JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 20, 105-126 (2004)

Optimal Algorithms for 2 × n AB Games – A Graph-Partition Approach*

SHAN-TAI CHEN AND SHUN-SHII LIN+
Department of Information and Computer Education
+*Graduate Institute of Computer Science and Information Engineering*
National Taiwan Normal University
Taipei, 106 Taiwan
E-mail: linss@csie.ntnu.edu.tw

This paper presents new and systematic methodologies to analyze deductive games and obtain optimal algorithms for 2 × n AB games, where n ≥ 2. We have invented a graphic model to represent the game-guessing process. With this novel approach, we find some symmetric and recursive structures in the process. This not only reduces the size of the search space, but also helps us to derive the optimum strategies more efficiently. By using this technique, we develop optimal strategies for 2 × n AB games in the expected and worst cases, and are able to derive the following new results: (1) $\lceil n/2 \rceil + 1$ guesses are necessary and sufficient for 2 × n AB games in the worst case, (2) the minimum number of guesses required for 2 × n AB games in the expected case is $(4n^3 + 21n^2 - 76n + 72)/12n(n-1)$ if n is even, and $(4n^3 + 21n^2 - 82n + 105)/12n(n-1)$ if n is odd.

The optimization of this problem bears resemblance with other computational problems, such as circuit testing, differential cryptanalysis, on-line models with equivalent queries, and additive search problems. Any conclusion of this kind of deductive game may be applied, although probably not directly, to any of these problems, as well as to any other combinatorial optimization problem.

Keywords: AB game, algorithms, game tree, mastermind, search strategies

1. INTRODUCTION

The game of Mastermind is a deductive game for 2 players: a codemaker and a codebreaker. The codemaker chooses a secret code consisting of four pegs out of six possible colors. Repeated colors are allowed, so the set of possible codes is $6^4 = 1296$. The codebreaker then tries to guess the code. After each guess, the codemaker responds with a hint that consists of black and white pegs; a black peg means that a peg in the codebreaker's guess is correct in both position and color; a white peg means that a peg in the guess is correct in color but not in position; and finally, no pegs means that there are pegs in the guess which are not correct in color. The purpose of the game is to solve the code (i.e., get four black pegs) in the smallest number of guesses.

Another well-known deductive game in England and Asia, called "Bulls and Cows" [1] or the AB game, is a variant of the Mastermind game. The difference is that all the

Received January 31, 2003; accepted July 4, 2003.
Communicated by Ming-Syan Chen.
* A preliminary version [15] of this paper has been presented at the 2002 International Computer Symposium, 2002.

digits of the code in the game must be distinct, but 10 colors are allowed. Hence, the set of possible codes is the number of permutations: P(10, 4) = 10 * 9 * 8 * 7 = 5040. Now we restate the game with a more precise description. The codemaker chooses a secret code (s_1, s_2, s_3, s_4). After each guess (g_1, g_2, g_3, g_4) made by the codebreaker, the codemaker responds with a pair of numbers [A, B], where A is the number of "direct hits," i.e., the number of positions j such that $s_j = g_j$, and B is the number of "indirect hits," i.e., the number of positions j such that $s_j \neq g_j$ but $s_j = g_k$ for some position $k \neq j$. For example, if the secret code is (1, 2, 3, 4) and the guesses are (3, 1, 5, 4) and (3, 1, 4, 5), then the responses are [1, 2] and [0, 3], respectively. The goal of the codebreaker is, based on the responses, to minimize the number of guesses needed, and to find the secret code.

Over the past three decades, much research has been done on this kind of game. Knuth [1] demonstrated a strategy for the Mastermind game that requires at most five guesses in the worst case and 4.478 in the expected case. The strategy used in [1] is to choose the guess that minimizes the maximum number of remaining possibilities at every stage. Later, Irving [2] and Nerwirth [3] used sophisticated heuristic strategies to improve the bounds in the expected case to 4.369 and 4.364, respectively. Finally, Koyama and Lai [4] used a recursive backtracking method to determine the optimal strategy for Mastermind, where the expected number of guesses is 4.34. Also, variants of the Mastermind game have been studied in [5, 6], and [7]. Furthermore, in [8, 9] and [10], the authors used evolutionary algorithms and genetic algorithms to solve related problems. More recently, Roche [11] analyzed the generalized Mastermind and obtained asymptotical bounds under some conditions. Kabatianski and Thorpe [12] investigated the Mastermind game and its related applications based on coding theorem.

Merelo et al. [10] transferred the optimal strategy for Mastermind game to a combinatorial optimization problem. It bears resemblance to other computational problems, such as circuit testing, differential cryptanalysis, on-line models with equivalent queries, and additive search problems. Consequently, any conclusion of this kind of deductive game may be applied, although probably not directly, to any of these problems, as well as to any other combinatorial optimization problem.

To describe and compare the variants of these games, we briefly introduce the notation defined in [5]. The Mastermind game is denoted by MM4 × 6, signifying four pegs and six colors with repetition of colors allowed. The AB game is denoted by MM4 × 10N, signifying four digits (pegs) and ten symbols (colors) with repetition of symbols prohibited. Likewise, MM2 × nN signifies two digits and an arbitrary number n of symbols with repetition of symbols prohibited.

Because the complexity of these games grows at an exponential rate, no optimal strategy for them when they have higher dimensions (i.e., when the games have more than 4 pegs and 6 colors, i.e., 4 × 6) has yet been found. In this paper, we develop a systematic methodology to discover the optimal strategies for general AB games with 2 digits—*MM2 × nN games* (or *2 × n AB games* in this paper, where $n \geq 2$).

This paper is organized as follows. In section 2, we introduce some properties of game trees and use the binary search technique to determine the optimum strategy for a simple deductive game. In section 3 we present the graphic model by means of a MM2 × 5N game. The optimal strategy for 2 × n AB games is developed in section 4. Section 5 contains our concluding remarks.

2. AN OPTIMUM STRATEGY FOR A SIMPLE DEDUCTIVE GAME

In this section, we deal with a simple type of deductive games, 1 × n games. By means of some properties of game trees, we will show how to determine the minimum numbers of guesses required both in the expected and the worst case for the game. By means of this comparatively simple work, we present some fundamental concepts that can be applied to develop the optimal strategy for 2 × n AB games in section 3 and section 4.

2.1 The Game Trees for Deductive Games

In 1 × n deductive games, the codemaker chooses a *secret number S*, S ∈ {0, 1, 2, …, n – 1}. After each guess g_i made by the codebreaker, the codemaker responds with a hint H_i , $H_i \in$ {<, =, >}, three elements of which refer to $S < g_i$, $S = g_i$, and $S > g_i$, respectively. The goal of the codebreaker is, based on the hints, to minimize the number of guesses required, and to find the secret number. Obviously, the guessing process for this game can be translated into a search problem. We can obtain the optimum strategy for this game by using the *binary search* technique, which is shown in Theorem 1. In order to demonstrate how to calculate the number of guesses required in the worst and expected cases for 1 × n games, we illustrate our strategy by way of a 1 × 16 game, the game tree *T* which is shown in Fig. 1.

In Fig. 1, the number in each node represents each guess g_i. We start by comparing the secret number S with the middle key of the possible secret numbers, which is 7 in the root in this case. According to the hint given by the codemaker, we decide which subtree should be chosen to guess next, and the same procedure can be used again until S is *hit*. More precisely, if the secret code S is equal to g_i, then our strategy goes down to the leaf under node g_i, which means we *hit* the code and the game is finished. If $S < g_i$, then our strategy follows the left subtree; similarly if $S > g_i$, the right subtree is used. For example, if S = 5, our guessing sequence will be 7, 3, and then 5 because S < 7, S > 3, and S = 5, where 3 guesses are required to hit the secret number. On the other hand, if S = 15, then our guessing sequence will be 7, 11, 13, 14, and then 15, where 5 guesses are required to finish the game. By doing so as shown in Fig. 1, we can easily obtain the following two observations. These observations can be applied to analyze arbitrary deductive games.

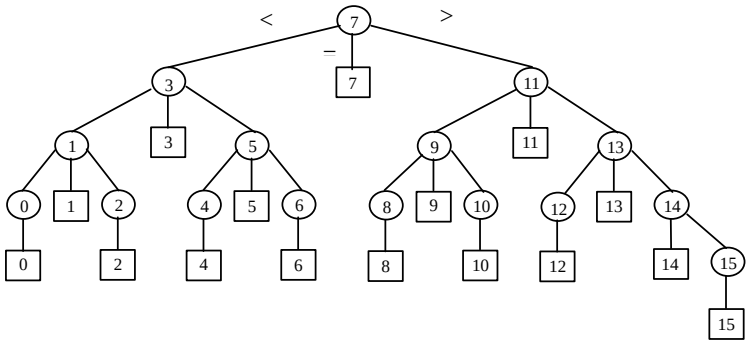


Fig. 1. A game tree T for a 1 × 16 game, where the binary search strategy is used.

Observation 1 The number of guesses required in the worst case for a game is H, where H is the height of the game tree, i.e., the length of a longest path from the root to a leaf in the game tree. For example, H = 5 in Fig. 1.

Observation 2 The number of guesses required in the expected case for a game is L/n, where L is the *external path length* [13] of the game tree; i.e., the sum of the distances from the root to each leaf in the game tree. For example, in Fig. 1, L = 1 * 1 + 2 * 2 + 3 * 4 + 4 * 8 + 5 * 1 = 54 and the number of guesses required in the expected case is L/n = 54/16 = 3.375.

Notice that if we remove all the leaf nodes from Fig. 1, then the game tree built by the binary search technique is always a full binary tree, as shown in Fig. 2. In the following paragraphs, to take advantage of the properties of binary trees, we ignore the leaf nodes of the game tree, thus forming a full binary tree T_b, whose *height* and *total path length* are denoted by H_b and L_b, respectively, where L_b is the sum of the distances from the root to each *node* in the game tree. It is easy to show that the number of guesses required in the worst case and the expected case for a 1 × n game will be $H_b + 1$ and $L_b/n + 1$, respectively.

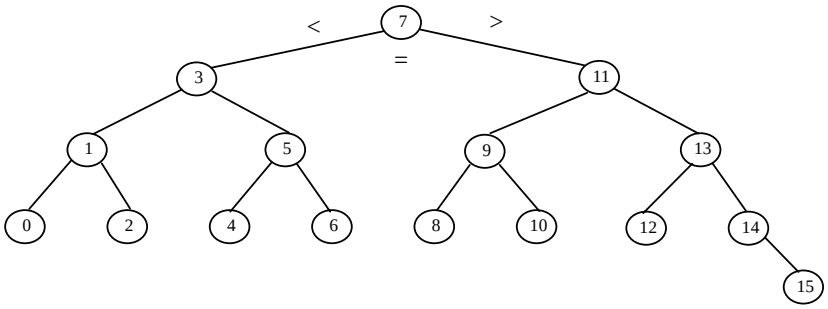


Fig. 2. A full binary game tree T_b obtained by removing all the leaf nodes from the game tree T shown in Fig. 1.

2.2 The Minimum Number of Guesses Required for a Deductive Game

In Theorems 1 and 2, we will present an optimum strategy based on the binary search technique and derive the minimum numbers of guesses required in the worst case and expected case. Before that, we will look at a property [14] concerning the minimum total path length in a binary tree, which we need to prove Theorem 1, as shown in Lemma 1.

Lemma 1 A binary tree has the minimum *total path length* if and only if all its external nodes occur on at most two adjacent levels.

***Proof* [14]:** Omitted. ❑

Theorem 1 $\lfloor \log_2 n \rfloor + 1$ guesses are necessary and sufficient for 1 × n games in the worst case.

Proof: The necessary and sufficient conditions can be proven by the following two facts: First, the binary game tree, T_b, built by the binary search technique is always *full*, which means that T_b always has minimum height. Second, the height H_b of a *full* binary tree with n nodes is $\lfloor \log_2 n \rfloor$. Therefore, the number of guesses required in the worst case is $H_b + 1 = \lfloor \log_2 n \rfloor + 1$. This then proves the result. ❑

Theorem 2 The minimum number of guesses required for 1 × n games in the expected case is $((n + 1)(\lfloor \log_2 n \rfloor - 1) + 2)/n + 1$ if $n = 2^k - 1$, where $k \in Z$; and it is $((n + 1)\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2)/n + 1$ if $n \neq 2^k - 1$, where $k \in Z$.

Proof: *Necessary:* Since the game tree, T_b, built by binary search techniques is always *full*, all external nodes of T_b occur on at most two adjacent levels. By Lemma 1, the necessary condition is proven.

Sufficient: We divide the proof into two cases to prove this property.

Case 1. If $n = 2^k - 1$, then the game tree T_b for the game is a *complete binary tree*. In this case $L_b = \sum_{i=1}^{\lfloor \log_2 n \rfloor} i * 2^i = (\lfloor \log_2 n \rfloor - 1) * 2^{(\lfloor \log_2 n \rfloor + 1)} + 2 = (n+1)(\lfloor \log_2 n \rfloor - 1) + 2.$

The number of guesses required is $L_b/n + 1 = ((n + 1)(\lfloor \log_2 n \rfloor - 1) + 2)/n + 1$.

Case 2. If $n \neq 2^k - 1$, then the game tree T_b for the game is full but not complete. The number of nodes at the bottom level is only $n - (2^{\lfloor \log_2 n \rfloor} - 1)$ rather than $2^{\lfloor \log_2 n \rfloor}$ as in a complete binary tree, and the distance from the root to each node in this level equals $\lfloor \log_2 n \rfloor$. Therefore, we should subtract the quantity $\lfloor \log_2 n \rfloor * (2^{\lfloor \log_2 n \rfloor} - (n - (2^{\lfloor \log_2 n \rfloor} - 1))/n = \lfloor \log_2 n \rfloor * (2^{\lfloor \log_2 n \rfloor + 1} - n - 1)/n$ from the formula obtained in Case 1. That is, $L_b = 2^{(\lfloor \log_2 n \rfloor + 1)} (\lfloor \log_2 n \rfloor - 1) + 2 - (\lfloor \log_2 n \rfloor * (2^{\lfloor \log_2 n \rfloor + 1} - n - 1)) = (n + 1)\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2$. This completes the proof. ❑

We can now apply the results of Theorems 1 and 2 to a 1 × 15 game, the game tree for which is complete, and to a 1 × 16 game, the game tree for which is not complete. For the 1 × 15 game, the numbers of guesses required in the worst and expected cases are 4 and $((15 + 1)(\lfloor \log_2 15 \rfloor - 1) + 2)/15 + 1 = (16 * 2 + 2)/15 + 1 = 3.2667$, respectively. For the 1 × 16 game, the numbers of guesses required in the worst and expected cases are 5 and $((1 * 2 + 2 * 4 + 3 * 8 + 4 * 16) - (4 * 15))/16 + 1 = 3.375$, respectively.

3. A NEW GRAPHIC MODEL

In this section, we present a graphic model to represent the guessing process of the deductive games. The main idea behind the proposed method is to represent the set of codewords compatible with the responses given so far as graphs for 2 × n games. By using this methodology, we can easily discover the symmetric, isomorphic, and recursive properties to reduce the search space of the game. Furthermore, we are able to develop an optimal strategy for 2 × n AB games in the next section.

3.1 The Graphic Model

First, we introduce the graphic model by means of a 2 × 5 AB game for simplicity. We start with some definitions. The game tree for the 2 × 5 AB game is shown in Fig. 3.

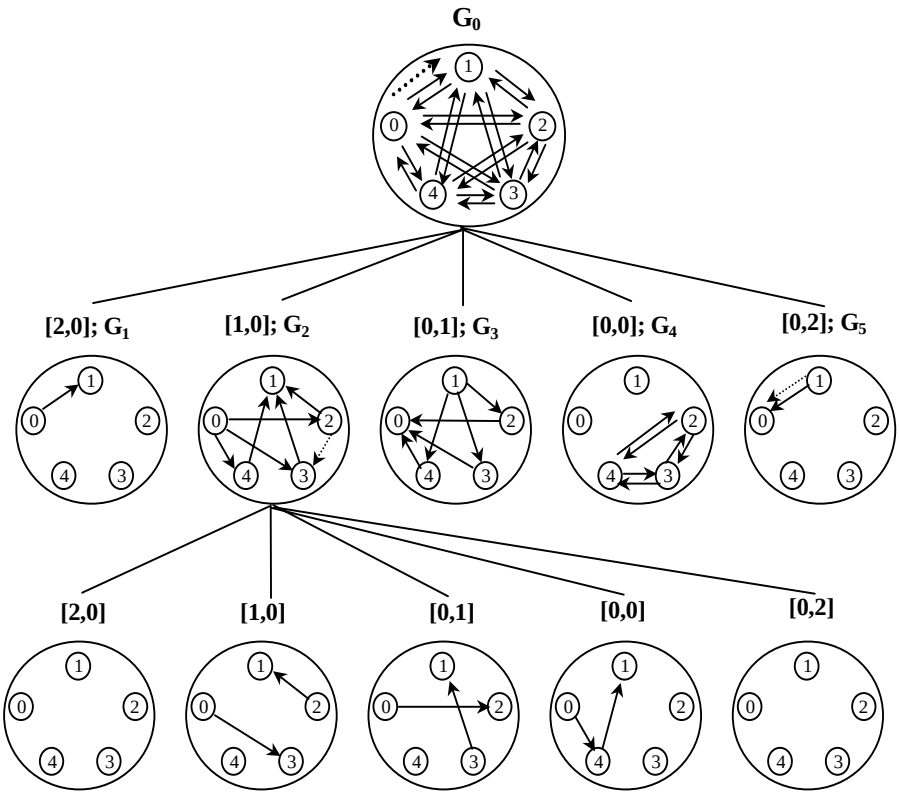


Fig. 3. A graphic model to represent the results after the first guess and one of second guesses for 2 × 5 AB game, where (0, 1) and (2, 3) are the partition edges in G_0 and G_2, respectively.

Any moment in the game is expressed by a node, represented by a *game graph* $G_i = <V_i, E_i>$, in the game tree. The root $G_0 = <V_0, E_0>$ of the game tree is a complete directed graph with 5 vertices and 20 edges. We map the game graphs to 2 × n AB games as follows:

Vertex: Each vertex in a game graph G_i corresponds to a *symbol* in the AB game; for example, 0, 1, 2, 3, and 4 are five symbols in the 2 × 5 AB game. Notice that we use the term “node” in the game tree and “vertex” in the game graphs.

Edge: Each directed edge in a game graph G_i refers to a possible codeword, so there are 5 × 4 = 20 edges in the root G_0 of the game tree. In the medium stages, the edges in a game graph G_i, $i \geq 1$, refer to the *remaining candidates*, i.e., unidentified codewords.

Partition edge: the edge codebreaker chooses to partition the game graph; for instance, the dashed arrow (0, 1) shown in G_0 of Fig. 3 refers to the first guess (0, 1) in a 2 × 5 AB game. Notice that, according the rules of this game, we can choose arbitrary partition edge $e \in E_0$ (not necessarily $e \in E_i$) in the medium stage, for example, (2, 3) can be chosen as the partition edge in G_2.

Each *class*, represented as a node in the game tree, in level 2 refers to one response [A, B] after guessing (0, 1) in the game. The five classes in level 2, i.e., [2, 0], [1, 0], [0, 1], [0, 0], and [0, 2], partition the set of 20 edges in the complete directed graph G_0 in the root node. Moreover, only one edge (0, 1) remains in G_1, which refers to class [2, 0]. It means that the edge (0, 1) was *hit*. On the other hand, if there is only one edge in a class, except for the class [2, 0], then this edge is said to be *identified*, and one more guess is required to hit it. For example, there is only one edge (1, 0) in G_5. We need one more guess (1, 0) to finish the AB game.

3.2 The Partition Principles

Now we will present how to partition the edges on the graph. Remember that a game graph G_i is a directed graph. If the partition edge is (j, k), then vertex j is called the *origin vertex*, and vertex k is called the *destination vertex*. We can partition all the edges in the game graph G_i according to the following simple rules:

(1) The outgoing edges (j, m) from the origin vertex j and the incoming edges (m, k) to the destination vertex k are classified as [1, 0], where $m \neq k$, $m \neq j$.
(2) The outgoing edges (k, m) from the destination vertex k and the incoming edges (m, j) to the origin vertex j are classified as [0, 1] , where $m \neq j$, $m \neq k$.
(3) The edges that are not adjacent to the origin and destination vertices are classified as [0, 0].
(4) The edge (j, k) that is both an outgoing edge from the origin vertex j and an incoming edge to the destination vertex k is classified as [2, 0].
(5) The edge (k, j) that is both an outgoing edge from the destination vertex k and an incoming edge to the origin vertex j is classified as [0, 2].

As depicted in Fig. 3, at the initial stage, the partition edge (0, 1) in G_0 partitions the 20 edges into five classes. The outgoing edges from the origin vertex 0, i.e., (0, 2), (0, 3), and (0, 4), and incoming edges to the destination vertex 1, i.e., (2, 1), (3, 1), and (4, 1), are classified into class [1, 0], i.e., G_2. The edge (0, 1) is both an incoming edge to vertex 1 and an outgoing edge from vertex 0, so it is classified into class [2, 0], i.e., G_1. The other classes can be obtained in the same way.

3.3 A Strategy for the 2 × 5 AB Game

Now we will describe the goals in this paper and show how we can achieve them using the graphic model. By means of the partition rules given above, we can translate the game-guessing process into a sequence of graph partition and tree traversal procedures. Notice that all the leaves in the game tree are “hits nodes”; i.e., one candidate

is *hit* in each of the leaf nodes. Therefore, we can directly apply the results described in Observations 1 and 2 to obtain the optimal strategy for 2 × n AB games. Our goals are, thus, to minimize the height H of the game tree for the worst case and to minimize the external path length L of the game tree in the expected case. The key to achieving these goals is simply to choose the best partition edge to partition the remaining edges at each stage, like playing the real game.

In Lemma 2, we show how to calculate the total number of guesses required to hit 2 remaining candidates (or 2 remaining edges) in a class (or a game graph). This lemma can be applied to m × n AB games with arbitrary m, n.

Lemma 2 If a game graph that is the root node of a game tree contains only two remaining edges, then the minimum possible values for the external path length L and the height H of the game tree are 3 and 2, respectively.

Proof: *Sufficient:* We can choose one of the two remaining edges as the partition edge. Then, this edge will be hit, and the other edge will be identified and one more guess is required. Therefore, a possible value for the external path length L of the game tree is 1 + 2 = 3, and that for the height H of the game tree is 2.

Necessary: For the situation where two edges remain in a game graph, there are only three possibilities for choosing the partition edge.

Case 1. Choose one of these two remaining edges as the partition edge. As described in the sufficient condition, the external path length L of the game tree is 1 + 2 = 3, and the height H of the game tree is 2.
Case 2. Choose an edge adjacent to at least one of the two remaining edges. The best result of these guesses is able to identify the two remaining edges simultaneously, each of which requires one more guess to be hit. So the external path length L of the game tree is 2 + 2 = 4, and the height H of the game tree is 2.
Case 3. Choose an edge that is not adjacent to the two remaining edges. The result of this guess makes no contribution to further guesses since the game graph after the guess is the same as the one before the guess. Thus, we can omit this possibility.

Therefore, to hit 2 remaining edges in a game graph (or a class), the external path length L of the game tree must be at least 3, and the height H of the game tree must be at least 2. In other words, the total number of guesses required to hit the two remaining candidates is at least 3. Hence, the number of guesses in the expected case is 3/2 = 1.5. In addition, the number of guesses in the worst case is at least 2. ❑

Now we will present a strategy for playing the 2 × 5 AB game on the graphic model and show how to calculate the external path length L and the height H of the game tree. In this way, we can develop sophisticated strategies for higher dimension games. To demonstrate the variety of strategies, the strategies used in the following examples are not necessarily optimal choice. Observing the first guess in Fig. 3, since G_0 is a symmetric and complete graph, choosing any edge as the partition edge will obtain the same result. We choose (0, 1) as the partition edge. After the first guess, there is only one remaining edge (0, 1) in G_1 (class [2, 0]). Notice that Fig. 3 shows only one leaf G_1, and the distance from the root G_0 to G_1 is 1. Therefore, edge (0, 1) only requires one guess to

be hit. Although the edge (1, 0) is also the only edge in G_5 (class [0, 2]), one more guess is required to "hit" the edge (1, 0), so it requires 2 guesses. Furthermore, it is easy to show that the game graphs G_2 and G_3 for classes [1, 0] and [0, 1] are *isomorphic*; that is, if we exchange vertex 0 and vertex 1 in the game graph G_2 for class [1, 0], then it will be equivalent to the game graph G_3 for class [0, 1]. Intuitively, they have the same number of guesses in both the worst case and the expected case. Therefore, we will concentrate on classes [1, 0] and [0, 0]. In the following paragraphs, we describe the general procedures used to calculate the number of guesses for the 2 × 5 AB game in the worst and expected cases.

In G_2 of Fig. 3, the partition edge (2, 3) partitions the remaining six edges into three nonempty classes, [1, 0], [0, 1], and [0, 0], each of which has two remaining edges. By Lemma 2, the two remaining edges can be hit in one and two more guesses, respectively. Therefore, the external path length for G_2, $L_2 = 2 + 3 + 2 + 3 + 2 + 3 = 15$; hence, by Observation 2, the expected number of guesses for G_2 is $L_2/6 = 2.5$.

Now we will consider how to calculate *the total number of guesses* (or the external path length L_4) for the 2 × 3 AB game in the expected case. In G_4 shown in Fig. 4, there is a complete subgraph with three vertices and six directed edges, which represents the 2 × 3 AB game. If (4, 2) is chosen as the partition edge, as shown in Fig. 4, we need 1, 2, 3, 2, 3, and 2 guesses to hit edges (4, 2), (4, 3), (3, 2), (3, 4), (2, 3), and (2, 4), respectively.

Therefore, the external path length L_4 for the node G_4 is $L_4 = 1 + 2 + 3 + 2 + 3 + 2 = 13$; thus, the expected number of guesses for the game graph G_4 is $L_4/6 = 13/6$.

Finally, we combine Figs. 3 and 4. If we choose (2, 3) and (4, 2) as the partition edges in G_2 and G_4, respectively, then the height of the game tree is 4; that is, by using our analysis technique the number of guesses in the worst case is 4. The total number of guesses L_0 for G_0 can be computed as follows: $L_0 = (L_1 + L_2 + L_3 + L_4 + L_5)$ + (the total number of edges in G_0) = $(L_1 + 1) + (L_2 + 6) + (L_3 + 6) + (L_4 + 6) + (L_5 + 1) = (0 + 1) + (15 + 6) + (15 + 6) + (13 + 6) + (1 + 1) = 64$, where $L_1 + 1$ is for G_1, $L_2 + 6$ is for G_2, and so on. Therefore, the expected number of guesses for $G_0 = 64/20 = 3.2$.

4. AN OPTIMUM STRATEGY FOR 2 × n AB GAMES

In this section, the graphic model described in section 3 is used to develop an optimum strategy for 2 × n AB games. We simplify the graphic representation for the game and define three types of subgraphs, denoted by $G_a(k)$, $G_b(k)$, and $G_c(k)$ in Figs. 5 (b), 5 (c), and 5 (d), respectively. The rectangle shown in Fig. 5 (a) and the ellipse shown in Fig. 5 (b), denoted by "i∼j" inside, refer to a graph with only $j - i + 1$ separate vertices named i, i + 1, …, j, and a complete directed graph with $j - i + 1$ vertices named i, i + 1, …, j, or "all to one" edges, each of which connects the vertex outside the rectangle to all the vertices inside the rectangle. Hence, there are $n - 1$ and $2(n - 2)$ edges in Figs. 5 (c) and 5 (d), respectively.

Definition 4.1 $T(k)$, $T_1(k)$, and $T_2(k)$ are *the minimal external path lengths* of the game trees whose roots are the nodes for $G_a(k)$, $G_b(k)$, and $G_c(k)$, respectively. In a similar way, we define $H(k)$, $H_1(k)$, and $H_2(k)$ as *the minimal possible values for the height of the game trees* for $G_a(k)$, $G_b(k)$, and $G_c(k)$, respectively.

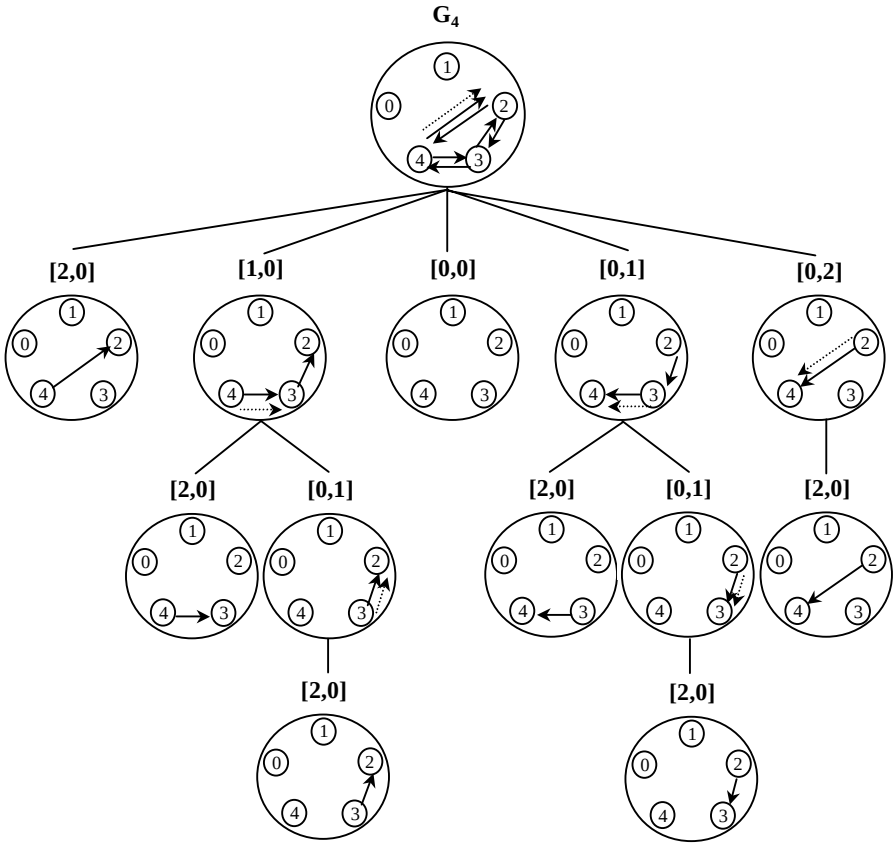


Fig. 4. The game tree for G_4, which represents the 2 × 3 AB game. The total number of guesses for the six edges is 13; the expected number of guesses for the graph is 13/6.

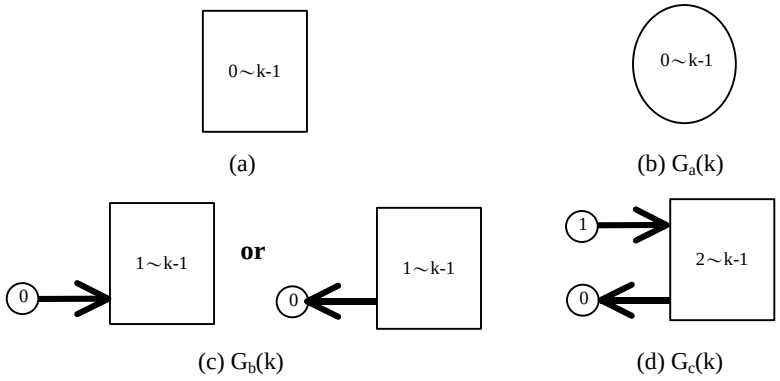


Fig. 5. (a) A graph with k vertices and no edges. (b) A complete directed graph, $G_a(k)$, with k vertices and k(k – 1) edges. (c) Two types of graphs, $G_b(k)$s, with k – 1 edges. (d) A graph, $G_c(k)$, with 2(k – 2) edges.

Lemma 3 The minimal number of guesses required for 2 × n AB games is T(n)/n(n – 1) for the expected case and H(n) for the worst case.

Proof: Since the initial state for 2 × n AB games is $G_a(n)$, by Definition 4.1, *the minimal external path length* and *the minimal height of the game tree* are T(n) and H(n), respectively. From Observations 1 and 2, the results of this property follow. ❑

Now we will demonstrate our procedure for deriving T(n) and H(n), by which we can obtain the optimal strategies for 2 × n AB games in the expected and worst cases. At the initial state, since the graph $G_a(n)$ is a symmetric and complete graph, we can choose any edge, (0, 1) in this example, as the partition edge; consequently, we will obtain the same result. The game tree after the first guess is shown in Fig. 6. The minimal numbers of further guesses required for classes [2, 0], [1, 0], [0, 1], [0, 0], and [0, 2] (the external path lengths of the subtrees whose roots are the nodes for classes [2, 0], [1, 0], [0, 1], [0, 0], and [0, 2]) are 0, $T_2(n)$, $T_2(n)$, T(n − 2), and 1, respectively. In addition, since the number of guesses for each remaining candidate (the length from root to each leaf) will be increased by one after the first guess, we have to add n(n − 1) to compute T(n), where n(n − 1) is the number of edges before the guess(i.e., in the root node). Hence, $T(n) = 0 + T_2(n) + T_2(n) + T(n-2) + 1 + n(n-1) = T(n-2) + 2T_2(n) + n^2 - n + 1$.

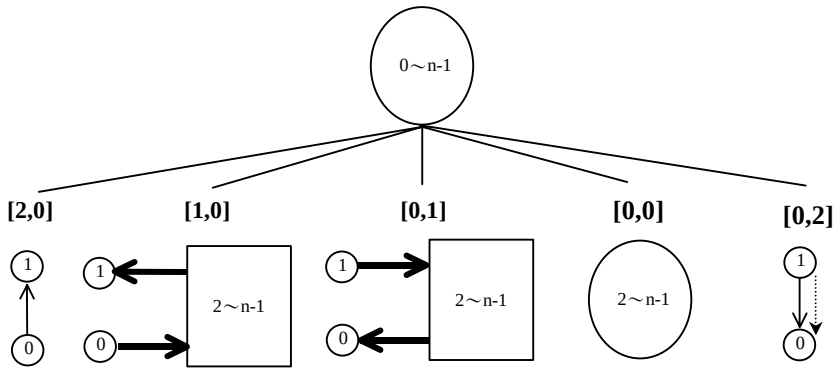


Fig. 6. The game tree for 2 × n AB games, where (0, 1) is chosen as the partition edge. $T(n) = 0 + T_2(n) + T_2(n) + T(n-2) + 1 + n(n-1) = T(n-2) + 2T_2(n) + n(n-1) + 1$.

In Fig. 6, since the game graphs for classes [1, 0] and [0, 1] are isomorphic, and since the game graph for class [0, 0] can be solved recursively, we only consider the class [1, 0]. The four possible ways to partition the class [1, 0], i.e., $G_c(n)$, and their recurrences are described as follows:

1. Choose (0, y) or (y, 1) as the partition edge, where y ∈ {2, 3, 4, …, n – 1}. The game tree after the first guess is shown in Fig. 7, where we choose (0, 2) as the partition edge. Now, the numbers of further guesses required for the classes [2, 0], [1, 0], [0, 1], and [0, 0] are 0, $T_1(n-2)$, 1, and $T_1(n-2)$, respectively. In addition, we have to add 2(n – 2) to compute $T_2(n)$, where 2(n – 2) is the number of edges before the guess (0, 2). Therefore, $T_2(n) \le 0 + T_1(n-2) + 1 + T_1(n-2) + 2(n-2) = 2T_1(n-2) + 2n - 3$.

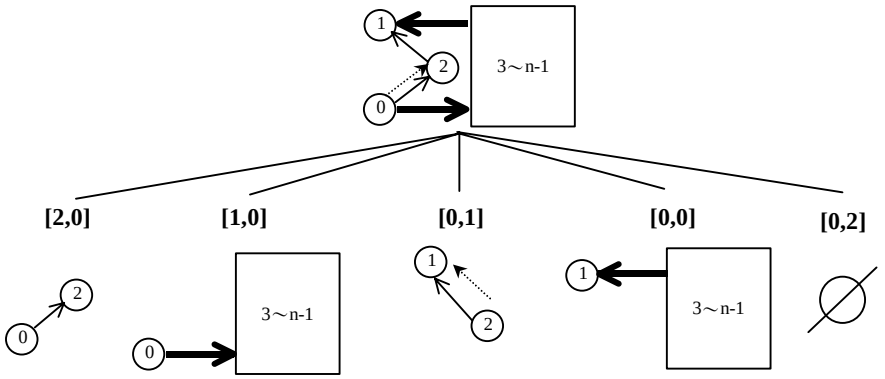


Fig. 7. The game tree for graph $G_c(n)$, where (0, 2) is chosen as the partition edge. The notation "∅" in class [0, 2] refers to an empty set. $T_2(n) \le 0 + T_1(n-2) + 1 + T_1(n-2) + 2(n-2) = 2\,T_1(n-2) + 2n - 3$.

2. Choose (y, 0) or (1, y) as the partition edge, where $y \in \{2, 3, 4, \ldots, n-1\}$. For example, in Fig. 8, we choose (2, 0) as the partition edge. The numbers of further guesses required for the classes [2, 0], [1, 0], [0, 1], [0, 0], and [0, 2] are 0, 1, $T_1(n-2)$, $T_1(n-2)$, and 1, respectively. Therefore, $T_2(n) \le 1 + T_1(n-2) + T_1(n-2) + 1 + 2(n-2) = 2\,T_1(n-2) + 2n - 2$. Choosing (1, 2) as the partition edge will lead to the same result according to similar analysis, so we omit it here.

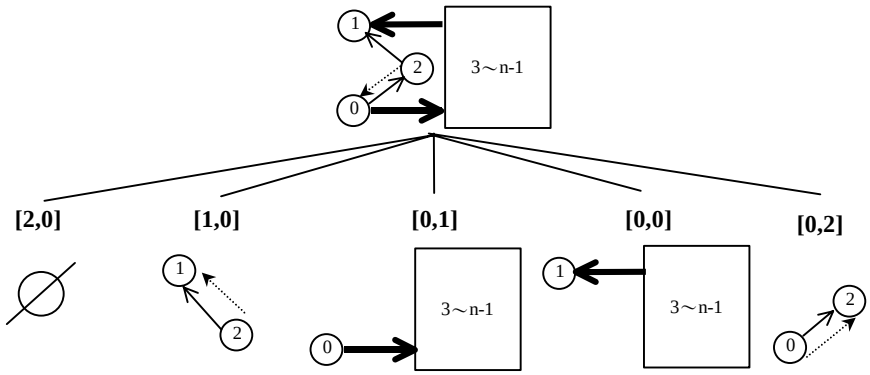


Fig. 8. The game tree for graph $G_c(n)$, where (2, 0) is chosen as the partition edge. $T_2(n) \le 0 + 1 + T_1(n-2) + T_1(n-2) + 1 + 2(n-2) = 2\,T_1(n-2) + 2n - 2$.

3. Choose (y_1, y_2) as the partition edge, where $y_1, y_2 \in \{2, 3, 4, \ldots, n-1\}$ and $y_1 \ne y_2$. For example, in Fig. 9, we choose (2, 3) as the partition edge. Now, $T_2(n) \le 3 + 3 + T_2(n-2) + 2(n-2) = T_2(n-2) + 2n + 2$.

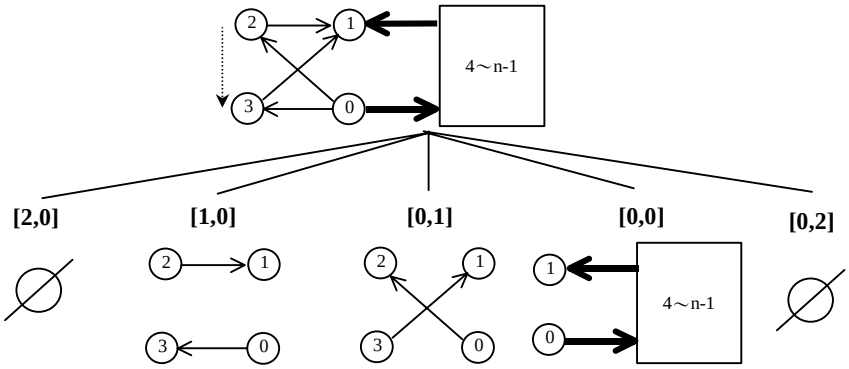


Fig. 9. The game tree for graph $G_c(n)$, where (2, 3) is chosen as the partition edge. $T_2(n) \le 3 + 3 + T_2(n-2) + 2(n-2) = T_2(n-2) + 2n + 2$.

4. Choose (0, 1) or (1, 0) as the partition edge. As shown in Fig. 10, if we choose (0, 1) as the partition edge, then there is only one nonempty class [1, 0], which contains all 2(n – 2) edges; similarly, if (1, 0) is chosen, the only nonempty class is [0, 1]. That is, we cannot derive further partition from this guess and also have to add 2(n – 2) to compute $T_2(n)$. Therefore, $T_2(n) \le T_2(n) + 2(n-2)$.

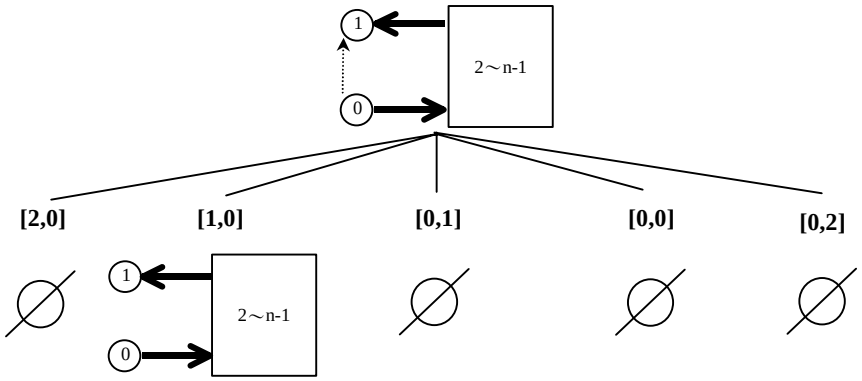


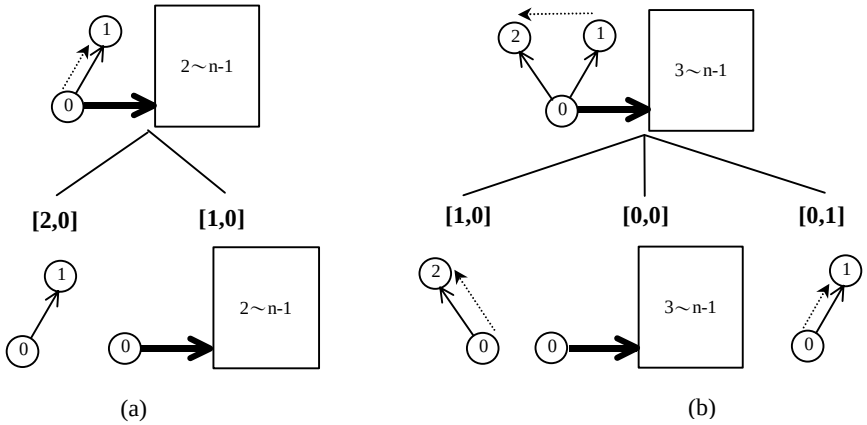
Fig. 10. The game tree for graph $G_c(n)$, where (0, 1) is chosen as the partition edge. $T_2(n) \le T_2(n) + 2(n-2) = T_2(n) + 2n - 4$.

Observing the above recurrences, the total number of guesses for strategy 2 is always one more than that for strategy 1. In addition, we cannot derive further partition by using strategy 4. Therefore, we can ignore strategies 2 and 4 here. Now, we can simply investigate strategies 1 and 3 to determine which one is the better. That is, we can determine the optimal strategy for $T_2(n) = \text{Min}(2T_1(n-2) + 2n - 3, T_2(n-2) + 2n + 2)$.

Now we will focus on the graph $G_b(n)$. There are two types of graphs, between which only the edge direction is different, as shown in Fig. 5 (c). Hence, we can obtain the same numbers of further guesses for these two types of graphs by changing the direction of the partition edge. Therefore, without loss of generality, we only need to

consider one type of graph. Now we will describe the three possible ways to partition $G_b(n)$ and their recurrences as follows:

(i) Choose (0, y) as the partition edge, where y ∈ {1, 2, 3, …, n − 1}. In Fig. 11 (a), we choose (0, 1) as the partition edge, which partitions the graph $G_b(n)$ into two nonempty classes, [2, 0] and [1, 0]. The numbers of further guesses for these two classes are 0 and $T_1(n-1)$, respectively. In addition, there are n − 1 edges in the graph $G_b(n)$. Therefore, $T_1(n) \le 0 + T_1(n-1) + n - 1 = T_1(n-1) + n - 1$.



(a) (b)

Fig. 11. (a) Strategy (i), where (0, 1) is the partition edge. $T_1(n) \le 0 + T_1(n-1) + n - 1$. (b) Strategy (iii), where (1, 2) is the partition edge. $T_1(n) \le 1 + T_1(n-2) + 1 + n - 1$.

(ii) Choose (y, 0) as the partition edge, where y ∈ {1, 2, 3, …, n − 1}. We choose (1,0) as the partition edge, which partition the graph $G_b(n)$ into two nonempty classes, [0, 2] and [1, 0]. The numbers of further guesses for these two classes are 1 and $T_1(n-1)$, respectively. Therefore, $T_1(n) \le 1 + T_1(n-1) + n - 1 = T_1(n-1) + n$.

(iii) Choose (y_1, y_2) as the partition edge, where $y_1, y_2 \in \{1, 2, 3, \ldots, n-1\}$ and $y_1 \neq y_2$. For example, we choose edge (1, 2) as the partition edge, as shown in Fig. 11(b). Now, $T_1(n) \le 1 + T_1(n-2) + 1 + n - 1 = T_1(n-2) + n + 1$.

From the above analysis, we have $T_1(n) = \text{Min}(T_1(n-2) + n + 1, T_1(n-1) + n, T_1(n-1) + n - 1) = \text{Min}(T_1(n-2) + n + 1, T_1(n-1) + n - 1)$.

To minimize the total number of guesses, we list the recurrences developed above and their solutions in Table 1. Theorem 3 demonstrates the minimum number of guesses required for 2 × n AB games in the expected case.

Theorem 3 The minimum number of guesses required for 2 × n AB games in the expected case is $(4n^3 + 21n^2 - 76n + 72)/12n(n-1)$ if n is even, and it is $(4n^3 + 21n^2 - 82n + 105)/12\ n(n-1)$ if n is odd.

Proof: By induction, the recurrences $T(n)$, $T_2(n)$, *and* $T_1(n)$ listed in Table 1 can be solved as collapsing sums, whose detailed proofs are shown in the Appendix. ❑

Table 1. The recurrences and their solutions for T(n), $T_2(n)$, and $T_1(n)$.

Functions	Recurrence relations	Solutions
T(n) =	$\begin{cases} 3, & \text{if } n = 2 \\ 13, & \text{if } n = 3 \\ T(n-2)+2T_2(n)+n^2-n+1, & \text{otherwise} \end{cases}$	$\begin{cases} (4n^3+21n^2-76n+72)/12, & \text{if n is even} \\ (4n^3+21n^2-82n+105)/12, & \text{if n is odd} \end{cases}$
$T_2(n)$ =	$\begin{cases} 3, & \text{if } n = 3 \\ 7, & \text{if } n = 4 \\ \text{Min}(2T_1(n-2)+2n-3, T_2(n-2)+2n+2), & \text{otherwise} \end{cases}$	$\begin{cases} n^2+4n-18)/2, & \text{if n is even} \\ (n^2+4n-19)/2, & \text{if n is odd} \end{cases}$
$T_1(n)$ =	$\begin{cases} 1, & \text{if } n = 2 \\ 3, & \text{if } n = 3 \\ \text{Min}(T_1(n-2)+n+1, T_1(n-1)+n-1), & \text{otherwise} \end{cases}$	$\begin{cases} n^2+4n-8)/4, & \text{if n is even} \\ (n^2+4n-9)/4, & \text{if n is odd} \end{cases}$

To determine the number of guesses for 2 × n AB games in the worst case, we consider the height instead of the external path length of the game tree. We can obtain the recurrences for H(n), $H_1(n)$, and $H_2(n)$ by slightly modifying the recurrences T(n), $T_1(n)$, and $T_2(n)$ shown in Table 1. Observing Fig. 6, the height of the game tree will be 1 plus the height of the highest among the five subtrees whose roots are the nodes for the five classes. Therefore, we have H(n) = Max(0, $H_2(n)$, $H_2(n)$, H(n − 2), 1) + 1 = Max($H_2(n)$, H(n − 2)) + 1. We can also obtain this recurrence from the recurrence T(n) = T(n − 2) + $2T_2(n)$ + n^2 − n + 1. That is, we can change coefficient 2 associated with the recurrence function $T_2(n)$ to 1, the cost (n^2 − n + 1) for each iteration into 1, and the *sum operations* between the recurrence functions in the right side to *Max function*. The recurrences $H_1(n)$ and $H_2(n)$ can be obtained in a similar way. The recurrence relations and their solutions for H(n), $H_1(n)$, and $H_2(n)$ are shown in the Table 2. Theorem 4 demonstrates the minimum number of guesses required for 2 × n AB games in the worst case.

Table 2. The recurrences and their solutions for H(n), $H_1(n)$, and $H_2(n)$.

Functions	Recurrence relations	Solutions
H(n) =	$\begin{cases} 2, & \text{if } n = 2 \\ 3, & \text{if } n = 3 \\ \text{Max}(H(n-2), H_2(n))+1, & \text{otherwise} \end{cases}$	$\lceil n/2 \rceil + 1$
$H_2(n)$ =	$\begin{cases} 2, & \text{if } n = 3 \\ 2, & \text{if } n = 4 \\ \text{Min}(H_1(n-2), H_2(n-2))+1, & \text{otherwise} \end{cases}$	$\lceil n/2 \rceil$
$H_1(n)$ =	$\begin{cases} 1, & \text{if } n = 2 \\ 2, & \text{if } n = 3 \\ \text{Min}(H_1(n-2), H_1(n-1))+1, & \text{otherwise} \end{cases}$	$\lceil n/2 \rceil$

Theorem 4 $\lceil n/2 \rceil + 1$ guesses are necessary and sufficient for 2 × n AB games in the worst case.

Proof: By induction, the recurrences H(n), $H_1(n)$, and $H_2(n)$ listed in Table 2 can be solved as collapsing sums, whose detailed proofs are shown in the Appendix. ❑

We apply the optimum strategy obtained in this section to the 2 × 5 AB game. The height of the game tree for the strategy used in section 2 for the 2 × 5 AB game is 4, which has already achieved the best result, $H(n) = \lceil n/2 \rceil + 1 = \lceil 5/2 \rceil + 1 = 4$, in Theorem 4. Nevertheless, the strategy achieved in G_2 of Fig. 3, i.e., choosing (2, 3) as the partition edge, is not the best choice if we consider the external path length. As proven in Theorem 3, if we use the best strategy, i.e., choosing (0, y) or (y, 1) as the partition edge, where $y \in \{2, 3, 4\}$, then the external path length L_2 of the game tree will be 13 instead of 15. Thus, the total number of guesses required for the 2 × 5 AB game is $L_0 = (L_1 + L_2 + L_3 + L_4 + L_5)$ + (the total number of edge in G_0) = (0 + 13 + 13 + 13 + 1) + 20 = 60. Therefore, the minimum number of guesses required for $G_0 = 60/20 = 3$, which achieves the result, $(4n^3 + 21n^2 - 82n + 105)/12n(n - 1) = (4 * 5^3 + 21 * 5^2 - 82 * 5 + 105)/(12 * 5 * 4) = 3$, in Theorem 3.

5. CONCLUDING REMARKS

In this paper we have presented a systematic methodology to obtain optimal strategies for 2 × n AB games in both the worst and expected cases. We have taken advantage of the properties of game trees, such as their height and external path length, to calculate the numbers of guesses required in the worst and expected cases for a deductive game. Furthermore, we have invented a graphic model to represent 2 × n AB games, and to discover some recursive and isomorphic properties in the graphic model. Using this approach, we can reduce the search space and obtain the optimal results in both the worst and expected cases. We summarize this paper's results for $2 \le n \le 12$ in Table 3. We hope that the methodologies proposed in this paper will prompt researchers to study other related problems.

Table 3. The minimum number of guesses required for 2 × n AB games.

n	Minimum number of guesses	
	Worst case	Expected case
	$\lceil n/2 \rceil + 1$	$(4n^3 + 21n^2 - 76n + 72)/12n(n - 1)$ if n is even, $(4n^3 + 21n^2 - 82n + 105)/12n(n - 1)$ if n is odd
2	2	1.5
3	3	2.166666667
4	3	2.5
5	4	3
6	4	3.433333333
7	5	3.833333333
8	5	4.25
9	6	4.611111111
10	6	5.011111111
11	7	5.354545455
12	7	5.742424242

ACKNOWLEDGEMENT

We thank all reviewers for their valuable comments and suggestions. This research was supported in part by a grant NSC92-2213-E-003-006 from National Science Council, R.O.C. Our gratitude also goes to the Academic Paper Editing Clinic, National Taiwan Normal University.

REFERENCES

1. D. E. Knuth, "The computer as mastermind," *Journal of Recreational Mathematics*, Vol. 9, 1976, pp. 1-6.
2. R. W. Irving, "Towards an optimum mastermind strategy," *Journal of Recreational Mathematics*, Vol. 11, 1978-79, pp. 81-87.
3. E. Neuwirth, "Some strategies for mastermind," *Zeitschrift fur Operations Research*, Vol. 26, 1982, pp. 257-278.
4. K. Koyama and T. W. Lai, "An optimal mastermind strategy," *Journal of Recreational Mathematics*, Vol. 25, 1993, pp. 251-256.
5. M. M. Flood, "Sequential search strategies with mastermind variants – Part 1," *Journal of Recreational Mathematics*, Vol. 20, 1988, pp. 105-126.
6. K. I. Ko and S. C. Teng, "On the number of queries necessary to identify a permutation," *Journal of Algorithms*, Vol. 7, 1986, pp. 449-462.
7. Z. X. Chen and C. Cunha, "Finding a hidden code by asking questions," in *Proceedings of the International Computing and Combinatorics Conference* (*COCOON '96*), 1996, pp. 50-55.
8. L. Bento, L. Pereira, and A. Rosa, "Mastermind by evolutionary algorithms," in *Proceedings of the International Symposium on Applied Computing*, 1996, pp. 307-311.
9. J. L. Bernier, C. I. Herraiz, J. J. Merelo, S. Olmeda, and A. Prieto, "Solving mastermind using gas and simulated annealing: a case of dynamic constraint optimization," in *Proceedings of Parallel Problem Solving from Nature IV* (*PPSN IV*), 1996, pp. 554-563.
10. J. J. Merelo, J. Carpio, P. Castillo, V. M. Rivas, and G. Romero (GeNeura Team), "Finding a needle in a haystack using hints and evolutionary computation: the case of genetic mastermind," in *Proceedings of the Genetic and Evolutionary Computation Conference late breaking papers*, 1999, pp. 184-192.
11. J. R. Roche, "The value of adaptive questions in generalized mastermind," in *Proceedings of IEEE on the International Symposium on Information Theory*, 1997, pp. 135-135.
12. G. Kabatianski and V. Lebedev, "The mastermind game and the rigidity of the hamming space," in *Proceedings of IEEE on the International Symposium on Information Theory*, 2000, pp. 375-375.
13. R. Sedgewick, *Algorithms*, 2nd edition, Addison-Wesley, 1988.
14. D. E. Knuth, "Sorting and searching," *The Art of Computer Programming*, Vol. 3, 2nd edition, Addison Wesley, 1998.
15. S. T. Chen, S. H. Hsu, and S. S. Lin, "An optimal strategy for 2 × n AB games," in

Proceedings of the 2002 International Computer Symposium on Algorithms and Computational Molecular Biology, 2002, pp. C2-2.

APPENDIX: THE PROOFS OF THEOREMS 3 AND 4

Theorem 3 The minimum number of guesses required for 2 × n AB games in the expected case is $(4n^3 + 21n^2 - 76n + 72)/12n(n-1)$ if n is even, and it is $(4n^3 + 21n^2 - 82n + 105)/12\ n(n-1)$ if n is odd.

Proof: We begin by solving the recurrences $T(n)$, $T_2(n)$, *and* $T_1(n)$ listed in Table 1.

$$1.\ T_1(n) = \begin{cases} 1, & \text{if } n = 2, \\ 3, & \text{if } n = 3, \\ \text{Min}(T_1(n-2)+n+1, T_1(n-1)+n-1), & \text{otherwise.} \end{cases}$$

First, we shall prove by induction that

$T_1(n) = T_1(n-2) + n + 1$
(i.e., $\text{Min}(T_1(n-2) + n + 1, T_1(n-1) + n - 1) = T_1(n-2) + n + 1$).

Basis step:

$T_1(4) = \text{Min}(T_1(4-2) + 4 + 1, T_1(4-1) + 4 - 1) = \text{Min}(6, 6)$, and
$T_1(5) = \text{Min}(T_1(5-2) + 5 + 1, T_1(5-1) + 5 - 1) = \text{Min}(3 + 5 + 1, 6 + 5 - 1)$
$= \text{Min}(9, 10)$,
so $T_1(n) = T_1(n-2) + n + 1$ is true for n = 4, 5.

Inductive step:
Assume that $T_1(k) = T_1(k-2) + k + 1$ is true for $4 \le k < n$.
We want to prove that $T_1(n) = T_1(n-2) + n + 1$ is also true.

$$\begin{aligned} T_1(n) &= \text{Min}(T_1(n-2)+n+1, T_1(n-1)+n-1) \\ &= \text{Min}(T_1(n-4)+(n-1)+(n+1), T_1(n-3)+n+(n-1)) \\ &= \text{Min}(T_1(n-6)+(n-3)+(n-1)+(n+1), T_1(n-5)+(n-2)+n+(n-1)) \\ &= \begin{cases} \text{Min}(T_1(2)+5+7+\ldots+(n+1), T_1(3)+6+8+\ldots+n+(n-1)), & \text{if n is even,} \\ \text{Min}(T_1(3)+6+8+\ldots+(n+1), T_1(2)+5+7+\ldots+n+(n-1)), & \text{if n is odd.} \end{cases} \\ &= \begin{cases} \text{Min}((n^2+4n-8)/4, (n^2+6n-16)/4), & \text{if n is even,} \\ \text{Min}((n^2+6n-9)/4, (n^2+6n-15)/4), & \text{if n is odd.} \end{cases} \end{aligned}$$

Since $(n^2 + 4n - 8)/4 \le (n^2 + 6n - 16)/4$ and $(n^2 + 4n - 9)/4 \le (n^2 + 6n - 15)/4$ for $n \ge 4$, we conclude that $T_1(n) = T_1(n-2) + n + 1$ is true, and that

$$T_1(n) = \begin{cases} (n^2 + 4n - 8)/4, & \text{if n is even,} \\ (n^2 + 4n - 9)/4, & \text{if n is odd.} \end{cases}$$

2. $$T_2(n) = \begin{cases} 3, & \text{if n = 3,} \\ 7, & \text{if n = 4,} \\ \text{Min}(2T_1(n-2) + 2n - 3, T_2(n-2) + 2n + 2), & \text{otherwise.} \end{cases}$$

Now, we shall prove by induction that $T_2(n) = 2T_1(n - 2) + 2n - 3$.

Basis step:

$T_2(5) = \text{Min}(2T_1(3) + 10 - 3, T_2(3) + 10 + 2) = \text{Min}(6 + 10 - 3, 3 + 10 + 2)$
$= \text{Min}(13, 15)$, and
$T_2(6) = \text{Min}(2T_1(4) + 12 - 3, T_2(4) + 12 + 2) = \text{Min}(12 + 12 - 3, 7 + 12 + 2)$
$= \text{Min}(21, 21)$,
so $T_2(n) = 2T_1(n - 2) + 2n - 3$ is true for n = 5, 6.

Inductive step:

Assume that $T_2(k) = 2T_1(k - 2) + 2k - 3$ is true for $5 \le k < n$.
We want to prove that $T_2(n) = 2T_1(n - 2) + 2n - 3$ is also true.

$$\begin{aligned} T_2(n) &= \text{Min}(2T_1(n-2) + 2n - 3, T_2(n-2) + 2n + 2) \\ &= \text{Min}(2T_1(n-2) + 2n - 3, 2T_1(n-4) + 2(n-2) - 3 + (2n+2)) \\ &= \text{Min}(2T_1(n-2) + 2n - 3, 2T_1(n-4) + 4n - 5) \\ &= \begin{cases} \text{Min}(2((n-2)^2 + 4(n-2) - 8)/4 + 2n - 3, 2((n-4)^2 + 4(n-4) - 8)/4 + 4n - 5)), & \text{if n is even,} \\ \text{Min}(2((n-2)^2 + 4(n-2) - 9)/4 + 2n - 3, 2((n-4)^2 + 4(n-4) - 9)/4 + 4n - 5)), & \text{if n is odd.} \end{cases} \\ &= \begin{cases} \text{Min}((n^2 + 4n - 18)/2, (n^2 + 4n - 18)/2), & \text{if n is even,} \\ \text{Min}((n^2 + 4n - 19)/2, (n^2 + 4n - 19)/2), & \text{if n is odd.} \end{cases} \end{aligned}$$

Therefore, we conclude that $T_2(n) = 2T_1(n - 2) + 2n - 3$ is true, and that

$$T_2(n) = \begin{cases} (n^2 + 4n - 18)/2, & \text{if n is even,} \\ (n^2 + 4n - 19)/2, & \text{if n is odd.} \end{cases}$$

3. $$T(n) = \begin{cases} 3, & \text{if n = 2,} \\ 13, & \text{if n = 3,} \\ T(n-2) + 2T_2(n) + n^2 - n + 1, & \text{otherwise.} \end{cases}$$

If n > 3 is even, then

$$\begin{aligned} T(n) &= T(n-2) + 2T_2(n) + n^2 - n + 1 \\ &= T(n-2) + 2(n^2 + 4n - 18)/2 + n^2 - n + 1 \\ &= T(n-2) + 2n^2 + 3n - 17 \end{aligned}$$

$$= T(2) + \sum_{i=0}^{(n-4)/2} [2(n-2i)^2 + 3(n-2i) - 17]$$
$$= T(2) + \sum_{i=0}^{(n-4)/2} [2n^2 + 3n - 17 + 8i^2 - 8ni - 6i]$$
$$= 3 + [(n-4)/2 + 1](2n^2 + 3n - 17) + \sum_{i=0}^{(n-4)/2} [8i^2 - 8ni - 6i]$$
$$= 3 + [(n-4)/2 + 1](2n^2 + 3n - 17) + 8\sum_{i=0}^{(n-4)/2} i^2 - (8n+6)\sum_{i=0}^{(n-4)/2} i$$
$$= 3 + [(n-4)/2 + 1](2n^2 + 3n - 17) + 8[(n^3 - 9n^2 + 26n - 24)/24] - (8n+6)[(n^2 - 6n + 8)/8]$$
$$= (4n^3 + 21n^2 - 76n + 72)/12$$

If n is odd, then

$$T(n) = T(n-2) + 2T_2(n) + n^2 - n + 1$$
$$= T(n-2) + 2(n^2 + 4n - 19)/2 + n^2 - n + 1$$
$$= T(n-2) + 2n^2 + 3n - 18$$
$$= T(3) + \sum_{i=0}^{(n-5)/2} [2(n-2i)^2 + 3(n-2i) - 18]$$
$$= (4n^3 + 21n^2 - 82n + 105)/12$$

Therefore,

$$T(n) = \begin{cases} (4n^3 + 21n^2 - 76 + 72)/2, & \text{if n is even,} \\ (4n^3 + 21n^2 - 82n + 105)/2, & \text{if n is odd.} \end{cases}$$

The necessary and sufficient conditions can be achieved based on the fact that our strategy always minimizes the external path length at each stage. Therefore, the minimum number of guesses required for 2 × n AB games is $(4n^3 + 21n^2 - 76n + 72)/12$ if n is even, and $(4n^3 + 21n^2 - 82n + 105)/12$ if n is odd. By Lemma 3, because the number of possible candidates of 2 × n AB games is n(n − 1), we have to divide these two numbers by n(n − 1) in the expected case. This completes the proof. ❑

Theorem 4 $\lceil n/2 \rceil + 1$ guesses are necessary and sufficient for 2 × n AB games in the worst case.

Proof: By induction, the recurrences H(n), $H_1(n)$, and $H_2(n)$ listed in Table 2 can be solved as collapsing sums, as follows.

1. Since $H_1(n-2) \le H_1(n-1)$, $H_1(n) = \text{Min}(H_1(n-2), H_1(n-1)) + 1 = H_1(n-2) + 1$ for $n > 3$.

$$
\begin{aligned}
H_1(n) &= H_1(n-2)+1 \\
&= H_1(n-4)+1+1 \\
&= \ldots \\
&= \begin{cases} H_1(2)+(n-2)/2 = (n-2)/2+1, & \text{if } n \text{ is even,} \\ H_1(3)+(n-3)/2 = (n-3)/2+2, & \text{if } n \text{ is odd.} \end{cases} \\
&= \lceil n/2 \rceil
\end{aligned}
$$

2. $H_2(n) = \text{Min}(H_1(n-2), H_2(n-2)) + 1$ for $n > 4$.

First, we shall prove by induction that

$H_2(n) = H_1(n-2) + 1$.

Basis step:

$H_2(5) = \text{Min}(H_1(3) + 1, H_2(3) + 1) = \text{Min}(3, 3)$, and
$H_2(6) = \text{Min}(H_1(4) + 1, H_2(4) + 1) = \text{Min}(2 + 1, 2 + 1) = \text{Min}(3, 3)$,
so $H_2(n) = H_1(n-2) + 1$ is true for $n = 5, 6$.

Inductive step:
Assume that $H_2(k) = H_1(k-2) + 1$ is true for $5 \le k < n$.
We want to prove that $H_2(n) = H_1(n-2) + 1$ is also true.

$$
\begin{aligned}
H_2(n) &= \text{Min}(H_1(n-2), H_2(n-2)) + 1 \\
&= \text{Min}(\lceil n/2 \rceil/2, H_1(n-4) + 1) + 1 \\
&= \text{Min}(\lceil n/2 \rceil - 1, \lceil n/2 \rceil - 1) + 1.
\end{aligned}
$$

Therefore, $H_2(n) = H_1(n-2) + 1$ is true and

$H_2(n) = \lceil n/2 \rceil$.

3. $H(n) = \text{Max}(H(n-2), H_2(n)) + 1$ for $n > 3$.

Now, we shall prove by induction that

$H(n) = H_2(n) + 1$.

Basis step:

$H(4) = \text{Max}(H(2), H_2(4)) + 1 = \text{Max}(2, 2) + 1$, and
$H(5) = \text{Max}(H(3), H_2(5)) + 1 = \text{Max}(3, 3) + 1$,
so $H(n) = H_2(n) + 1$ is true for $n = 4, 5$.

Inductive step:
Assume that $H(k) = H_2(k) + 1$ is true for $4 \le k < n$.
We want to prove that $H(n) = H_2(n) + 1$ is also true.

$$H(n) = \text{Max}(H(n-2), H_2(n)) + 1$$
$$= \text{Max}(H_2(n-2) + 1, \lceil n/2 \rceil) + 1$$
$$= \text{Max}(\lceil n/2 \rceil, \lceil n/2 \rceil) + 1.$$

Therefore, $H(n) = H_2(n) + 1$ is true and

$$H(n) = \lceil n/2 \rceil + 1.$$ ❑

Shan-Tai Chen (陳善泰) was born on September 17, 1965 in Taizhong, Taiwan. He received his B.S. degree from the Department of Computer Science at Chung-Cheng Institute of Technology in 1994 and his M.S. degree from the Department of Information Science and Computer Education at National Taiwan Normal University in 1997. He was the winner of the 1997 Acer Long Term Award. From 1997 to 2000, he was an adjunct instructor at the Department of Computer Science of Chung-Cheng Institute of Technology. Now, he is pursuing Ph.D. at the Department of Information Science and Computer Education of National Taiwan Normal University. His research interests include algorithms, parallel computing, evolutionary algorithms, and game theory.

Shun-Shii Lin (林順喜) was born on June 10, 1959 in Changhwa, Taiwan, Republic of China. He received the B.S. degree in Computer Engineering from the National Chiao Tung University in 1981 and the M.S. and Ph.D. degrees in Computer Science and Information Engineering from the National Taiwan University in 1985 and 1990, respectively. From August 1986 to July 2001, he was on the faculty of the Department of Information and Computer Education of the National Taiwan Normal University. He also served as a department head of the university from August 1994 to July 1996. He was a Visiting Scholar in the Department of Computer Science, University of Illinois at Urbana-Champaign in 1993. Starting August 2001 he is a professor and also serves as the director of the Graduate Institute of Computer Science and Information Engineering of the National Taiwan Normal University, Taiwan, R.O.C. His research interests include design and analysis of algorithms, parallel processing, real-time system scheduling, and artificial intelligence. Dr. Lin is a member of the Institute of Information and Computing Machinery and the Phi Tau Phi Scholastic Honor Society. He was the winner of the 1996 and 1997 Acer Long Term Award for Outstanding M.S. Thesis Supervision, the winner of 2002 National Science Council Award for Outstanding M.S. Thesis Supervision, R.O.C. He also obtained the 1994-1998 Research Achievement Award of the National Science Council of the Republic of China and the 1999 Research Award of the National Taiwan Normal University.