Proceedings of the 10th
World Congress on Intelligent Control and Automation
July 6-8, 2012, Beijing, China

"Knowing More Is Less" in Combinatorial Games *

Jing HAN*, Huawei HAN and Xin WANG
Key Laboratory of Systems and Control
Institute of Systems Science
Academy of Mathematics and Systems Science, Chinese Academy of Sciences
No.55, Zhong Guan Cun East Road, 100190, Beijing, China
hanjing@amss.ac.cn

***Abstract* - In Complex Adaptive Systems, agents co-adapt to each other through interaction. A typical example is game: players learn and adapt to the opponent through game playing. This paper studies the adaptive characteristic of co-adaptation through a combinatorial game "Five-in-a-row" focusing on the evaluation function and game tree. The computer simulations show that a high-level player (with a good evaluation function) will win more if she knows the opponent's next move, but a low-level player (with a relatively worse evaluation function) will lose more if she knows the opponent's next move. We call this phenomenon "knowing more is less". To explore the reason and the generality of this phenomenon, an abstract theoretical model is built on a full *k*-ary game tree. Analysis and numerical simulations based on this model prove that "knowing more is less" will happen for a player if her evaluation function accuracy rate is below 0.5. This result indicates that during combinatorial game playing, identification of the opponent only is not enough; the player also need to improve her evaluation function for the board in the sense of mini-max solution as well.**

***Keywords* - combinatorial game, game tree, evaluation function, information, complex adaptive systems**

I. Introduction

Complex adaptive systems (cas) is proposed by John Holland [1][2]and it is defined as 'systems that involve many components that adapt or learn as they interact'. Cas is one of the key problems of Complex Systems research. In a cas, agents are co-adapting to each other through interactions. The current machine learning theories [3] is mainly for a single agent adapting to a static objective. Adaptive control [4] in control systems is to handle uncertainty of the system and environment [5]. There are very few theories about co-adaption and co-evolution [6]. Game is a typical example to study co-adaption because it has the prominent nature of co-evolution: two players learn and adjust their strategies through game playing. The study of how players co-evolve in games will shed light on co-adaptation in cas.

Combinatorial game [7] is an important branch of game. It is almost impossible to get an explicit solution for many combinatorial games, such as *Go*, *Five-in-a-row*, etc. The simple and popular way to find a solution is to exhaustively search the whole game tree, which considers all actions of one player and all reactions of the other player corresponding to each action. It is expanded to final states that can explicitly tell who wins or loses, so the player can make a good choice by searching the whole game tree. This search algorithm is called standard *Minimax* process, which consider the best move given that the opponent also play her best. It minimizes the maximum possible loss. The solution of the standard *Minimax* process is sometimes called the perfect play. Yet for many games, the fully-expanded game trees are huge and the computational complexity of *Minimax* process is not lower than NP [8]: *pspace-complete* for some combinatorial games, *exptime-hard* for some others (such as *Go*). So it is impossible to search the whole tree. To reduce the computation, people use α–β *pruning* [9] to cut the tree. Yet the remained tree is still huge for many games. An effective way is to bound search, for example, stop searching when reaches to a certain depth. In this case, the end-nodes at the given search depth usually are not related to final states. So it is still unknown whether these states lead to ultimate winning or losing. Therefore, as a heuristic, the evaluation function is needed to evaluate these end-nodes. The strategies and style of the player are actually embedded in the evaluation function. It reflects how the player predicts the future from non-final states. The evaluation function is the core of a bounded search algorithm. Constructing a good evaluation function needs expert knowledge. But even a world-class expert can make mistakes when he is playing game. So in many combinatorial games, it remains infeasible to construct a perfect evaluation function, which gets the same values for the non-leaf nodes as what the standard Minimax process does. To play better, the player should keep adjusting the evaluation function during game playing. If two players both learn through interaction during a game, it turns out to be a typical co-adaptation problem: "while the priest climbs a post, the devil climbs ten."

This paper starts from a nontrivial two-player game, Five-in-a-row (FIR, also called *Go-Moku* and *goband*) [10], to explore how two players adapt to each other. The rule of FIR is as follows: two players, Black and White, move in turn by placing a stone of her colour (black or white) on an empty square of an $m \times m$ board; Black starts the game; the player who first makes a line of five consecutive stones of her colour (horizontally, vertically or diagonally) wins the game (see fig.1). The game rule of FIR is very simple so that everyone can play. But it is not easy to play well because it shows very rich and complicated patterns, so high level strategies are crucial to win the game. FIR is not solved yet, i.e., there is no explained solution. The computational complexity of

* Corresponding Author. This work is supported by the National Natural Science Foundation of China (No. 60574068, No. 60821091, and No.60804043) and the Knowledge Innovation Program of the Chinese Academy of Sciences (Grant No. KJCX3-SYW-S01).

searching the game tree of FIR is *pspace-complete* [11], which is not easier than NP. So bounded search with evaluation function is the usual way to find the next move. In this paper, adaptation focuses on changes of the evaluation function. To explore the adaptation problem in FIR theoretically, we focus on the game tree with evaluation function. It is a good example to study co-adaptation between two players.

Following this line, adaptation problems can be studied step by step: 1. how one agent adapts to the other agent with static evaluation function, i.e., how one player adjust herself (including her evaluation function) given that the other agent's evaluation function is fixed; 2. how two agents adapt to each other, i.e., how two players learn from each other while their own evaluation functions are also changing. Before we start step 1, we first discuss what the advantage is if one player has learned the opponent's evaluation function, especially, in the case of one player knowing the other's next move. We have written a computer program to play FIR: computer vs. computer and computer vs. people. An interesting phenomenon is observed from simulations: in the case of player X knows the next move of player Y, player X will play better if X has a good evaluation function, otherwise X will play worse if X doesn't have a good evaluation function. So in some circumstance, "*knowing more is less*". A related theorem is proposed. To explore this phenomenon theoretically, a theoretical game tree search model is built and it can show the same interesting phenomenon. So this phenomenon is general in game tree search, not only can be found in FIR. The analysis based on this model also proves the existence of this phenomenon and finds the critical value for the accuracy rate of the evaluation function to distinguish what is good and bad evaluation function. This result indicates that learning the opponent's strategy (evaluation function or next move) is not enough; one has to improve her own evaluation function. This is what people called "*know both your opponent and yourself ever victorious*".

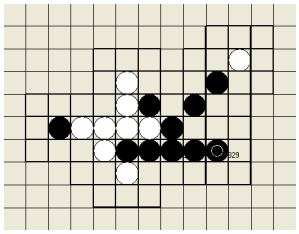


Fig. 1 A case of Five-in-a-row. A five connected pattern for black pieces can be found so black wins in this case.

This paper is organized as follows. In section 2 the FIR game, the computer algorithm and simulation results are described. Then in section 3, we introduce a search model based on the theoretical *k*-ary game tree, and analyze the reason of the phenomenon of "*knowing more is less*". We finish with some conclusions and comments in section 4.

II. FIR: Algorithms and Simulations

A. Algorithms

FIR is proved to be *PSPACE-complete*, which means it is not tractable to decide the outcome (who wins and the rational move) from any arbitrary configuration. The basic idea of making decision of a player is to search the related game tree, where a game tree is a directed graph whose nodes relate to states(positions or configurations) in a game and whose edges relate to moves. The complete game tree for a game is the game tree starting from the initial state and containing all possible moves from each state. It actually lists all possible moves for both players from the beginning to the end of the game. Searching through the whole tree can find the best move for a player in the worst case (the opponent also playing her best) in the zero-sum game, because she has considered all possible following complete sequences of moves.

This recursive search is called *Minimax* algorithm [12]. Both players can use this algorithm to find her best move for a given state. The depth of a node in the game tree is the length of the path to its root. Nodes at odd depth usually represent the choices of move of Black player, while nodes at even depth represent moves of White player. The end-nodes (leaves) of the game tree relate to final states: If Black wins, the value of this leaf node is 1; if Black loses, the value is -1. So nodes at even depth (represent the states that Black faces) prefer child nodes valued 1, which is a *Maximize* operation. Nodes at odd depth (represent states that White faces) prefer child nodes valued -1 which is a *Minimize* operation. By using minimize and maximize alternatively, values of leaves propagate upwards to their parent nodes and finally get to the root. If the root value is 1, Black can find moves (related node valued 1) that can win at the end no matter what the opponent plays. If the root value is -1 and White plays perfectly at every move, Black will lose. So the *Minimax* algorithm finds the solution in the worst case, i.e., the opponent plays perfectly. However, in many combinatorial games, it is impossible that players can play perfectly at every step. Therefore, even thought the root's value is -1, Black still has chance to win if White makes mistakes during game playing.

However, the number of nodes of the complete game tree is $(m^2)!$. For the game of FIR on the 15*15 board, it is $(15^2)!$. In this case, it is impossible to search the whole tree even though redundant sub-trees are removed and α-β *pruning* is used to cut hopeless sub-trees ahead. One popular and effective method is to bound the *Minimax* search: stops searching when reaches depth *D*, then uses evaluation function f:{state}$\rightarrow$[-1,1] to evaluate nodes at depth *D*. The evaluation function gives values to non-final game states without considering all possible following complete sequences of moves: 1 means the state is good for Black; -1 means the state is bad for Black but good for White. These values will propagate upwards to the root, so Black can select a move with the highest value from all child nodes of the root. This method is widely used in many combinatorial games. For example, the chess computer Deep Blue [13] (that beat Garry Kasparov) searches to the depth D = 12, then applied an evaluation function. The following shows the bounded *Minimax* algorithm with depth *D*:

$$Minimax(s,d)=\begin{cases} f(s), & \text{if } s \text{ is a final state or } d=D, \\ \underset{m\in moves(s)}{Max}\{Minimax(s\otimes m,d+1), & \text{if } s \text{ is at even depth}, \\ \underset{m\in moves(s)}{Min}\{Minimax(s\otimes m,d+1), & \text{if } s \text{ is at odd depth}, \end{cases} \quad (1)$$

where s represent the current node/state of the game, d is the search depth so far, moves(s) is the set of all possible moves while facing the state of s, and $s\otimes m$ will create a new node

which represents the new state after taking move m on state s. When a player faces the state of s and it is her turn to move, she will call *Minimax*(s, 0) and then take the move related to the node that directly contribute the value to s. If D=3, the computational complexity is O(m^6).

Notice that players usually do not know the opponent's evaluation function, so in the *Minimax* algorithm (1), a player assume her opponent using the same evaluation function as hers. Yet in reality, usually two players are different and using different evaluation functions. Therefore, even though they use the same *Minimax* algorithm, the search result would be different since their $f(s)$ are different.

The evaluation function f is crucial for the search algorithm. It predicts the future of a non-final state. It leads search and reflexes the style of the player. The expert knowledge and skills are embedded in the evaluation function. Different players can have different evaluation functions. For example, for the game of FIR, we construct an evaluation function f_1:{state}→[-1,1] as follows:

$$f_1(s) = -\sum_{i=2,3,4,5} n_i^W \times 5^i + \sum_{i=2,3,4,5} n_i^B \times 5^i, \qquad (2)$$

where n_i^W is the number of pattern "*live i-connected*" for white piece and n_i^B is the number of pattern "*live i-connected*" for black piece. "*live i-connected*" for white piece is the pattern with i white pieces on a 5-connected row without any black pieces, which means this pattern will have chance to become a 5-connected row for white piece in the future. With more these patterns for white piece, White will have more chance to win. On the other hand, White will be more dangerous if Black have more these patterns for black piece. The term of 5^i is a weigh for each pattern. Obviously "live 4-connected" is more important, i.e., closer to winning, than "live 3-connected". So its weight 5^4 is much larger 5^3. Adaptation of a player can be shown by changes in weights and patterns. Many learning approaches, such as Samuel checker's [14] focus on adjusting the weights during game playing. Yet it will be much more exciting and difficult for the machine to discover new patterns, such as "live double 3-connected", during game playing.

As the first step to study co-adaptation, we consider the case of one knows the other's evaluation function. If one player, for example, say Black, knows the evaluation function of the opponent, say White, and Black also knows that White does not know Black's evaluation function, the *Minimax* algorithm of Black will change to be

$$Minimax_{BkW}(s,d) = \begin{cases} f_B(s), & \text{if } s \text{ is a final state or } d = D, \\ \max\limits_{m \in moves(s)} \{Minimax_{BkW}(s \otimes m, d+1)\}, & \text{if } s \text{ is at even depth}, \\ Minimax_B(s \otimes \arg\min\limits_{m \in moves(s)} \{Minimax_W(s \otimes m, d+1)\}, d+1), & \text{if } s \text{ is at odd depth}, \end{cases} \qquad (3)$$

where $Minimax_W$ is the *Minimax* search of White, with $f(s)$ being replaced by $f_W(s)$ in (1).

$Minimax_{BkW}$ is actually not a *Minimize-Maximize* process; it is a Maximize process instead. In this case that White uses $Minimax_W$ and Black uses $Minimax_{BkW}$ to play the game, Black can know White's next move if $D_B > D_W$ and $D_W < 3$. When Black knows White's next move exactly, $Minimax_{BkW}$ is no longer thinking in the worst case because the information of the opponent reduces the uncertainty. The odd depth nodes have only one child. We can easily prove the following theorem:

Theorem 1: for any $s \in$ {state}, $d > 0$, f_B and f_W, we have $Minimax_{BkW}(s, d) \geq Minimax_B(s, d)$ for the case of Black knows White; similarly, $Minimax_{WkB}(s, d) \leq Minimax_W(s, d)$ for the case of White knows Black.

This indicates if one player (called X) knows the evaluation function (the next move in some cases) of the opponent (called Y), X can always find a 'better' or equally 'good' move, where the standard of 'good' is based on her evaluation function f_X. It means if f_X is a 'good' evaluation function, X will play better; otherwise, X will not play better — or even worse? What is the advantage and disadvantage if one player knows the other's evaluation function (next move)? This problem is explored by the following computer experiments.

B. Simulations

The setting of the computer experiments list below:

[1] The program is written by VisualBasic on the system of Windows XP;

[2] All games are played on a 15*15 board;

[3] The search depth D_X=3 and D_Y=2, so X knows Y's next move if X knows Y's evaluation function;

[4] Two evaluation functions are tested, one is defined in (2), the other is defined as below:

$$f_2(s) = -\sum_{i=2,3,5} n_i^W \times 5^i + \sum_{i=2,3,5} n_i^B \times 5^i. \qquad (4)$$

The difference between f_1 and f_2 is that f_2 does not consider the pattern of "live 4-connected". So f_2 might not be able to stop an indirect dangerous "double live 3-connected" if the search depth $D < 2$. f_2 seems worse than f_1. Yet, it can still beat f_1 sometimes. So it can at least represent a low-level player. Therefore, to explore the problem we proposed above, f_1 can be considered as a relatively good evaluation function while f_2 is considered as a "bad" evaluation function.

[5] The test set consists of 2000 random 6-piece configurations and 3000 random 4-piece configurations: the initial 4 or 6 pieces are randomly placed on a 5*6 area in the center of board (see Fig.2). So the simulation of a game is played by X vs. Y starting from one configuration of the test set.

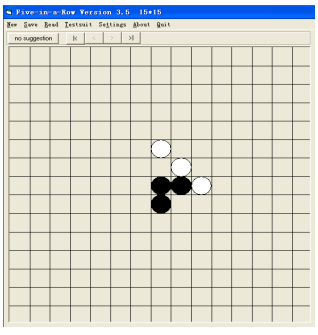


Fig.2. The interface of the program. This snapshot shows a 6-piece configuration.

To answer the question about advantages and disadvantages of the case of one player X knows the next

move of the other player Y, the simulation is done on the test set in different scenarios defined by all combinations of move orders, evaluation functions and who knows who:{X moves first, Y moves first}× {$f_X = f_1$ and $f_Y = f_2$, $f_X = f_2$ and $f_Y = f_1$}× {X and Y don't know each other, X knows Y}. The results of the percentage of X winning the game in all games of the test set are shown in Table 1. Cases of 'X knows Y' are compared to cases of "X and Y don't know each other", so that we can see the advantage and disadvantage of X knows Y's next move.

For two test sets, results are similar: in the scenario of "$f_X = f_1$ and $f_Y = f_2$", i.e., X takes a good evaluation function while Y takes a bad evaluation function, X will win more compared to the case of "X and Y do not know each other" no matter whether X moves first or Y moves first. However, **surprisingly, in the scenario of "$f_X = f_2$ and $f_Y = f_1$", i.e., X takes a bad evaluation function while Y takes a good evaluation function, X will lose much more in the case of "X knows Y" comparing to the case of "X and Y do not know each other" no matter whether X moves first or Y moves first**. Why this phenomenon of "knowing more is less" happens? Is it a general phenomenon of combinatorial games, or FIR only? This problem is studied in the following section.

TABLE I
THE PERCENTAGES OF "X WINS" IN DIFFERENT SCENARIOS IN SIMULATIONS FOR TWO TEST SETS

Test set	Scenario		Don't know each other	X knows Y
2000 random 6-piece configurations	$f_X = f_1$ $f_Y = f_2$	X moves first	92.9%	95.1% ↑
		Y moves first	49.6%	53.8% ↑
	$f_X = f_2$ $f_Y = f_1$	X moves first	77.6%	37.9% ↓
		Y moves first	31.2%	2.3% ↓
3000 random 4-piece configurations	$f_X = f_1$ $f_Y = f_2$	X moves first	92.4%	94.8% ↑
		Y moves first	50.6%	54.1% ↑
	$f_X = f_2$ $f_Y = f_1$	X moves first	77.1%	38.5% ↓
		Y moves first	30.8%	2.6% ↓

X and Y start playing FIR from initial configurations defined by the test set. The numbers inside the table are the percentage of results of X beating Y. The arrows indicate increase(↑) or decease(↓) of the percentage of X beating Y in the case of "X knows Y" compared to the case of "X and Y do not know each other" .

III. Theoretical Game Tree: Search and Analysis

A. Basic Model

It is too complicated to analyze the game tree of FIR and the search process. Even if we can do analysis on FIR, it might not be extended to other games. So in this section, inspired by models of [15], we build an abstract model which is a simplified version of the FIR game tree. This model keeps the essence of the problem, especially still shows the phenomenon of "knowing more is less". What's more, this model is general to two-player combinatorial games.

An H-depth full k-ary game tree is constructed (see Fig.3 for a 3-depth full triple game tree):

a) except leaf nodes, every node has k child nodes;
b) depth of every leaf node is H. So this is a symmetric tree;
c) every node i has a value $v_i \in \{-1, 1\}$, called the *perfect value*, where 1 means Black will win and -1 means White will win, when both players play perfectly(i.e., both players use the standard Minimax process to search entire game tree). 1 and -1 are assigned to leaf nodes with identically independent distribution. For non-leaf nodes i, v_i can be assigned by the standard Minimax process which consider the whole game tree and propagates perfect values from leaf node. Because of symmetry of the tree and i.i.d for the leaf values, the values are i.i.d. for all nodes at the same depth. Let $dep(i)$ denote the depth of node i. For all nodes i satisfies $dep(i) = d$, the probability of $v_i = 1$ is denoted as p_d . According to the Minimax process, we have:

$$p_d = \begin{cases} 1-(1-p_{d+1})^K & d \text{ is even}, \\ p_{d+1}^{\ K} & d \text{ is odd}. \end{cases} \quad (5)$$

To be fair for both players, we assumed $p_0 = 0.5$ for the root, so that two players will have the same probability to win the game if both of them play perfectly. Then for initialization of the game tree, we can get the probability p_H for the leaves at depth H by (5) with p_0=0.5.

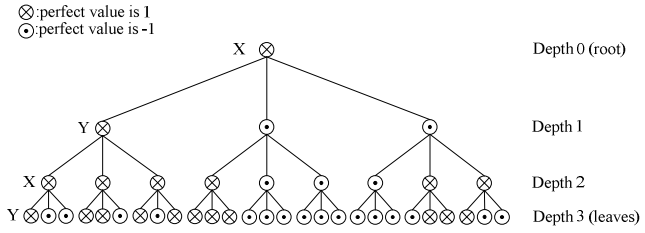


Fig.3 A 3-depth full triple game tree.

Once the game tree is built and initialized by generating v_i for all leaf nodes i, a two-person zero-sum game can be played by two players X and Y. If both of them use the standard Minimax process to search the whole tree, each time they will find a perfect move (it is assumed that the player will make a random choice if she has two or more equally good moves). The result will be consistent with v_0: v_0=1 the one who moves first wins; v_0=-1, the one who moves first loses.

However, as in many combinatorial games, the *Minimax* search is bounded by depth $D<<H$. So an evaluation function is used to give a heuristic value e ($e \in [-1,1]$) to non-leaf nodes. For simplicity, in this paper we assume that the heuristic value e is either 1 or -1, $e \in \{-1,1\}$: for every leaf node i, $e_i=v_i$ since it is easy to judge a final state; for every non-leaf node i, e_i will be equal to the *perfect* value v_i, i.e., $e_i=v_i$, with probability c, and with probability 1-c the node will get a wrong heuristic value, i.e., $e_i=-v_i$. c is called the *accuracy rate*. Evaluation functions with large c are good evaluation functions. If c=1, the related evaluation function f is a perfect evaluation

function which can predict the future in the same way as the standard Minimax process does. Note that c is just a rough measure to the performance of an evaluation function. In fact, the accuracy rate will increase when search the bottom of the tree since the future is clearer when closer to the final states.

To answer the question we have asked in the previous section, we assume that two players play the game with two different evaluation functions f_1 and f_2. Denote c_1 and c_2 to be the accuracy rate for f_1 and f_2 respectively. We know that the phenomena of "Knowing more is less" shown in section II does not relate to the move order; moreover, the theoretical game tree is set to be $p_0 = 0.5$, which is fair to both players. . So the moving order does not matter. The fact that matters is the performance of the evaluation function. We mimic the FIR game on this theoretical model: two players X and Y move alternatively on the H-depth full k-ary game tree; X moves first (starts from the root of the tree, making decision on even depth nodes) and uses f_1 with search depth D_1=2; Y follows X and makes decision on odd depth nodes uses f_2 with search depth D_2=1. The winning probabilities of X in two scenarios are compared: "X and Y do not know each other" vs. "X knows Y's next move".

B. Numerical Experiment and Analysis

The above game playing process can be described as a Markov chain. Note that root node takes perfect value 1 with a probability of 0.5. g_d denotes the probability for the node selected at depth d has perfect value 1. It can be calculated in . Obviously g_H is the ultimate winning probability of X.

$$g_d = \begin{cases} 0.5, & d = 0, \\ g_{d-1} \cdot \varphi_d, & d \text{ is odd and } 0 < d \le H, \\ g_{d-1} + \phi_d \cdot (1 - g_{d-1}), & d \text{ is even and } 0 < d \le H. \end{cases} \quad (6)$$

In the above formula φ_d represents the probability of state transition from 1 to 1 when X makes decision at even depth $d-1$, while ϕ_d represents the probability of state transition from -1 to 1 when Y makes decision at odd depth $d-1$ .They can be calculated as below:

$$\begin{aligned} \varphi_d &= \left(\sum_{\alpha=1}^{k} \binom{k}{\alpha} \cdot (p_d)^{\alpha} \cdot (1-p_d)^{k-\alpha} \cdot t(\bar{x}_d, \bar{y}_d, \alpha) \right) / 1 - (1-p_d)^k, \\ \phi_d &= 1 - \left(\sum_{\alpha=1}^{k} \binom{k}{\alpha} \cdot (1-p_d)^{\alpha} \cdot (p_d)^{k-\alpha} \cdot t(\bar{y}_d, \bar{x}_d, \alpha) \right) / 1 - (p_d)^k, \end{aligned} \quad (7)$$

where $t(\bar{x}_d, \bar{y}_d, \alpha)$ represent the probability that X selects a node i satisfies that v_i=1 and facing α nodes with perfect value 1 and $k-\alpha$ nodes with perfect value -1; while $t(\bar{y}_d, \bar{x}_d, \alpha)$ represents the probability that Y selects a node i satisfies that v_i=-1 and facing α nodes with perfect value -1 and $k-\alpha$ nodes with perfect value 1. It is given as follows:

$$\begin{aligned} t(\eta, \varpi, \alpha) = &\sum_{\beta=1}^{\alpha} \sum_{\gamma=0}^{k-\alpha} \frac{\beta \cdot \binom{\alpha}{\beta} \cdot \binom{k-\alpha}{\gamma} \cdot (\eta)^{\beta} \cdot (1-\eta)^{\alpha-\beta} \cdot (\varpi)^{\gamma} \cdot (1-\varpi)^{k-\alpha-\gamma}}{k-\alpha-\gamma+\beta} \\ &+ \frac{\alpha \cdot (1-\eta)^{\alpha} \cdot (\varpi)^{k-\alpha}}{k}. \end{aligned} \quad (8)$$

Here $\bar{x}_d$ and $\bar{y}_d$ are equal to $x_d(1)$ and $y_d(1)$ respectively, which can be calculated by the following recursive function (9). $x_d(z)$ (or $y_d(z)$) means the probability for being evaluated correctly of a node with perfect value 1(or -1) at depth $d-1+z$ as the player makes decision at depth $d-1$ by using *Minimax* search with bounded depth D.

$$x_d(z) = \begin{cases} 1, & d-1+z \ge H, \\ c, & d-1+D < H, z = D, \\ (x_d(z+1))^k, & d-1+D < H, z < D, d+z \text{ is even}, \\ 1 - \Omega(p_{d+z}, x_d(z+1), y_d(z+1)), & \\ & d-1+D < H, z < D, d+z \text{ is odd}; \end{cases}$$

$$y_d(z) = \begin{cases} 1, & d-1+z \ge H, \\ c, & d-1+D < H, z = D, \\ 1 - \Omega(1-p_{d+z}, y_d(z+1), x_d(z+1)), & \\ & d-1+D < H, z < D, d+z \text{ is even}, \\ (y_d(z+1))^k, & d-1+D < H, z < D, d+z \text{ is odd}; \end{cases} \quad (9)$$

where

$$\Omega(\rho, \delta, \sigma) = \frac{\sum_{\alpha=1}^{k} \rho^{\alpha} \cdot (1-\rho)^{k-\alpha} \cdot (1-\delta)^{\alpha} \cdot \sigma^{k-\alpha}}{\left(1-(1-\rho)^k\right)}.$$

The second scenario is "X knows Y's move" with D_x=2 and D_y=1. Y does the same way as described above. But X is different when she makes decisions at even depth nodes. Suppose X is making decisions at even depth d-2 on node i (see in Fig.4). It has 3 choices corresponding to 3 children nodes of i at depth d-1. For each node at depth d-1, Y has 3 choices that represented by 3 nodes at depth d. X knows exactly which child node at depth d Y will choose. Then the search tree of X is much reduced since there is only one child node remained for each node at odd depth. Therefore, the ultimate winning probability of X, g_H can be calculated as follows:

$$g_d = \begin{cases} 0.5, & d = 0, \\ g_{d-2} \cdot \theta_d + (1 - g_{d-2}) \cdot \psi_d, & d \text{ is even and } 0 < d \le H, \\ g_{H-1}, & d \text{ is odd and } d = H. \end{cases}$$

where θ_d , ψ_d are the probability of $v_j = 1$ when $v_i = 1$ or $v_i = -1$ respectively, where j is the chosen node at depth d, see in Fig.4. They can be calculated as follows:

$$\theta_d = \sum_{\mu=1}^{k}\sum_{\eta=0}^{k-\mu}\binom{k}{\mu}\cdot\binom{k-\mu}{\eta}\cdot\left(p_{d-1}\right)^{\mu}\cdot(1-p_{d-1})^{k-\mu}\cdot\phi_d{}^{\eta}\cdot\left(1-\phi_d\right)^{k-\mu-\eta}$$
$$\cdot t(c_X, c_X, \mu+\eta)/1-(1-p_{d-1})^{k},$$
$$\psi_d = \sum_{\eta=1}^{k}\binom{k}{\eta}\cdot(\phi_d)^{\eta}\cdot(1-\phi_d)^{k-\eta}\cdot t(c_X, c_X, \eta).$$

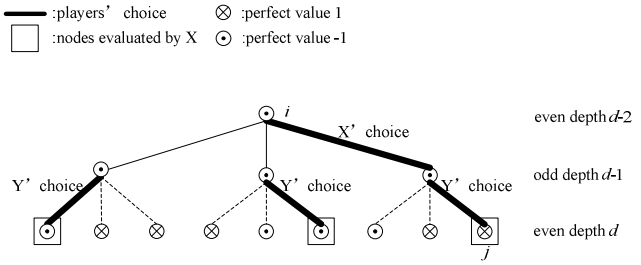
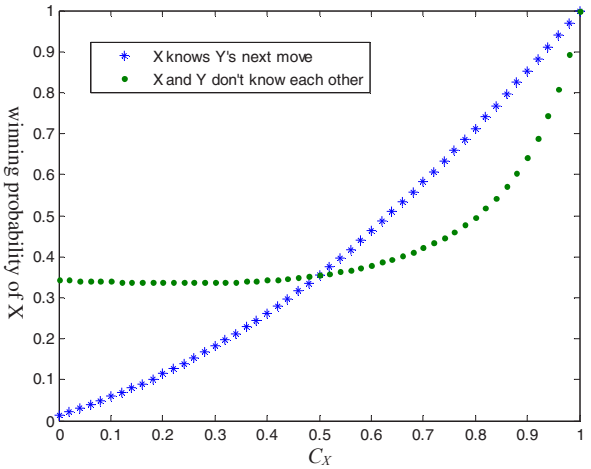


Fig. 4 A case of decision making of X in the scenario of "X knows Y's move". X faces the state represent by node i and she has 3 possible moves represented by 3 child node of i at depth *d*-1. Facing each node at depth *d*-1, Y will choose one corresponding child node with a square frame around it at depth *d*. If X doesn't know Y's next move, X will have to consider all possible moves of Y, which represented by 9 nodes at depth *d*. But in the case of X knows Y's next move, X will only need to evaluate those 3 nodes with square frames rather than all 9 nodes at depth *d*. By comparing these 3 nodes' evaluation values, X chooses the maximum and finally makes her decision shown as the 3rd child node of *i*.

To explore the problem of whether the winning probability of X increases if X knows Y's next move, some numerical experiments are shown in Fig. 5. It can be found that if the *accuracy rate* of evaluation function of player X satisfies $c_X > 0.5$, X knows Y's next move will promote winning probability of X, whereas X knows Y's next move will decrease winning probability of X if $c_X < 0.5$.

(a) 100-depth 3-ary game trees with C_Y=0.8



(b) 100-depth 3-ary game trees with C_Y=0.3

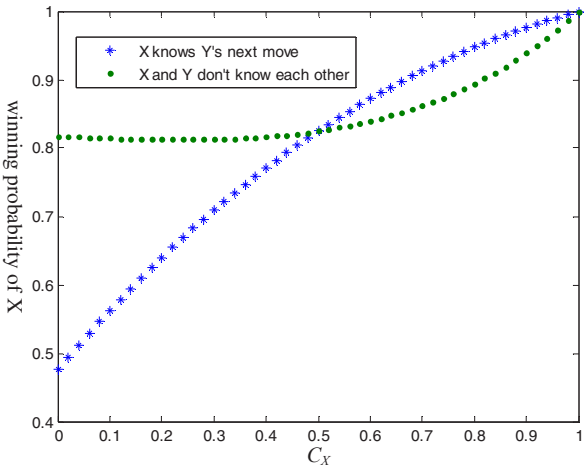


Fig. 5. The winning probability of X in two scenarios: X knows Y's move (stars) vs. X and Y don't know each other.

So far the analysis is based on search depth setting of $\{D_1=2, D_2=1\}$, which is the simplest setting for analysis. Actually, similar analytic results can be obtained for the simulation setting $\{D_1=3, D_2=2\}$. With $D_2>2$, X knowing Y's evaluation function and search depth doesn't means X knows Y's exact next 2 or more moves when X is searching for her best move, because D_1 is fixed and limited. Yet, we still believe that similar results could be obtained.

IV. Discussions and Conclusions

Game based study can be used to probe into complex adaptive systems theoretically because game has the prominent nature of co-adaptation: players learn and adapt to each other through game playing. Based on a classical combinatorial game *Five-in-a-row*, this paper investigates the problem of what is the advantage if one knows the opponent's evaluation function (or next move). A new phenomenon called "knowing more is less" is found from computer simulation, that is, if player X takes a bad evaluation function, X will lose more if X knows Y's next move no matter who moves first. To go deeper, a theoretical model based on an *H*-depth full *k*-ary game tree is built and two players play the game by using the bounded Minimax process to search the game tree. To simplify the problem, the performance of an evaluation function is roughly indexed by the accuracy rate, i.e., the probability of making a correct prediction of the board future (win or lose). This framework is a simplified general model for many combinatorial games. The process of two players playing game based on this model can be described by a Markov chain. The numerical experiment of this model also shows the phenomenon of "knowing more is less", when the accuracy rate is less than 0.5. This indicates that generally in combinatorial games, if a player cannot predict the future of a composition well, say, a low-level player, she will lose more if she knows more about her opponent; on the other hand, a high-level player who can predict the composition well, will

win more if she knows more about the opponent. So knows the opponent only is not enough for a player. If her evaluation function is not well constructed, the judgment to the composition will be unreliable. Thus with more information, it even leads to worse decisions.

Based on the result of this paper, it is very important for a player to adjust her evaluation function except to identify the strategy of the opponent. In the future, we will continue to study the problems of how one player adapts to the other and how two players co-adapt to each other. We believe more new properties about co-adaptation will be discovered under this framework.

"Knowing more is less" is a special property in games -- not only in combinatorial games. The similar phenomenon called "price of information" is found in paper [16] based on the *n*-person linear-quadratic differential games. Their approach is from a different angle and the reason is different. We believe the approach of this paper can be extended to other complete and perfect information games with boundedly rational players, including non-zero sum games. We will systematically study how strategic information of the opponent affects the result of these games. All these indicate the special property of information in game playing, which reflect partly the essence of co-adaptation. But what makes adaptation in games different from other problems? What is the essence of a game from the computational/mathematical viewpoint? Will anything new about adaptation be found from games? We need further study to answer the above questions.

.

REFERENCES

[1] J. H. Holland, Studying Complex Adaptive Systems, Journal of Systems Science and Complexity , 19:1-8, 2006.

[2] J. H. Holland, Hidden Order: How Adaptation Builds Complexity, Addison-Wesley, 1995.

[3] T. Mitchell, Machine Learning, McGraw Hill.1997.

[4] L. Guo, Adaptive systems theory: some basic concepts, methods and results, Journal of Systems Science and Complexity, Vol.16 No. 3, pp.293-306, July, 2003.

[5] This comparison between cas and adaptive control is based on the discussions with John Holland, Lei Guo, Jiang Zhang, Zhixin Liu, Gongguo Tang and Jing Han, Beijing, June 2006.

[6] P. R. Ehrlich, and P. H. Raven, Butterflies and Plants: A Study in coevolution, Evolution, Vol. 18, No. 4. (Dec., 1964), pp. 586-608.

[7] E. R. Berlekamp, J. H. Conway, R. K. Guy, *Winning Ways for your Mathematical Plays*, Academic Press, 1982.

[8] A. S. Freankel, Complexity, Appeal, and Challenges of Combinatorial Games, Theoretical Computer Science, Volume 313, Issue 3, 2004, Pages 393-415.

[9] J. Pearl， The solution for the branching factor of the alpha-beta pruning algorithm and its optimality, Communications of the ACM, 1982

[10] E. Lasker, Go and go-moku: the oriental board games, Courier Dover Publications, 1960.

[11] S. Reisch, Gobang ist PSPACE-vollst¨andig, Acta Informatica, 13:59–66, 1980.

[12] H. Kaindl, Minimaxing Theory and Practice, AI Magazine, Volume 9 Number 3, 1988.

[13] M. Campbell, A. J. Hoane, and F. Hsu, Deep Blue, Artificial Intelligence, Vol.134, Issues 1-2, Pages 57-83, 2002.

[14] A. L. Samuel, Some Studies in Machine Learning Using the Game of Checkers II-Recent Progress, IBM Journal, November 1967.

[15] M. Lustrek, M. Gams, I. Bratko, Is real-valued minimax pathological, Artificial Intelligence, Vol.170, pp: 620 – 642, 2006.

[16] Q. Zhu, and T. Basar, Prices of Anarchy, Information, and Cooperation in Differential Games, Dynamic Games and Applications, Volume 1, Number 1, pp: 50-73, 2011.