping humans to play the game better instead.

Go is a two-player full-information board game played on a square grid (usually 19×19 lines) with black and white stones; the goal of the game is to surround the most territory and capture enemy stones. We assume basic familiarity with the game.

Many Go players are eager to play using computers (usually over the internet) and review games played by others on computers as well. This means that large amounts of game records are collected and digitally stored, enabling easy processing of such collections. However, so far only little has been done with the available data — we are aware only of uses for simple win/loss statistics (TODO: KGS Stats, KGS Analytics, Pro Go Rating) and ”next move” statistics on a specific position (TODO: Kombilo, Moyo Go Studio).

We present a more in-depth approach — from all played moves, we devise a compact evaluation of each player. We then explore correlations between evaluations of various players in light of externally given information. This way, we can discover similarity between moves characteristics of players with the same playing strength, or discuss the meaning of the ”playing style” concept on the assumption that similar playing styles should yield similar moves characteristics.

P. Baudiš is student at the Faculty of Math and Physics, Charles University, Prague, CZ, and also does some of his Computer Go research as an employee of SUSE Labs Prague, Novell CZ.

J. Moudřík is student at the Faculty of Math and Physics, Charles University, Prague, CZ.

II. Data Extraction

As the input of our analysis, we use large collections of game records[1] organized by player names. In order to generate the required compact description of most frequently played moves, we construct a set of n most occuring patterns (*top patterns*) across all players and games from the database.[2]

For each player, we then count how many times was each of those n patterns played during all his games and finally assign him a *pattern vector* $\vec{p}$ of dimension n, with each dimension corresponding to the relative number of occurences of a given pattern (relative with respect to player’s most played *top pattern*). Using relative numbers of occurences ensures that each dimension of player’s *pattern vector* is scaled to range $[0, 1]$ and therefore even players with different number of games in the database have comparable *pattern vectors*.

A. Pattern Features

We need to define how to compose the patterns we use to describe moves. However, there are some tradeoffs – overly general descriptions carry too few information to discern various player attributes; too specific descriptions gather too few specimen over the games sample and the vector differences are not statistically significant.

We have chosen an intuitive and simple approach inspired by pattern features used when computing ELO ratings for candidate patterns in Computer Go play. [**?**] Each pattern is a combination of several *pattern features* (name–value pairs) matched at the position of the played move. We use these features:

- capture move flag
- atari move flag
- atari escape flag
- contiguity-to-last flag — whether the move has been played in one of 8 neighbors of the last move
- contiguity-to-second-last flag
- board edge distance — only up to distance 4
- spatial pattern — configuration of stones around the played move

The spatial patterns are normalized (using a dictionary) to be always black-to-play and maintain translational and rotational symmetry. Configurations of radius between 2 and 9 in the gridcular metric[3] are matched.

[1]We use the SGF format (TODO) in our implementation.

[2]We use $n = 500$ in our analysis.

[3]The *gridcular* metric $d(x, y) = |\delta x| + |\delta y| + \max(|\delta x|, |\delta y|)$ defines a circle-like structure on the Go board square grid. [**?**]

B. Implementation

We have implemented the data extraction by making use of the pattern features matching implementation within the Pachi go-playing program (TODO). We extract information on players by converting the SGF game records to GTP (TODO) stream that feeds Pachi's `patternscan` engine which outputs a single *patternspec* (string representation of the particular pattern features combination) per move.

III. Data Mining

To assess the properties of gathered *pattern vectors* and their influence on playing styles, we have processes the data using a few basic data minining techniques.

The first two methods (*analytic*) rely purely on data gathered from the game collection and serve to show internal structure and correlations within the data set.

Principal component analysis finds orthogonal vector components that have the largest variance. Reversing the process can indicate which patterns correlate with each component. Additionally, PCA can be used as a vector-preprocessing for methods that are (negatively) sensitive to *pattern vector* component correlations.

A second method – Kohonen maps – is based on the theory of self-organizing maps of abstract units (neurons) that compete against each other for the representation of the input space. Because neurons in the network are organized in a two-dimensional plane, the trained network virtually spreads vectors to the 2D plane, allowing for simple visualization of clusters of players with similar "properties".

Furthermore, we have used two *classification* methods that assign each *pattern vector* $\vec{P}$ some additional data (*output vector* $\vec{O}$), representing e.g. information about styles, player's strength or even a country of origin. Initially, the methods must be nonetheless calibrated (trained) on some expert or prior knowledge, usually in the form of pairs of *reference pattern vectors* and their *output vectors*.

Moreover, the reference set can be divided into training and testing pairs and the methods can be compared by the square error on testing data set (difference of *output vectors* approximated by the method and their real desired value).

k-Nearest Neighbor [1] classifier (the first method) approximates $\vec{O}$ by composing the *output vectors* of k *reference pattern vectors* closest to $\vec{P}$.

The other classifier is based on a multi-layer feed-forward Artificial Neural Network: the neural network can learn correlations between input and output vectors and generalize the "knowledge" to unknown vectors; it can be more flexible in the interpretation of different pattern vector elements and discern more complex relations than the kNN classifier, but e.g. requires larger training sample.

A. Principal Component Analysis

We use Principal Component Analysis *PCA* [2] to reduce the dimensions of the *pattern vectors* while preserving as much information as possible.

Briefly, PCA is an eigenvalue decomposition of a covariance matrix of centered *pattern vectors*, producing a linear mapping o from n-dimensional vector space to a reduced m-dimensional vector space. The m eigenvectors of the original vectors' covariance matrix with the largest eigenvalues are used as the base of the reduced vector space; the eigenvectors form the transformation matrix W.

For each original *pattern vector* $\vec{p}_i$, we obtain its new representation $\vec{r}_i$ in the PCA base as shown in the following equation:

$$\vec{r}_i = W \cdot \vec{p}_i \tag{1}$$

The whole process is described in the Algorithm 1.

Algorithm 1 PCA – Principal Component Analysis

```
Require: $m > 0$, set of players $R$ with pattern vectors $p_r$
  $\vec{\mu} \leftarrow 1/|R| \cdot \sum_{r \in R} \vec{p}_r$
  for $r \in R$ do
    $\vec{p}_r \leftarrow \vec{p}_r - \vec{\mu}$
  end for
  for $(i,j) \in \{1,...,n\} \times \{1,...,n\}$ do
    $Cov[i,j] \leftarrow 1/|R| \cdot \sum_{r \in R} \vec{p}_{ri} \cdot \vec{p}_{rj}$
  end for
  Compute Eigenvalue Decomposition of $Cov$ matrix
  Get $m$ largest eigenvalues
  Most significant eigenvectors ordered by decreasing eigen-
  values form the rows of matrix $W$
  for $r \in R$ do
    $\vec{r}_r \leftarrow W\vec{p}_r$
  end for
```

We will want to find dependencies between PCA dimensions and dimensions of some prior knowledge (player rank, style vector). For this, we use the well-known *Pearson's* χ^2 *test* [3]; the test yields the probability of a null hypothesis that two distributions are statistically independent, we will instead use the probability of the alternative hypothesis that they are in fact dependent.

TODO: Chi-square computation.

B. Kohonen Maps

Kohonen map is a self-organizing network with neurons organized in a two-dimensional plane. Neurons in the map compete for representation of portions of the input vector space. Each neuron $\vec{n}$ represents a vector and the network is trained so that the neurons that are topologically close tend to represent vectors that are close as well.

First, a randomly initialized network is sequentially trained; in each iteration, we choose a random training vector $\vec{t}$ and find the neuron $\vec{w}$ that is closest to $\vec{t}$ in Euclidean metric (we call $\vec{w}$ a *winner*).

We then adapt neurons n from the neighbourhood of $\vec{w}$ employing an equation:

$$\vec{n} = \vec{n} + \alpha \cdot \mathit{Influence}(\vec{w}, \vec{n}) \cdot (\vec{t} - \vec{n}) \tag{2}$$

where α is a learning parameter, usually decreasing in time. $Influence()$ is a function that forces neurons to spread. Such function is usually realised using a mexican hat function or a difference-of-gaussians (see [**?**] for details). The state of the network can be evaluated by calculating mean square difference between each $\vec{t} \in T$ and its corresponding *winner neuron* $\vec{w}_t$:

$$Error(N,T) = \sum_{\vec{t} \in T} |\vec{w}_t - \vec{t}| \quad (3)$$

Algorithm 2 Kohonen maps – training

```
Require: Set of training vectors $T$, input dimension $D$
Require: max number of iterations $M$, desired error $E$
  $N \leftarrow \{\vec{n} | \vec{n}$ random, $dim(\vec{n}) = D\}$
  repeat
    $It \leftarrow It + 1$
    $\vec{t} \leftarrow PickRandom(T)$
    for all $\vec{n} \in N$ do
      $D[\vec{n}] \leftarrow EuclideanDistance(\vec{n}, \vec{t})$
    end for
    Find $\vec{w} \in N$ so that $D[\vec{w}] <= D[\vec{m}], \forall \vec{m} \in N$
    for all $\vec{n} \in TopologicalNeigbors(N, \vec{w})$ do
      $\vec{n} \leftarrow \vec{n} + \alpha(It) \cdot Influence(\vec{w}, \vec{n}) \cdot (\vec{t} - \vec{n})$
    end for
  until $Error(N,T) < E$ or $It > M$
```

C. k-nearest Neighbors Classifier

Our goal is to approximate player's *style vector* $\vec{S}$ based on their *pattern vector* $\vec{P}$. To achieve this, we require prior knowledge of *reference style vectors* (see section V-A).

In this method, we assume that similarities in players' *pattern vectors* uniformly correlate with similarities in players' *style vectors*. We try to approximate $\vec{S}$ as a weighted average of *style vectors* $\vec{s}_i$ of k players with *pattern vectors* $\vec{p}_i$ closest to $\vec{P}$. This is illustrated in the Algorithm 3. Note that the weight is a function of distance and it is not explicitly defined in Algorithm 3. During our research, exponentially decreasing weight has proven to be sufficient.

Algorithm 3 k-Nearest Neighbors

```
Require: pattern vector $\vec{P}$, $k > 0$, set of reference players $R$
  for all $r \in R$ do
    $D[r] \leftarrow EuclideanDistance(\vec{p}_r, \vec{P})$
  end for
  $N \leftarrow SelectSmallest(k, R, D)$
  $\vec{S} \leftarrow \vec{0}$
  for all $r \in N$ do
    $\vec{S} \leftarrow \vec{S} + Weight(D[r]) \cdot \vec{s}_r$
  end for
```

D. Neural Network Classifier

As an alternative to the k-Nearest Neigbors algorithm (section III-C), we have used a classificator based on feed-forward artificial neural networks [**?**]. Neural networks (NN) are known for their ability to generalize and find correlations and patterns between input and output data. Neural network is an adaptive system that must undergo a training period before it can be reasonably used, similarly to the requirement of reference vectors for the k-Nearest Neighbors algorithm above.

1) Computation and activation of the NN: Technically, neural network is a network of interconnected computational units called neurons. A feedforward neural network has a layered topology; it usually has one *input layer*, one *output layer* and an arbitrary number of *hidden layers* inbetween.

Each neuron i is connected to all neurons in the previous layer and each connection has its weight w_{ij}

The computation proceeds in discrete time steps. In the first step, the neurons in the *input layer* are *activated* according to the *input vector*. Then, we iteratively compute output of each neuron in the next layer until the output layer is reached. The activity of output layer is then presented as the result.

The activation y_i of neuron i from the layer I is computed as

$$y_i = f\left(\sum_{j \in J} w_{ij} y_j\right) \quad (4)$$

where J is the previous layer, while y_j is the activation for neurons from J layer. Function $f()$ is so-called *activation function* and its purpose is to bound the outputs of neurons. A typical example of an activation function is the sigmoid function.[4]

2) Training: The training of the feed-forward neural network usually involves some modification of supervised Backpropagation learning algorithm. [**?**] We use first-order optimization algorithm called RPROP [4].

Because the *reference set* is usually not very large, we have devised a simple method for its extension. This enhancement is based upon adding random linear combinations of *style and pattern vectors* to the training set.

TODO: Tohle je puvodni napad?

As outlined above, the training set consists of pairs of input vectors (*pattern vectors*) and desired output vectors (*style vectors*). The training set T is then extended by adding the linear combinations:

$$T_{base} = \{(\vec{p}_r, \vec{s}_r) | r \in R\} \quad (5)$$

$$T_{ext} = \left\{ (\vec{p}, \vec{s}) \;\middle|\; \exists D \subseteq R : \vec{p} = \sum_{d \in D} g_d \vec{p}_d, \vec{s} = \sum_{d \in D} g_d \vec{s}_d \right\} \quad (6)$$

TODO zabudovat g_d dovnitr? where $g_d, d \in D$ are random coeficients, so that $\sum_{d \in D} g_d = 1$. The training set is then constructed as:

$$T = T_{base} \cup SomeFiniteSubset(T_{ext}) \quad (7)$$

The network is trained as shown in Algorithm 4.

3) Architecture details: TODO num layers, num neurons, ..

E. Implementation

We have implemented the data mining methods as an open-source framework "gostyle" [**?**], made available under the GNU GPL licence. We use python for the basic processing and most of the analysis; the MDP library [5] is used for PCA analysis, Kohonen library [**?**] for Kohonen maps. The neuron network classifier is using the libfann C library. [**?**]

[4]A special case of the logistic function, defined by the formula $\sigma(x) = \frac{1}{1+e^{-(rx+k)}}$; parameters control the growth rate (r) and the x-position (k).

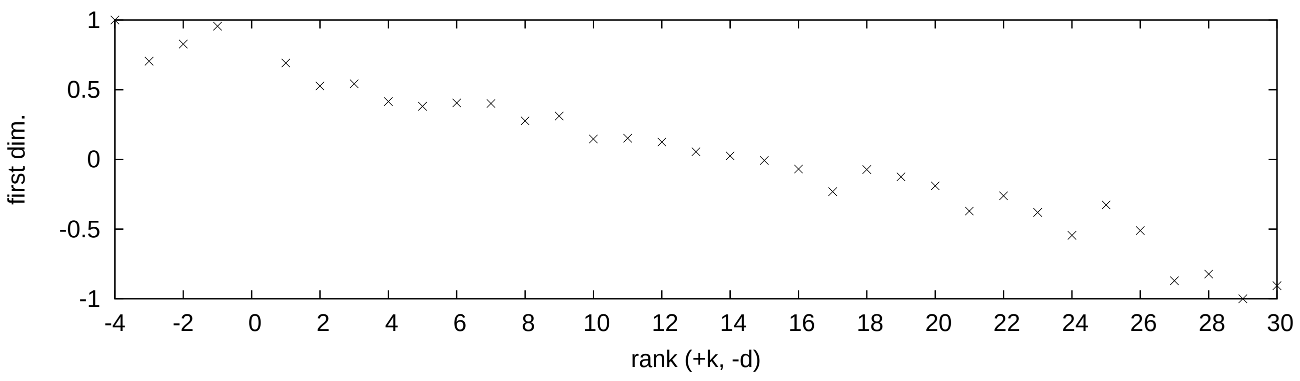

Fig. 1. PCA of by-strength vectors

Algorithm 4 Training Neural Network

```
Require: Train set $T$, desired error $e$, max iterations $M$
  $N \leftarrow RandomlyInitializedNetwork()$
  $It \leftarrow 0$
  repeat
    $It \leftarrow It + 1$
    $\Delta\vec{w} \leftarrow \vec{0}$
    $TotalError \leftarrow 0$
    for all $(Input, DesiredOutput) \in T$ do
      $Output \leftarrow Result(N, Input)$
      $Error \leftarrow |DesiredOutput - Output|$
      $\Delta\vec{w} \leftarrow \Delta\vec{w} + WeightUpdate(N, Error)$
      $TotalError \leftarrow TotalError + Error$
    end for
    $N \leftarrow ModifyWeights(N, \Delta\vec{w})$
  until $TotalError < e$ or $It > M$
```

IV. Strength Estimator

First, we have used our framework to analyse correlations of pattern vectors and playing strength. Like in other competitively played board games, Go players receive real-world rating based on tournament games, and rank based on their rating.[5][6] The amateur ranks range from 30kyu (beginner) to 1kyu (intermediate) and then follows 1dan to 7dan (9dan in some systems; top-level player). Multiple independent real-world ranking scales exist (geographically based) and online servers maintain their own user ranking; the difference can be up to several stones.

As the source game collection, we use Go Teaching Ladder reviews[7] [**?**] — this collection contains 7700 games of players with strength ranging from 30k to 4d; we consider only even games with clear rank information, and then randomly separate 770 games as a testing set. Since the rank information is provided by the users and may not be consistent, we are forced to take a simplified look at the ranks, discarding the differences between various systems and thus increasing error in our model.[8]

First, we have created a single pattern vector for each rank, from 30k to 4d; we have performed PCA analysis on the pattern vectors, achieving near-perfect rank correspondence in the first PCA dimension[9] (figure 1).

In order to measure the accuracy of approximation of strength by the first dimension, we have used the χ^2 test, yielding probability $p = TODO$ that it is dependent on the player strength. Using the eigenvector position directly for classification of players within the test group yields MSE TODO, thus providing reasonably satisfying accuracy.

To further enhance the strength estimator accuracy, we have tried to train a NN classifier on our train set, consisting of one $(\vec{p}, \mathrm{rank})$ pair per player — we use the pattern vector for activation of input neurons and rank number as result of the output neuron. We then proceeded to test the NN on per-player pattern vectors built from the games in the test set, yielding MSE of TODO with TODO games per player on average.

V. Style Estimator

As a second case study for our pattern analysis, we investigate pattern vectors $\vec{p}$ of various well-known players, their relationships and correlations to prior knowledge to explore its correlaction with extracted patterns. We look for relationship between pattern vectors and perceived "playing style" and attempt to use our classifiers to transform pattern vector $\vec{p}$ to style vector $\vec{s}$.

The source game collection is GoGoD Winter 2008 [**?**] containing 55000 professional games, dating from the early Go history 1500 years ago to the present. We consider only games of a small subset of players (fig. I); we have chosen these for being well-known within the players community and having large number of played games in our collection.

A. Expert-based knowledge

In order to provide a reference frame for our style analysis, we have gathered some expert-based information about various

[5]Elo-like rating system [6] is usually used, corresponding to even win chances for game of two players with the same rank, and about 2:3 win chance for stronger in case of one rank difference.

[6]Professional ranks and dan ranks in some Asia countries may be assigned differently.

[7]The reviews contain comments and variations — we consider only the actual played game.

[8]Since our results seem satisfying, we did not pursue to try another collection

[9]The eigenvalue of the second dimension was four orders of magnitude smaller, with no discernable structure revealed within the lower-order eigenvectors.

TABLE I
STYLE ASPECTS OF SELECTED PROFESSIONALS[1]

Player	τ	ω	α	θ
Yoda Norimoto	6.3 ± 1.7	4.3 ± 2.1	4.3 ± 2.1	3.3 ± 1.2
Yi Se-tol	5.3 ± 0.5	6.6 ± 2.5	9.3 ± 0.5	6.6 ± 1.2
Yi Ch'ang-ho[2]	7.0 ± 0.8	5.0 ± 1.4	2.6 ± 0.9	2.6 ± 1.2
Takemiya Masaki	1.3 ± 0.5	6.3 ± 2.1	7.0 ± 0.8	1.3 ± 0.5
Sakata Eio	7.6 ± 1.7	4.6 ± 0.5	7.3 ± 0.9	8.0 ± 1.6
Rui Naiwei	4.6 ± 1.2	5.6 ± 0.5	9.0 ± 0.8	3.3 ± 1.2
Otake Hideo	4.3 ± 0.5	3.0 ± 0.0	4.6 ± 1.2	3.6 ± 0.9
O Meien	2.6 ± 1.2	9.6 ± 0.5	8.3 ± 1.7	3.6 ± 1.2
Ma Xiaochun	8.0 ± 2.2	6.3 ± 0.5	5.6 ± 1.9	8.0 ± 0.8
Luo Xihe	7.3 ± 0.9	7.3 ± 2.5	7.6 ± 0.9	6.0 ± 1.4
Ishida Yoshio	8.0 ± 1.4	5.0 ± 1.4	3.3 ± 1.2	5.3 ± 0.5
Gu Li	5.6 ± 0.9	7.0 ± 0.8	9.0 ± 0.8	4.0 ± 0.8
Cho U	7.3 ± 2.4	6.0 ± 0.8	5.3 ± 1.7	6.3 ± 1.7
Cho Chikun	9.0 ± 0.8	7.6 ± 0.9	6.6 ± 1.2	9.0 ± 0.8
Yuki Satoshi	3.0 ± 1.0	8.5 ± 0.5	9.0 ± 1.0	4.5 ± 0.5
Yamashita Keigo	2.0 ± 0.0	9.0 ± 1.0	9.5 ± 0.5	3.0 ± 1.0
Takao Shinji	5.0 ± 1.0	3.5 ± 0.5	5.5 ± 1.5	4.5 ± 0.5
Miyazawa Goro	1.5 ± 0.5	10 ± 0	9.5 ± 0.5	4.0 ± 1.0
Kobayashi Koichi	9.0 ± 1.0	2.5 ± 0.5	2.5 ± 0.5	5.5 ± 0.5
Kato Masao	2.5 ± 0.5	4.5 ± 1.5	9.5 ± 0.5	4.0 ± 0.0
Hane Naoki	7.5 ± 0.5	2.5 ± 0.5	4.0 ± 0.0	4.5 ± 1.5
Go Seigen	6.0 ± 2.0	9.0 ± 1.0	8.0 ± 1.0	5.0 ± 1.0
Fujisawa Hideyuki	3.5 ± 0.5	9.0 ± 1.0	7.0 ± 0.0	4.0 ± 0.0
Chen Yaoye	6.0 ± 1.0	4.0 ± 1.0	6.0 ± 1.0	5.5 ± 0.5

[1] Including standard deviation. Only players where we got at least two out of tree answers are included.
[2] We consider games only up to year 2004, since Yi Ch'ang-ho was prominent representative of a balanced, careful player until then, but is regarded to have altered his style significantly afterwards.

traditionally perceived style aspects. This expert-based knowledge allows us to predict styles of unknown players based on the similarity of their pattern vectors, as well as discover correlations between styles and proportions of played patterns.

Experts were asked to mark each style aspect of the given players on the scale from 1 to 10. The style aspects are defined as shown:

Styles		
Style	1	10
Territoriality τ	Moyo	Territorial
Orthodoxity ω	Classic	Novel
Aggressivity α	Calm	Figting
Thickness θ	Safe	Shinogi

Averaging this expert based evaluation yields *reference style vector* $\vec{s}_r$ (of dimension 4) for each player r from the set of *reference players* R.

Three high-level Go players (Alexander Dinerstein 3-pro, Motoki Noguchi 7-dan and Vít Brunner 4-dan) have judged style of the reference players. Mean standard deviation of the answers is 0.952, making the data reasonably reliable, though much larger sample would of course be more desirable. The complete list of answers is in table I.

B. Style Components Analysis

We have looked at the three most significant dimensions of the pattern data yielded by the PCA analysis (fig. 2). We have again performend χ^2–test between the three most significant PCA dimensions and dimensions of the prior knowledge style vectors to find correlations; the found correlations are presented in table **??**. We also list the characteristic spatial patterns of the PCA dimension extremes (table **??**).

It is immediately obvious that by far the most significant vector corresponds very well to the player territoriality,[10] confirming the intuitive notion that this aspect of style is the one easiest to pin-point and also most obvious in the played shapes and sequences (that can obviously aim directly at taking secure territory or building center-oriented framework).

The other PCA dimensions are far less obvious — TODO. Kohonen map view.

C. Style Classification

We then tried to apply the NN classifier with linear output function on the dataset and that yielded Y (see fig. Z), with MSE abcd.

VI. PROPOSED APPLICATIONS

We believe that our findings might be useful for many applications in the area of Go support software as well as Go-playing computer engines.

The style analysis can be an excellent teaching aid — classifying style dimensions based on player's pattern vector, many study recommendations can be given, e.g. about the professional games to replay, the goal being balancing understanding of various styles to achieve well-rounded skill set. This was also our original aim when starting the research and a user-friendly tool based on our work is now being created.

We hope that more strong players will look into the style dimensions found by our statistical analysis — analysis of most played patterns of prospective opponents might prepare for the game, but we especially hope that new insights on strategic purposes of various shapes and general human understanding of the game might be achieved by investigating the style-specific patterns.

Classifying playing strength of a pattern vector of a player can be used e.g. to help determine initial real-world rating of a player before their first tournament based on games played on the internet; some players especially in less populated areas could get fairly strong before playing their first real tournament.

Analysis of pattern vectors extracted from games of Go-playing programs in light of the shown strength and style distributions might help to highlight some weaknesses and room for improvements. (However, since correlation does not imply causation, simply optimizing Go-playing programs according to these vectors is unlikely to yield good results.) Another interesting applications in Go-playing programs might

[10] Cho Chikun, perhaps the best-known super-territorial player, is not well visible in the cluster, but he is positioned just below -0.5 on the first dimension.

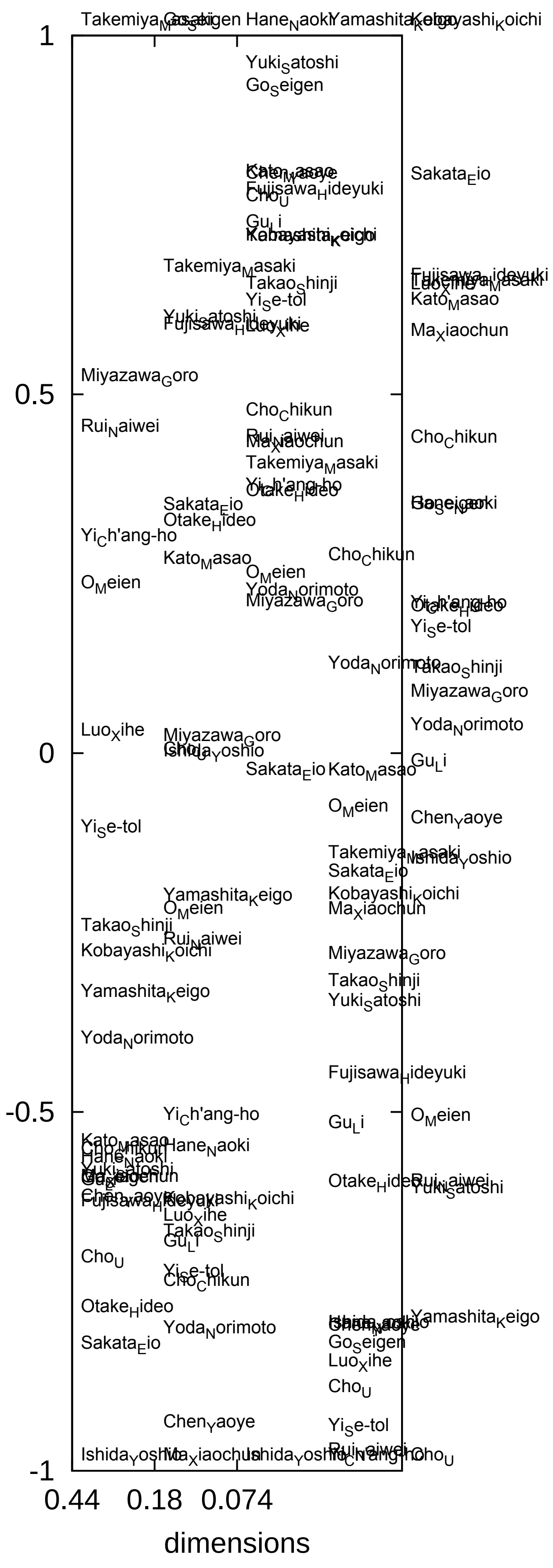

Fig. 2. PCA of per-player vectors

be strength adjustment; the program can classify the player's level based on the pattern vector from its previous games and auto-adjust its difficulty settings accordingly to provide more even games for beginners.

VII. Conclusion

The conclusion goes here. We have shown brm and proposed brm.

Since we are not aware of any previous research on this topic and we are limited by space and time constraints, plenty of research remains to be done. There is plenty of room for further research in all parts of our analysis — different methods of generating the $\vec{p}$ vectors can be explored; other data mining methods could be tried. It can be argued that many players adjust their style by game conditions (Go development era, handicap, komi and color, time limits, opponent) or styles might express differently in various game stages. More professional players could be consulted on the findings and for style scales calibration. Impact of handicap games on by-strength $\vec{p}$ distribution should be investigated.

TODO: Future research — Sparse PCA

Acknowledgment

We would like to thank Radka "chidori" Hanečková for the original research idea and X for reviewing our paper. We appreciate helpful comments on our general methodology by John Fairbairn, T. M. Hall, Robert Jasiek, František Mráz and several GoDiscussions.com users. [**?**] Finally, we are very grateful for detailed input on specific go styles by Alexander Dinerstein, Motoki Noguchi and Vít Brunner.

References

[1] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[2] I. Jolliffe, *Principal Component Analysis*. Springer, New York, 1986.

[3] R. L. Plackett, "Karl pearson and the chi-squared test," *International Statistical Review*, vol. 51, no. 1, pp. 59–72, 1983.

[4] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, 1993, pp. 586–591.

[5] Z. Tiziano, W. Niko, W. Laurenz, and B. Pietro, "Modular toolkit for data processing (mdp): a python data processing framework," *Frontiers in Neuroinformatics*, vol. 2, 2008.

[6] A. Cieply *et al.*, "Egf ratings system — system description." [Online]. Available: http://www.europeangodatabase.eu/EGD/EGF_rating_system.php

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.