Abstract
In this paper, we present a novel method, the random walk method, for comparing and evaluating the chess game and the related situations. We first give the background of the game and the random walk method. We then provide the computational analysis of the random walk method. By applying the method to the Chinese chess game, we can efficiently evaluate the current situation of the game, which in turn can be used to evaluate the similarity of two situations.

Key words: Board Game, Random Walk Method, Dynamic Programming, Distributed Hash Function

1. INTRODUCTION

Chinese chess, as the most popular board game in the world, has a long history. In this paper, we first propose how to construct a Bayesian network based on the states of the board, and then use a random walk method, which is one of the most natural and intuitive mathematical approaches to solve the Chinese chess game. Typically a “drunkard” takes consecutive steps based on his current position, and then walks in a random direction. This method can be applied to one, two, or multiple domains. Although many people did the same area research (Yen et al., 2004; Ong, 2007), none of them use the random walk method. In this paper, we use the random walk method for computing similarities and dissimilarities to evaluate the semantic similarities between game situation and functional similarities of situation (Fouss, Pirotte and Saerens, 2007).

Chinese Chess is a two-player Chinese board game in the same family as Western chess. The present-day form of Chinese Chess originated in China and is therefore commonly called Chinese chess in English. The game represents a battle between two armies, with the object of capturing the enemy’s “general” piece. Distinctive features of Chinese Chess include the unique movement of cannon piece, a rule prohibiting the general (similar to chess kings) from facing each other directly, and the river and palace board features, which restrict the movement of some pieces.

Chinese Chess is played on a board that is 9 lines wide by 10 lines long. In a manner similar to the game, the pieces are played on the intersections, which are known as points. The vertical lines are known as files, while the horizontal lines are known as ranks. With a few awkward substitutions, it is possible to play this game using a Western chess set (Skookumpete, 2010).

Dividing the two opposing sides (between the fifth and sixth ranks) is the river. Although the river provides a visual division between the two sides, only a few pieces are affected by its presence: “pawn” pieces have an enhanced move after crossing the river, while “elephant” pieces cannot cross.

2. BACKGROUND

Chinese Chess pieces are represented by disks marked with a Chinese character identifying the piece and painted in a color identifying to which player the piece belongs. One player’s pieces are usually painted red (or, less commonly, white), and the other player’s pieces are usually painted black (or, less commonly, blue or green).

2.1. General
The generals start the game at the midpoint of the back edge (within the palace). The generals may move one point either vertically or horizontally, but not diagonally. The generals cannot leave the palace except to perform the “flying generals” move. If the two generals face one another on the same file with no other pieces between them, the “flying generals” move can be made one general moves across the board and captures the other. The side who capture the other side’s general is the winner.
2.2. Advisor
The advisors start to the sides of the generals. They move one point diagonally and may not leave the palace, which confines them to five points on the board. They serve to protect the generals.

2.3. Elephant
These pieces move exactly two points diagonally and may not jump over intervening pieces. If an elephant is blocked by an intervening piece, it is known as “blocking the elephant’s eye”. They may not cross the river, thus, they serve as defensive pieces.

2.4. Horse
A horse moves one point vertically or horizontally and then one point diagonally away from its former position. Thus, if there were a piece lying on a point one point away horizontally or vertically from the horse, then the horse’s path of movement is blocked and it is unable to move in that direction.

2.5. Rook
The rook is the most powerful piece as it can move horizontally and vertically for any steps.

2.6. Cannon
Cannons move like the rook, horizontally and vertically, but capture by jumping exactly one piece (whether it is friendly or enemy) over to its target. When capturing, the cannon is moved to the point of the captured piece.

2.7. Pawn
Each side has five pawns. They move and capture by advancing one point. Once they have crossed the river they may also move (and capture) one point horizontally. Pawns cannot move backward, and therefore cannot retreat.

2.8. Play
Because of the size of the board and the low number of long range pieces, there is a tendency for the battle to focus on a particular area of the board.

Which player moves first has varied throughout history, and also varies from one part of China to another. Situational, red goes first in most modern formal tournaments.

Each player in turn moves one piece from the point it occupies to another point. Pieces are not permitted to move through a point occupied by another piece. A piece can be moved onto a point occupied by an enemy piece, in which case the enemy piece is “captured” and removed from the board. A player cannot capture one of his own pieces.

The game ends when one player captures the other’s situation. When the situation is in danger of being captured by the enemy player on his next move, the situation is “in check”. A check should be announced. If the situational player can make no move to prevent the situational capture, the situation is called “checkmate”.

Table I shows the approximate relative values of the pieces. Without considering the position of each piece, we can easily evaluate the values of pieces. However, with considering the position of each piece, we have too many states to consider.

<table>
<thead>
<tr>
<th>Table 1. The single letter piece abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piece</td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td>The pawns before crossing the river</td>
</tr>
<tr>
<td>The pawns after crossing the river</td>
</tr>
<tr>
<td>The advisor</td>
</tr>
<tr>
<td>The elephants</td>
</tr>
<tr>
<td>The horses</td>
</tr>
<tr>
<td>The cannon</td>
</tr>
<tr>
<td>The chariots</td>
</tr>
<tr>
<td>The situations</td>
</tr>
</tbody>
</table>

3. NOTATIONS AND DATA STRUCTURES
Although almost all the AI abstracts the progress of the whole game as a tree, we argue that it is better to abstract it as a pseudo directed acyclic graph (DAG). The initial position is rooted “all”. Due to the daunting amount of data, it’s impossible to save all the states of the game. However, we can use some special way. We use the hash function to obtain a unique value, and this value is the current state of the game. Even in this way,
the number of states is still daunting. We can group different states into one state. We choose the following way,
first, when there is the same amount of pieces; second, we check if there is a checkmate. If there is a checkmate,
it will split into a new state as the chess is the sudden death game.

3.1. Retrograde analysis to reduce the number of states

Retrograde analysis is a technique employed by chess problem solvers to destine which moves were played leading up to a given position. While this technique is rarely needed for solving ordinary chess problems, there is a whole subgenre of chess problems in which it is an important part; such problems are known as retro. In order to reduce the size of the states, we use this method. Sometimes it is necessary to destine if a particular position is legal, with “legal” meaning that it could be reached by a series of legal moves, no matter how bad. Another important branch of retrograde analysis problems is proof game problems.

3.2. Semantic Similarities of Two Pieces

Although the Chinese Chess game has prepare the organized game states, one challenge is to accurately measure the semantic similarity of two pieces, from which we can destine the functional similarities of positions.

3.3. Notations and Definitions

To be able to calculate the information content (IC) of a game state, it is necessary to define a probability function $p(t) : T \rightarrow [0,1]$ for the set $T$ of the game states. The probability of a state $t$ to occur is defined in states of its frequency in the annotations in the game database.

The frequency of a state is defined as

$$ freq(t) = annotation(t) + \sum_{i\in \text{children of } t} freq(i) \quad (1) $$

where $annotation(t)$ is the number of situation products annotated with this state in the game database. $Children of (t)$ is the set of all child states of state $t$.

For each state $t \in T$, we can define $p(t)$ as the sum of the probabilities of finding the state $t$ and all the children of the state $t$:

$$ p(t) = \frac{freq(t)}{freq(root)} \quad (2) $$

It is easy to see that $p(t)$ monotonically increase as one moves from a state to the root, which implies that the $IC(t)$ can be defined in state of $p(t)$.

For the game project, at the location of the state $t$, one moves up toward the root node of the game, $p(t)$ monotonically increase toward a value equal to 1.

According to the principle of information theory, we can naturally define the information content of a state $t$ as

$$ IC(t) = -\log_b(p(t)) \quad (3) $$

where the base for the logarithm, $b$, is not important. Typically the base is 2 or $e$.

4. SOME VARIANTS OF THE RANDOM WALK METHODS

Whether you are a mouse escaping from a cat, or elude your opponents when playing hide-and-seek, or a cryptographer evading a spy, the best strategy is always a random one. If you commit to a destined strategy, the adversary can exploit it, in algorithmic states, he can choose the instance on which it will perform the worst. But if you make unpredictable choices, you may be able to keep him off balance, and do well with high probability regardless of what instance he throws at you. In this paper, we will explore the power of randomized algorithms on chess playing.
There exist a number of interesting variants of random walk models that one “drunkard” or several “drunkards” can take depending on their current positions, past paths, or some other conditions. Before we introduce the details of our random walk method, we first introduce some variants of it which are used in our model.

4.1. Walking Directions

Let U and V denote two sets of game states. When using the random walk method, there are two approaches that the drunkard(s) can take: one direction only, from U to V or from V to U, or two directions simultaneously, from U to V and from V to U.

4.2. Decay Factor

The further the “drunkard” walks away from his home, the less energy he has. The decay factor is introduced to describe this.

4.3. The Maximum Steps to Take

The total steps the random walk method can take in one direction are also an important factor for us to analyze the complexity of the algorithm. For simplicity, we can use the depth of the game DAG as the maximum total steps a “drunkard” needs to take or to analyze the worst case of the algorithm.

4.4. Quantities to Evaluate the Random Walk Method

There are a few quantities to evaluate the random walk method. Two basic quantities to evaluate our random walk method are as follows (Fouss, Pirotte and Saerens, 2007). The first one is the average first passage time while the second one is the average commute time. Typically the latter can only be used on undirected graph and it also lead to much longer walks. In this paper, we let two “drunkards” start to walk simultaneously, the sum of the distances from their starting points to the rendezvous is a natural quantity to evaluate these two “drunkards” random walk method.

4.5. Different Probabilities for Different Game states

Annotation is the process of setting up the connection between game states and situation products. The strengths of game states supporting the situation product can be quantitatively described as the weights of the evidence codes.

5. A FORWARD RANDOM WALKING METHOD FOR MEASURING THE SIMILARITY OF THE TWO PIECES

Based on the review in previous sections, we use forward random walking method from two states to measure the similarity/dissimilarity of two states.

5.1. Notations

Let $G=(V, E)$ be the DAG representing the game states. Without loss of generality, we assume that all edges are pointing from parents to their children in this paper, i.e. all the edges point from old state to more new states. Accordingly, if the walking direction is the same as that of the edge, we call it forward moving; otherwise, we call it backward moving. A unique property of the game DAG is that it has only one root state $r$ (sink), if we take forward moving from anyone state, no matter we consider the hole DAG of the game states.

For any $v \in V$, we let $\delta^+(v)$ denote the sets of out bound edges of state $v$. According to the different properties of edges $e \in E$, we can assume that there is a weight we associated with it. Naturally, we can set according to it is an “is-a” or “part-of” edge. Let $W+(v)$ denote the total weight of all outbound edges from $v$. A random walk moving forward from state $v$ will choose each outbound edge $uv \in \delta^+(v)$ with probability $w_{uv}/W+(v)$.
5.2. Model Description

Without any optimization techniques used, we can simply describe our algorithm as let these two “drunkards” walk forward independently from two states. We then use the combined path of the two paths both from these two states to the random rendezvous as a quantity to evaluate the algorithm. This naive method can be easily deployed into a distributed environment. Without considering it, we can use some optimization techniques to implement the algorithm in a more elegant way.

5.3. Algorithm Description

The algorithm is described as follows. As a preprocessing step, we topologically sort the G in which all states \( v \in V \) are numbered as \( \{1, 2, \ldots, n\} \). The unique root state \( r \) is number 1 in the topological ordering.

Given two states \( a \) and \( b \), we define two forward random paths \( \text{path}_a \) and \( \text{path}_b \) as follows. We start with two states \( u \) and \( v \) initially set to \( a \) and \( b \). As long as \( u \neq v \), we alternatively take a random forward step from \( u \) if \( u > v \) and from \( v \) if \( u < v \). We repeatedly take these steps until \( u = v \). The \( \text{path}_a \) consists of the states traced out by \( u \), and the \( \text{path}_b \) consists of those traced by \( v \). Let \( x_{ab} \) denote the random rendezvous of \( a \) and \( b \) at which \( \text{path}_a \) and \( \text{path}_b \) meet.

As shown in Figure 3, there exists the expected combined average distance from \( a \) and \( b \) forward to \( x_{ab} \). In the topologically sorted \( G \), given any two states \( u \) and \( v \), if \( u \) is an ancestor of \( v \), we define \( d_{max}(v, u) \) and \( d_{max}(u, v) \) the number of average states and the number of longest states of \( v \rightarrow u \) path. The combined random walk distance between \( a \) and \( b \) is defined accordingly as a similarity function, which can be defined to convert distance measures into a similarity function on the unit interval \([0, 1]\) for some appropriate constant.

\[
d(a, b) = \frac{E[d_{max}(a, X_{ab}) + d_{max}(b, X_{ab})]}{E[d_{max}(X_{ab}, \text{root})]}
\]

\[
\text{Sim}(a, b) = \rho^{d(a, b)}
\]

The measure \( d(a, b) \) obtained from the random walk is a very natural and very robust quantity of the dissimilarity between \( a \) and \( b \) in the game as the algorithm requires no additional parameters or constants to be specified, except for the weights of edges, and the constant value \( \rho \) used in the similarity function.

5.4. A Distance-Based Measure for Similarity of Two Situations

All methods we mentioned before distinct the methods of comparing the similarity of two game states and those of comparing two situations which are annotated by two sets of game states and handle them independently. One of the amazing aspects of our method is that it can combine these two kinds of comparisons into one method. Given two situations, \( g_a \) and \( g_b \), which are annotated by two sets of states \( A, B \in V \) respectively, their distance is defined as the average pair wise distance between \( a \in A \) and \( b \in B \)

\[
d(A, B) = \frac{1}{|A| \times |B|} \sum_{a \in A} \sum_{b \in B} d(a, b)
\]

The computation of \( d(A, B) \) can be conveniently reduced to the case of computing the random walk distance between two dummy states because of the linearity of expectation as follows. Two dummy states, \( a' \) and \( b' \), and the corresponding edges, are introduced. We assume that \( |a' a| = 1 \) for each \( a \in A \) and \( |b' b| = 1 \) for each \( b \in B \). Also, all the weights of \( w_{a' a} \) and \( w_{b' b} \) of are uniform. We can easily obtain \( d(A, B) \) from \( d(A, B) = d(a', b') - 2 \).
6. COMPUTATIONAL ISSUES

In this section, we give two methods of how to calculate the similarity/dissimilarity of two game states.

6.1. Random Sampling Numbers

Let \( Z = d_{ave}(a, x_{ab}) + d_{ave}(b, x_{ab}) \). We can estimate \( E[d(A, B)] = E[Z]/E[d(x_{ab}, root)] \) via the random samplings from \( Z \)'s distribution as follows.

We compute two random paths \( \text{path}_a \) and \( \text{path}_b \) by simply performing forward random walks from \( a \) and \( b \). We let \( X_i \) be the combined length of these paths as shown above which we assume that \( X_i \) is iid. If we sample the combined path \( k \) times, where \( k > 0 \), for these \( k \) iterations, our estimate of \( E[Z] \) is then the statistic

\[
X = \frac{1}{k}(X_1 + X_2 + \cdots + X_k).
\]

If we assume that the depth of \( G, D = \max_{v \rightarrow r} d_{\text{max}}(v, r) \) is longest path from state \( v \) to the root \( r \), the running time required to obtain the statistic \( X \) is at most \( O(kD) \) time. Here we use the implication of \( d_{\text{ave}}(a, b) \leq d_{\text{max}}(v, r) \).

As our algorithm works on topologically sorted DAGs, it is easy to be satisfied; otherwise, this estimation time of random sampling would not work.

Note that \( E[X] = E[Z] \) and that \( \delta X = \frac{D}{\sqrt{k}} \leq \frac{D}{\sqrt{k}} \).

According to the Chebyshev inequality we have

\[
P(|X - E[Z]| \geq \varepsilon) \leq \frac{D^2}{\varepsilon^2 k} (7)
\]

which gives us the upper bound of the probability \( X \) away from \( E[Z] \).

If we need to ensure that the probability \( X \) being more than \( \varepsilon \) units away from \( E[Z] \) is at most \( \delta \), we need to take at least \( K = \left[ \frac{D^2}{\varepsilon^2 \delta} \right] \) number of samples to suffice the confidence in that probability. Note that when we need to calculate two set of game states, the samples size should be \( |A||B| \) times of that of the samples of \( d_{\text{ave}}(a^* b^*) \).

6.2. Exact Computation via Dynamic Programming

For any two vertices \( v < u \) such that \( v \) is an ancestor of \( u \), let \( d_{\text{min}}(u, v), d_{\text{ave}}(u, v), \) and \( d_{\text{max}}(u, v) \) respectively denote the number of edges in a shortest \( u \rightarrow v \) path, an average \( u \rightarrow v \) path, and a longest \( u \rightarrow v \) path. More precisely, we can define \( d_{\text{ave}}(u, v) \) as the expected number of edges traversed along the way to \( v \) by a random walk originating at \( u \), with the condition that the random walk arriving \( v \). Also, we define \( \pi(s, v) \) the probability that a random walk originating at \( s \) passes through \( v \). For any source vertex \( s \), we can compute \( d_{\text{min}}(s, v), d_{\text{ave}}(s, v), \) and \( d_{\text{max}}(s, v) \) for all other destination vertices \( v \) in \( O(m) \) time using dynamic programming, based on the following recurrences:

\[
d_{\text{min}}(s, v) = \min_{u \in \delta^+(s)} \{1 + d_{\text{min}}(s, u)\} \quad (8)
\]

\[
d_{\text{max}}(s, v) = \min_{u \in \delta^-(s)} \{1 + d_{\text{max}}(s, u)\} \quad (9)
\]

Figure 3. Combined distance to compute the distance of two situations
\[ d_{av}(s, v) = \frac{\sum (1 + d_{av}(s, v))\pi(s, u)w_u / w^+(v)}{\sum \pi(s, u)w_u / w^+(u)} \] (10)

\[ \pi(s, v) = \sum \pi(s, u)w_u / w^+(u) \] (11)

with \( d_{\text{min}}(s, s) = d_{\text{ave}}(s, s) = d_{\text{max}}(s, s) = 0 \) and \( d(s, s) = 1 \) as initial conditions. In a similar fashion, one can compute \( d_{\text{min}}(u, v), d_{\text{ave}}(u, v), \) and \( d_{\text{max}}(u, v) \) in \( O(m) \) time for all source vertices \( u \) given a particular destination vertex \( v \). We first compute \( d_{\text{ave}}(a, v) \) and \( d_{\text{ave}}(b, v) \) for all vertices \( v \) as a preprocessing step in \( O(m) \) time. Letting \( q(v) = \Pr[v=x_{ab}] \), we then have

\[ E[Z] = \sum q(v)(d_{\text{ave}}(v, a) + d_{\text{ave}}(v, b)). \]

So now all we need to do is to compute \( q(v) \) for all \( v \in V \).

Suppose we let our random paths path_a and path_b continue together in tandem once they meet at \( x_{ab} \), walking all the way forward to the root \( r \). Let us define \( \pi_{ab}(v) = \Pr[v \in \text{path}_a \cap \text{path}_b] \), the probability that both path_a and path_b visit vertex \( v \). Similarly, we define \( \pi_a(v) = \Pr[v \in \text{path}_a \setminus \text{path}_b] \) and \( \pi_b(v) = \Pr[v \in \text{path}_b \setminus \text{path}_a] \) to be the probabilities of \( v \) being visited only by path_a or path_b. Let us also define \( \pi_{ab}(uv), \pi_a(uv), \) and \( \pi_b(uv) \) in an analogous fashion for an edge \( uv \). We now can compute \( q(v), \pi_{ab}(v), \pi_a(v), \) and \( \pi_b(v) \) for all \( v \in V \) in \( O(m) \) time by dynamic programming, using the recurrences:

\[ \pi_a(uv) = \pi_a(u)w_u / w^+(u) \] (12)

\[ \pi_b(uv) = \pi_b(u)w_u / w^+(u) \] (13)

\[ \pi_{ab}(uv) = \pi_{ab}(u)w_u / w^+(u) \] (14)

\[ q(v) = (\sum_{uv \in \delta^+(v)} \pi_a(uv))(\sum_{uv \in \delta^-(v)} \pi_b(uv)) - \sum_{uv \in \delta^-(v)} \pi_{ab}(uv) \] (15)

\[ \pi_{ab}(v) = q(v) + \sum_{uv \in \delta^-(v)} \pi_{ab}(uv) \] (16)

\[ \pi_a(v) = -q(v) + \sum_{uv \in \delta^-(v)} \pi_a(uv) \] (17)

\[ \pi_b(v) = -q(v) + \sum_{uv \in \delta^-(v)} \pi_b(uv) \] (18)

With all variables initially set to zero except for \( \pi_a(a)=1 \) and \( \pi_b(b)=1 \). This takes only \( O(m) \) time as we apply the equations above as we perform a single linear scan through all the vertices and edges in \( G \) in reverse topological order. We spend a total of \( O(1) \) time for each edge \( uv \) evaluating \( \pi_a(uv), \pi_b(uv), \) and \( \pi_{ab}(uv) \), and we spend \( O(|\delta^-(v)|) \) time evaluating \( q(v), \pi_a(v), \pi_b(v), \) and \( \pi_{ab}(v) \) for each vertex \( v \), for a total of \( O(m) \) over all vertices \( v \).

**Figure 4.** Two dummies states introduced to compute two set of states
7. SIMULATION WORK WITH BRUTE FORCE METHOD

In this section, we use some simulation work as an example to demonstrate the correctness and efficiency of our random walk method. We compared our random walk method with the brute force method adapted from (source forge, 2010), both using α-β pruning to reduce the number of states. We run two programs on the Intel Pentium Processor E5500, with 4G memory. We run our game online and the other side, and use a similar elaborating method (Dangauthier, et al., 2007) to calculate the rank of the programs. With the same memory and CPU, and with the same amount of maximum searching time in each step, in Figure 5 and Figure 6, we found that our game is better than the naive brute force method. We argue the main advantage of our game is from the efficient of organizations of the state and the effective of the dynamic programming method. However, when we increase the amount of time for a single step, we found both of the methods are flat. The reason is because the limitation of our programs, i.e. the level of our playing chess.

![Figure 5. The Strength of random walk method with brute force method](image)

Figure 5. The Strength of random walk method

![Figure 6. The Strength of random walk method with brute force method with increased single step time limit](image)

Figure 6. The Strength of random walk method with brute force method with increased single step time limit

8. CONCLUSIONS

In this paper, we propose a novel method to compute the situations of chess game using the random walk method. More, we quantitatively analyze these methods. These methods not only have addressed the accuracy of the game but also the considerations of large scale computation issue. From the examples we compared with other methods, we find that our model is the best one. However, unlike the problems of playing solitaire: we have an opponent, who will do their best to win. Verifying these best moves far exceeds the capabilities of human beings, since it requires us to check every possibility of play states. We can also leverage the latest techniques recently proposed, such as deep learning (Lecun et al., 2015) with deep neural networks and Monte Carlo tree search method (Silver et al., 2016), into our program to increase our program’s self-learning ability and aesthetic pattern, but we believe, for humans with our finite abilities, Chinese Chess will always keep its secrets.

ACKNOWLEDGEMENTS

This work was supported by Fund Project of Hebei Province National Science (F2014209086) 2014.
REFERENCES