Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 183 (2007) 757–766

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Computing, Artificial Intelligence and Information Management

Strategy optimization for deductive games ☆

Shan-Tai Chen [a], Shun-Shii Lin [b,*], Li-Te Huang [b], Sheng-Hsuan Hsu [c]

[a] Department of Computer Science, Chung Cheng Institute of Technology, National Defense University, Tao-Yuan, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, National Taiwan Normal University, No. 88, Sec. 4, Ting-Chow Road, Taipei 117, Taiwan, ROC
[c] Department of Computer Science, Chinese Culture University, Taipei, Taiwan, ROC

Received 7 August 2005; accepted 14 August 2006
Available online 29 November 2006

Abstract

This paper presents novel algorithms for strategy optimization for deductive games. First, a k-way-branching (KWB) algorithm, taking advantage of a clustering technique, can obtain approximate results effectively. Second, a computer-aided verification algorithm, called the Pigeonhole-principle-based backtracking (PPBB) algorithm, is developed to discover the lower bound of the number of guesses required for the games. These algorithms have been successfully applied to deductive games, Mastermind and "Bulls and Cows." Experimental results show that KWB outperforms previously published approximate strategies. Furthermore, by applying the algorithms, we derive the theorem: 7 guesses are necessary and sufficient for the "Bulls and Cows" in the worst case. These results suggest strategies for other search problems.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Algorithm; Bulls and cows; Mastermind; Pigeonhole principle; Search strategies

1. Introduction

Deductive games are *two-player* games of *imperfect information*. Player I, called the *codemaker*, chooses a secret code. Player II, called the *codebreaker*, does not know the choice player I made and has to guess the secret code. After each guess, the codebreaker gets a hint about the accuracy of the guess from the codemaker. The goal of the codebreaker is to discover the secret code, based on the hints, in the smallest number of guesses. Merelo et al. [17] formulated the problem as a combinatorial optimization problem. It bears some resemblance to other combinatorial 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.

Over the past three decades, much research has been done on this kind of games. Knuth [11] focused on two games of this kind, Mastermind and "Bulls and Cows," and demonstrated a strategy for the Mastermind game that requires at most five guesses

☆ Part of this work has appeared in the Proceedings of the 2004 International Computer Symposium, pp. 517–522, 2004, Taipei, Taiwan, ROC [5].
* Corresponding author. Tel.: +886 2 29322411; fax: +886 2 29322378.
E-mail address: linss@csie.ntnu.edu.tw (S.-S. Lin).

in the worst case and 4.478 in the expected case. Later, Irving [8] and Neuwirth [18] used sophisticated heuristic strategies to improve the bounds in the expected case to 4.369 and 4.364, respectively. Finally, Koyama and Lai [15] 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 [6,7,13]. Furthermore, in [1,2,10,17], the authors used evolutionary algorithms to solve related problems. Roche [19] analyzed the generalized Mastermind and obtained asymptotical bounds under some conditions. Kabatianski et al. [9] investigated the Mastermind game and its related applications based on coding theory. A graph-partition approach was introduced to determine the optimal strategies for various games with two-digit secret code [3,4]. More recently, Kooi analyzed four well-known strategies and proposed a novel strategy that performs better than the four previous strategies [14].

We now introduce the rules of the deductive games investigated in this paper. "Bulls and Cows" [11] is a well-known deductive game, which is very popular in England and Asia. The codemaker chooses a secret code, e.g., (s_1, s_2, s_3, s_4), consisting of 4 digits out of 10 symbols, where repetition of symbols is prohibited. Hence, the set of possible codes is the number of permutations, $P(10,4) = 5040$. 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]. The symbols A and B are defined as follows:

- A is the number of "direct hits," i.e., the number of positions j such that $s_j = g_j$.
- 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 $(1,2,3,4)$, then the responses for the guesses $(3,1,5,4)$, $(3,1,4,5)$ are $[1,2]$, $[0,3]$, respectively. The goal of the codebreaker is, based on the responses, to minimize the number of guesses needed, and to find the secret code. Since the search space for "Bulls and Cows" is extremely large, in the past no optimal strategy for the game has yet been found.

Another well-known deductive game, Mastermind [11], is very popular in America. The two main differences from "Bulls and Cows" are that (1) a secret code consists of four pegs (digits) out of six possible colors (symbols) and that (2) repeated colors are allowed.

In this paper, we develop systematic optimization algorithms, *k-way-branching* (*KWB*) and *pigeonhole-principle-based backtracking* (PPBB), for deductive games. Experimental results show that KWB outperforms previously published *approximate* strategies. Moreover, by applying the algorithms, we derive the exact bound: *7 guesses are necessary and sufficient for the "Bulls and Cows" in the worst case.* This is the first paper to achieve the theorem.

This paper is organized as follows. Section 2 describes the optimization problem for deductive games. In Sections 3 and 4, the algorithms KWB and PPBB are introduced in detail. We also apply KWB to find a strategy for a deductive game, Mastermind; then a comparison of our results with previously published results is given. In Section 5, we derive the exact bound *on the number of guesses required* for the game "Bulls and Cows" in the worst case. Section 6 presents our conclusions.

2. Problem definitions

Optimization problems for deductive games are classified into two categories, i.e., to find an optimal strategy for the games in *the worst case* and in *the expected case*. Optimal strategies for the two cases can be defined as follows:

- *An optimal strategy in the worst case* is a strategy that minimizes the maximum number of guesses required to discover an arbitrary secret code given by the codemaker.
- *An optimal strategy in the expected case* is a strategy that minimizes the total number of guesses required to discover all possible secret codes, assuming a uniform distribution over the possible secret codes.

The problem of finding optimal strategies for a game in the worst and expected cases can be translated to that of minimizing *the height* H and *the external path length* [22] L of a game tree, respectively. Table 1 summarizes the definitions of variables used hereafter in the mathematical formulation of the problem and in the algorithm description for solving it.

We now investigate how large the search space is for a deductive game. For a game tree with height H and branching factor b, the search space will be b^H. For a deductive game, the branching factor for each

Table 1
Notation

Variable	Definition
H	The height of a game tree, i.e., the length of a longest path from the root to a leaf in the game tree
L	The *external path length* of a game tree, i.e., the sum of the distances from the root to each leaf in the game tree
S	The set of *remaining candidates*, i.e., the codewords compatible with responses given so far, which also represents a state of a game
r	The number of response classes in a game, e.g., $r = 14$ for both the games of Mastermind and "Bulls and Cows"
S_g^r	r partitions (or subsets) of S after a guess g, i.e., $S_g^r = \{S_{g,1}, S_{g,2}, \ldots, S_{g,r}\}$, where $\cup_{1 \leqslant i \leqslant r} S_{g,i} = S$ and $S_i \cap S_j = \emptyset$ for $i \neq j$, $1 \leqslant i,j \leqslant r$
$\langle S_g^r \rangle$	The number of remaining candidates of the r response classes after a guess g, i.e., $\langle S_g^r \rangle = \langle \lvert S_{g,1}\rvert, \lvert S_{g,2}\rvert, \ldots, \lvert S_{g,r}\rvert \rangle$
$S_{g,\max}$	The largest of response classes in S_g^r. i.e., $S_{g,\max} = \max_{1 \leqslant j \leqslant r} (\lvert S_{g,j}\rvert)$

move depends on the number of possible responses r and possible codewords c of the game. Accordingly, the branching factor $b = r \times c$, and hence an upper bound of the search space is equal to $b^H = (r \times c)^H$.

For a game with m digits (pegs) out of n symbols (colors), i.e., an $m \times n$ game, we have $r = (m^2 + 3m)/2$ and $c = n^m$ or $n!/(n-m)!$ for a game with repeated colors allowed or not. Therefore, $b^H = (n^m (m^2 + 3m)/2)^H$ or $(n!(m^2 + 3\,m)/2\,(n-m)!)^H$. As the value b becomes larger, the game tree will become too huge to deal with.

3. The k-way-branching algorithm

In this section, we first introduce the KWB approach and demonstrate how it works. Then we apply it to solve a deductive game, Mastermind. Finally, a comparison of our results with previously published results is given.

3.1. Algorithm description

The KWB algorithm is a *modified exhaustive depth-first search* on a game tree. The main modification to depth-first search is that at each visited node we consider only k guesses that are most likely to obtain the optimal strategy, instead of all possible guesses. The idea is called "beam search." [21] Depending on the execution time and space allowed, we can increase the value of k to approach the optimal result. Now the problem is how to choose the "best" k guesses to explore the game tree while performing the search. The problem can be efficiently solved by a *clustering approach*. KWB performs clustering using a concept of *hash collision groups* (*HCG*).

In KWB, the next guesses which seem to lead to the same results are clustered together in an HCG by a given *hash function* to the problem at hand. That is, the guesses with the same hash value will be clustered together. We will give detailed examples of how the clustering mechanism works later. Fig. 1 illustrates the relation between HCGs and equivalent classes in a search space of next guesses (components). The important properties of HCGs include:

- For two components in the same HCG, they are equivalent with high probability. On the other hand, for two equivalent components, they are definitely in the same HCG.
- Given a hash function, it is efficient to obtain the k best HCGs.
- Without losing the generality, an arbitrary component can be chosen to represent its HCG.

The sketch of the KWB algorithm for deductive games is shown in Fig. 2. Initially, the parameter S contains all the possible candidates. After a guess g, S will be partitioned into different response classes, namely, $S_g^r = \{S_{g,1}, S_{g,2}, \ldots, S_{g,r}\}$, where r is the number of response classes in a game, e.g., $r = 14$ for both of Mastermind and "Bulls and Cows." The 14 response classes are shown in the first row of Table 2. The objective to be minimized by KWB is the local variable *Obj* in line 1 of Fig. 2.

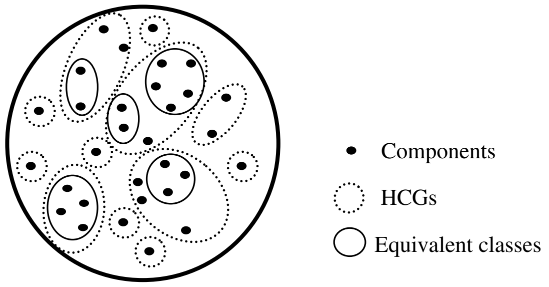

Fig. 1. An illustration of search space of next possible guesses.

```
    Function KWB (S) {                                      // S: current state which is the set of remaining candidates
01      Var Obj;                                            //Obj : the objective to be minimized by KWB
02      If (final state reached) then Return 0 ;
03      For (each guess $g \in$M) {                         // M: the set of possible next guesses
04          id = Hash $\langle S_g^r \rangle$ ;             // classify possible next guesses to HCGs by a hash function
05          $HCG_{id} \leftarrow HCG_{id} \cup \{g\}$ ;
06      }
07      B = {$HCG_{id}$ | $HCG_{id}$ is the top k groups by a given heuristic} ;   // B: the set of k selected HCGs
08      For (each $HCG_{id} \in$B) {
09          $g_{id}$ = Choose($HCG_{id}$) ;                 // $g_{id}$ : an arbitrarily selected representative for $HCG_{id}$
10          $g_{id}$ partitions S into r classes, $S_{id,j}$, j = 1, …, r ;   // r : number of possible response classes
11          For (each $S_{id,j}$) {
12              $Obj_{id,j}$ = KWB($S_{id,j}$) ;           // recursively k-way search to find the best solution
13          }
14          $Obj_{id}$ = accumulate($Obj_{id,j}$) for {j| j = 1, …, r };   // accumulate results from the r response classes
15      }
16      Obj = minimize($Obj_{id}$) for {id| $HCG_{id} \in$B };   // select the minimal value of the k $Obj_{id}$s from k-way search
17      Return Obj+ cost (S);                               // cost(S): the cost of current node
18  }
```

Fig. 2. The sketch of the KWB algorithm for deductive games.

Table 2
The number of remaining candidates in each class after the first guess for the game of Mastermind

	Class													
Guess	[4,0]	[3,0]	[2,2]	[2,1]	[2,0]	[1,3]	[1,2]	[1,1]	[1,0]	[0,4]	[0,3]	[0,2]	[0,1]	[0,0]
0000	1	20	0	0	150	0	0	0	500	0	0	0	0	625
0001	1	20	3	24	123	0	27	156	317	0	0	61	308	256
0011	1	20	4	32	114	0	36	208	256	1	16	96	256	256
0012	1	20	5	40	105	4	84	230	182	2	44	222	276	81
0123	1	20	6	48	96	8	132	252	108	9	136	312	152	16

The objective could be H or L for the worst or expected cases respectively. In lines 14, 17, the two functions, *accumulate* and *cost*, could have various designs tailored to different objectives to be minimized. For example, the function *accumulate* will return *the maximum height* and *the sum of external path lengths* of the r subtrees rooted at the current node for the game in the worst and expected cases, respectively. On the other hand, the function *cost* will return 1 and $|S|$ for the game in the worst and expected cases, respectively.

Note that the performance of KWB critically depends on the choice of heuristic and the corresponding hash function to the problem at hand. KWB cannot guarantee to yield the optimal strategies even the value of k equals 1296, except that each of the HCGs produced by the chosen hash function is also an equivalent class.

Now we demonstrate two hash functions that are tailored to the two cases of optimization problems for deductive games. In Section 3.4, we will give the experimental results for KWB with the two hash functions.

3.2. KWB for deductive games in the worst case

For the worst case, we have to minimize the height, H, of a game tree so as to obtain the optimal strategy for the game. To achieve this goal, we give the following heuristic.

- *Heuristic for deductive games in the worst case.* Minimize the number of remaining candidates of larger response classes after each guess.

According to the heuristic, we design the corresponding hash function as follows:

- *Hash function for deductive games in the worst case.*

$$\begin{aligned}\text{Hash}_w(\langle S^r_g\rangle) &= \langle |S_{g,1}|, |S_{g,2}|, \ldots, |S_{g,r}|\rangle \\ &= \langle |S'_{g,1}|, |S'_{g,2}|, \ldots, |S'_{g,r}|\rangle,\end{aligned}$$

where $|S'_{g,1}| \leqslant |S'_{g,2}| \leqslant \cdots \leqslant |S'_{g,r}|$. That is, the hash function sorts the original sequence $\langle S^r_g\rangle$ into a nondecreasing sequence $\langle S'^r_g\rangle$. If $\langle S'^r_g\rangle = \langle S'^r_h\rangle$, i.e., $|S'_{g,i}| = |S'_{h,i}|$, $1 \leqslant i \leqslant r$, then the guess g and the guess h are classified into the same HCG since any "norm" on the vector $\langle S^r_g\rangle$ will have the desired properties.

Let $\langle S'^r_g\rangle < \langle S'^r_h\rangle$ denote $\langle S'^r_{g,r}\rangle < \langle S'^r_{h,r}\rangle$ or $(\langle S'^r_{g,i}\rangle = \langle S'^r_{h,i}\rangle$ and $\langle S'^r_{g,j}\rangle < \langle S'^r_{h,j}\rangle$, $j < i \leqslant r$ for some $j \geqslant 1$.) Then, according to the above heuristic, the k HCGs with the smallest $\langle S'^r_g\rangle s$ would be selected in the KWB algorithm.

3.3. KWB for deductive games in the expected case

In this case, we will minimize the external path length, L, of a game tree to achieve the optimum. The heuristic used in this case is a concept from information theory, called *entropy*.

- *Heuristic for deductive games in the expected case.* Maximize the entropy caused by partitioning the remaining candidates after a guess.

The corresponding hash function is defined as follows:

- *Hash function for deductive games in the expected case.* Let $\langle S^r_g\rangle = \langle |S_{g,1}|, |S_{g,2}|, \ldots, |S_{g,r}|\rangle$ and $|S| = \sum_{i=1}^{r} |S_{g,i}|$. We define the following hash function.

$$\text{Hash}_e(g) = \sum_{i=1}^{r} -p_i \log_2 p_i, \text{ where } p_i = \frac{|S_{g,i}|}{|S|}.$$

Note that a guess that causes even partitions will have larger entropy. For example, in Table 2, the entropies for the guesses 0000, 0001, 0011, 0012, and 0123 are 1.498, 2.693, 2.885, 3.044, and 3.057, respectively. Now, the k HCGs with the largest entropies will be selected in the KWB algorithm.

3.4. Experimental results

A. Mastermind. A comparison between previous results and our results is shown in Table 3. For the game in the worst case, Knuth [11] developed an optimal strategy, where $H = 5$. For the game in the expected case, the best "approximate" strategy in literature, $L = 5656$, was obtained by Neuwirth [18]. Finally, Koyama and Lai [15] found, by exhaustive search, that $L = 5625$ is the optimal result in the expected case. This strategy should take six guesses in the worst case. A strategy that is optimal in the worst case is also shown, in which $H = 5$ and $L = 5626$. However, the run time could be very long but has not been reported.

We performed two series of experiments, denoted series (1) and (2). All experiments were run on a Pentium IV 2.8 GHz computer. From the experimental results, we have the following observations:

- In series (1), when $k = 1$, our result is optimal in the worst case, i.e., $H = 5$, and is within 97.386% of optimum, which is better than the result [11] in the expected case.
- In series (2), when $k = 20$, the result $L = 5649$ is within 99.575% of optimum, which outperforms the best result of previous heuristic strategies [18].
- All experiments in series (1), KWB with Hash$_w$, have achieved the optimum for the game in the worst case, i.e., $H = 5$.
- All experiments in series (2), KWB with Hash$_e$, obtain better results for L than those obtained by the corresponding experiments in series (1). For the example of $k = 1$, the experiment in series (2) obtains $L = 5683$, which is smaller than 5776 obtained by the experiment in series (1).
- Compared to the latest published *genetic algorithm* [10], KWB obtains higher quality of results both in the worst and expected case for the game. In [10], although the average run time is shorter,

Table 3
Comparison of our results and previous results for the game of Mastermind

Previous methods and our experiments	External path length (L)	The number of guesses in the expected case ($L/6^4$)	The number of guesses in the worst case (H)	% of optimum in the expected case (**5625**/L)	Run time (seconds)
Knuth [11]	5803	4.478	**5**	96.933%	N/A
Irving [8]	5662	4.369	6	99.346%	N/A
	5664	4.370	5	99.311%	N/A
Neuwirth [18]	**5656**	4.364	6	99.452%	N/A
Koyama [15]	**5625**	4.340	6	100%	**N/A**
	5526	4.341	5	99.982%	**N/A**
Kooi [14]	5668	4.373	6	99.241%	**N/A**
(1) *KWB with Hash$_w$*					
$k = 1$	**5776**	4.459	**5**	**97.386**%	1.19
$k = 5$	5749	4.436	5	97.860%	12.55
$k = 10$	5711	4.407	5	98.494%	46.55
$k = 20$	5669	4.374	5	99.224%	199.16
(2) *KWB with Hash$_e$*					
$k = 1$	**5683**	4.385	6	98.979%	1.25
$k = 5$	5673	4.377	6	99.154%	11.56
$k = 10$	5662	4.369	6	99.347%	47.80
$k = 20$	**5649**	4.359	6	**99.575%**	993.20

the average number of guesses is 4.75, which is even larger than 4.457 obtained by KWB with $k = 1$. Furthermore, the number of guesses in the worst case cannot guarantee any upper bound by using genetic algorithms.

B. "Bulls and Cows": By using the KWB algorithm, we obtain strategies and build a game tree for the game, where the $L = 26376$ and $H = 7$ when $k = 10$. The program was run on a Pentium IV 2.8 GHz computer for about 216 minutes. According to the game tree developed above, we have also implemented an interactive program for the game, which is available on the website [16]. Hence, we have the following lemma evidently. The lemma gives an upper bound on the number of guesses required for "*Bulls and Cows*" in the worst case.

Lemma 1. *For the game*, "*Bulls and Cows*", *there exists a strategy such that the number of guesses required for the code-breaker to obtain the secret code is at most* 7.

4. Pigeonhole-principle based backtracking (PPBB)

In this section, we will demonstrate a computer-aided verification method, PPBB, which is helpful to derive the lower bound on *the number of guesses required for "Bulls and Cows" in the worst case*; i.e., the lower bound on the height H of the game tree. Instead of using the decision tree model to determine a lower bound, PPBB takes advantage of two key features to make the verification algorithm more efficient. The features are *constrained adversary strategies* and *backtracking mechanisms.*

4.1. Constrained adversary strategy

PPBB explores only a small part of a game tree, instead of exhaustive search, to acquire a meaningful lower bound by a strategy called *constrained adversaries* [12]. That is, there exists an adversary who is omnipotent and tries to make a very hard situation for a problem. Then, the optimal solution for solving the situation can obviously serve a lower bound for the problem. Accordingly, PPBB acquires the lower bound by expanding, at each stage, only the response class, $S_{g,\max}$, i.e., the class with the maximum number of remaining candidates after a guess g.

4.2. Backtracking mechanism

PPBB uses a backtracking mechanism which is similar to the idea used in the A^* search [21]. At each node PPBB estimates a lower bound of the height of the subtree rooted at the node. By the estimation, PPBB will make backtrackings occur earlier at the lower level of the game tree. This would significantly reduce the search space. In Lemma 3, we will propose *the extended Pigeonhole principle* for getting the estimations for the game. To develop the mechanism, we define the following terms.

- $g(n)$ denotes the path length from the root of the game tree to node n.
- $h^*(n)$ denotes the *true* height of the subtree rooted at node n.
- $h(n)$ denotes an estimation of $h^*(n)$. $h(n)$ is *admissible* if $h(n) \leqslant h^*(n)$.
- $f(n) = g(n) + h(n)$ denotes the estimation of the height of the game tree through node n.
- lb denotes the lower bound to be verified on the height of the game tree.

Now, we have the following *backtracking condition* for a lower bound verification procedure.

Lemma 2 (Backtracking condition). *If $h(n)$ is admissible and $f(n) \geqslant lb$, then the entire subtree below the node n can be pruned with no influence on the completeness and correctness of the lower-bound verification procedure.*

For our problem, $g(n)$ denotes the level of node n in the game tree. However, it is very hard to compute $h^*(n)$. We propose a new mechanism for deriving an admissible estimation $h(n)$, called the *extended pigeonhole principle*. Let the set of the volumes $V = \{v_1, v_2, v_3, \ldots, v_n\}$ and a proper subset of V, $V' = v'_1, v'_2, v'_3, \ldots, v'_k$, where $1 \leqslant k < n$, i.e., $V' \subset V$. We have the following lemma.

Lemma 3 (Extended pigeonhole principle). *If m pigeons occupy n pigeonholes with different volumes and $m > \sum_{i=1}^{k} v'_i$, then there exists one pigeonhole with at least $\lceil (m - \sum_{i=1}^{k} v'_i)/(n-k) \rceil$ pigeons roosting in it, where $1 \leqslant k < n$.*

Proof. Given $m > \sum_{i=1}^{k} v'_i$, we divide the proof into two cases:

Case 1. If $\sum_{i=1}^{k} v'_i$ pigeons fill up the k pigeonholes whose respective volumes belong to V', then $(m - \sum_{i=1}^{k} v'_i)$ remaining pigeons have to be distributed among the other $(n-k)$ pigeonholes.

Case 2. Otherwise, there are more than $(m - \sum_{i=1}^{k} v'_i)$ remaining pigeons that have to be distributed among the other $(n-k)$ pigeonholes.

Therefore, by *the generalized pigeonhole principle* [20], in both cases there exists one pigeonhole with at least $\lceil (m - \sum_{i=1}^{k} v'_i)/(n-k) \rceil$ pigeons roosting in it. This completes the proof. $\square$

Now, we apply the principle to derive the estimation $h(n)$ for a given node n. The main idea is that the guess made by the code-breaker in each turn may evenly divide the elements in the set S into r response classes. Hence, this strategy can minimize the height of subtree rooted at the current node. In other words, the *optimistic* strategy will lead to an estimation, $h(n)$. Obviously, $h(n) \leqslant h^*(n)$; i.e., $h(n)$ is admissible.

We illustrate how to derive the estimation $h(n)$ by the example of "Bulls and Cows." Table 4 shows the number of remaining candidates in each class after the first guess. Note that the size of each class shown in Table 4 is the maximum possible size during the game-guessing process because we obtained it from partitioning all 5040 possible candidates. Therefore, the sizes of the 14 classes can be considered to be the volumes of the corresponding 14 pigeonholes.

4.3. The algorithms

The PPBB algorithm is shown in Fig. 3. Initially, the main program makes the function call, PPBB $(S, 1)$, where S is the set of all the 5040 possible candidates. In line 5, Get_Estimation $(S_{g,\max})$ is a subroutine for estimating the number of further guesses required. The details of the subroutine are given in Fig. 4. Lines 6–8 determine one of the following cases for the program: to backtrack, to report failure, or to expand recursively. Line 10 returns "success!" if every $g \in M$ successfully passes the verification program, where M is the set of possible next guesses.

For applying Lemma 3 to derive the estimation $h(n)$, we let $\langle S'_g \rangle = \langle |S'_{g,1}|, |S'_{g,2}|, \ldots, |S'_{g,r}| \rangle$ be a non-decreasing sequence of the 14 pigeonholes, i.e., $\langle S'_g \rangle = \langle 1, 6, 8, 9, 24, 72, 180, \ldots, 1440 \rangle$. Fig. 4 shows the sketch of the algorithm for deriving $h(n)$ in PPBB. Lines 2–6 employ the extended pigeonhole

Table 4
The number of remaining candidates in each class after the first guess for "Bulls and Cows"

Class	[4,0]	[3,0]	[2,2]	[2,1]	[2,0]	[1,3]	[1,2]	[1,1]	[1,0]	[0,4]	[0,3]	[0,2]	[0,1]	[0,0]
Size	1	24	6	72	180	8	216	720	480	9	264	1260	1440	360

```
01  PPBB (S, level) {
02    For (each $g \in M$) {                          // M is the set of possible next guesses
03      g partitions S into r response classes, $S_g^r = \{S_{g,1}, S_{g,2}, \ldots, S_{g,r}\}$
04      $S_{g,max} = \max_{1 \le j \le r}(|S_{g,j}|)$;
05      $hn$ = Get_Estimation($S_{g,max}$);               // hn: estimation for the number of further guesses
                                                       required
06      If ($level + hn \ge lb$) Continue;             // backtrack and deal with another guess $g \in M$
                                                     // lb: the lower bound to be verified
07      ElseIf ($hn = 1$)  Return "failure!"           // final state is reached and $level + hn < lb$
08      Else result = PPBB($S_{g,max}$, level + 1) ;   // recursively expand the lower levels
09    }
10    Return "success!"                                // every $g \in M$ passes the verification program
11  }
```

Fig. 3. The sketch of PPBB.

```
    Get_Estimation(S) {
01    Estimation_hn = 0
02    While (|S|>2) {
03      Estimation_hn ++;
04      Find the smallest q, such that $\lfloor (|S| - \sum_{i=1}^{q} |S'_{g,i}|)/(n-q) \rfloor \le |S'_{g,q+1}|$;
05      |S| = $\lfloor (|S| - \sum_{i=1}^{q} |S'_{g,i}|)/(n-q) \rfloor$;
06    }
07    If (|S|=2) Estimation_hn = Estimation_hn +2;
08    If (|S|=1) Estimation_hn ++;
10  Return Estimation_hn;
11  }
```

Fig. 4. The sketch of the algorithm for deriving the estimation $h(n)$.

principle to deal with the case $|S| > 2$. Lines 7 and 8 are for the case $|S| \leqslant 2$. Finally, the algorithm will return the estimation $h(n)$ for a given state S.

The estimations $h(n)$ for states with different ranges of $|S|$ can efficiently be calculated by *Get_Estimation*, as shown in Table 5. In Table 5, the notation $[a,b)$ denotes a half-open interval, which means $a \leqslant |S| < b$. Note that the lower bound for $h(n)$ is better than the trivial $1 + \lceil \log_{14}(n) \rceil$.

As a comparison, applying PPBB to Mastermind, we should set the lower bound to be verified to be five, $lb = 5$. Observing Table 2, no matter what the first guess is chosen, it will leaves at least 256 remaining candidates, i.e., $S_{g,\max} \geqslant 256$. By the pigeonhole principle, the estimation $h(n)$ will be 4, and hence $level + h(n) = 5$. All the five possible

Table 5
The estimation $h(n)$ for different states calculated by *Get_Estimation*

The interval of $\lvert S\rvert$	The estimation, $h(n)$
1	1
$[2, 15)$	2
$[15, 165)$	3
$[165, 1433)$	4
$[1433, 5040]$	5

guesses 0000, 0001, 0011, 0012, 0123 will backtrack at line 6 in Fig. 3 and return "success!" Therefore, we get the lower bound, which is 5.

5. The exact bound analysis for "Bulls and Cows" in the worst case

Now we give an example which applies *the extended pigeonhole principle* to prove a lower bound on the height of the game tree for "Bulls and Cows".

Lemma 4. *The height of the game tree for "Bulls and Cows" is at least* 6.

Proof. As shown in Table 4, after the first guess, class [0,1] contains 1440 remaining candidates. Furthermore, we can obtain $h(n) = 5$ for the state, $|S| = 1440$, from Table 5. Hence the estimation of lower bound is $f(n) = g(n) + h(n) = 1 + 5 = 6$. Therefore, at least 6 guesses are required in the worst case, and hence the height of the game tree for "Bulls and Cows" is at least 6. □

The above lemma only shows a loose lower bound 6 for this problem. Now we apply the PPBB shown in Fig. 3 to verify that the height of the game tree is at least 7. As mentioned in Section 2, the search space would be $(5040 \times 14)^7$ for the verification. It is impossible to obtain the lower bound by exhaustive search. PPBB effectively reduces the size of the search space in the following ways.

First, since the *constrained adversary strategy* is used, only $S_{g,\max}$ is explored at each stage. The search space is reduced to $(5040)^7$.

Second, a simple *equivalent* property is used for the first and the second guesses. At the first guess, since all ten digits are unused, we can guess any one of the 5040 possible secret codes, say $(0,1,2,3)$. At the second guess, because only four out of ten digits are used by the first guess, for example $(0,1,2,3)$, the other six unused digits $\{4,5,\ldots,9\}$ can be treated as equivalent symbols. Hence, in the second guess, we can only consider

$$\sum_{i=0}^{4} C(4,i) * P(4,i) = 209$$

representative guesses, where i is the number of digits used in both the first and the second guesses. Now, the search space can be further reduced to $209 \times (5040)^5$. To make the verification procedure more efficient, an anonymous reviewer of this paper noted that the 209 cases can be further reduced to 19 symmetric cases by using algebra packages, such as Nauty. The 19 representatives are: 1235, 1243, 1245, 1256, 1325, 1342, 1345, 1356, 1567, 2143, 2145, 2156, 2315, 2341, 2345, 2356, 2546, 2567, and 5678.

Third, we use the backtracking technique and thus reduce the search space to $209 \times (5040)^3$ if all backtrackings occur at levels 5.

Finally, the PPBB approach is applied to the verification procedure for the game. The numbers of backtrackings occurred at levels 3, 4, and 5 are 407,528, 3,254,981,544, and 59,149,440, respectively. According to the numbers of backtrackings, it is easy to show that all $209 \times (5040)^3$ paths do backtrack at levels 3, 4, and 5 in the verification procedure.

Now, we can derive the exact bound of the number of guesses required for the "Bulls and Cows" in the worst case, as shown in the following Theorem.

Theorem 1. *Seven guesses are necessary and sufficient for the "Bulls and Cows" in the worst case.*

Proof. "Bulls and Cows" has successfully passed the verification procedure PPBB. That is, it returns "success!" in Line 10 of Fig. 3. The verification program was run on a Pentium III 750 computer for about two days. We conclude that seven guesses are *necessary* for "Bulls and Cows" in the worst case. Furthermore, the *sufficient* condition can be obtained by Lemma 1. This completes the proof. □

6. Conclusions

In this paper, we have investigated properties and given formulations for strategy optimization for deductive games. Moreover, two new algorithms, KWB and PPBB, for deductive games have been proposed. These algorithms have successfully been applied to solve deductive games, *Mastermind* and "*Bulls and Cows*." Experimental results show that KWB performs best among previously published *approximate* strategies for the game of Mastermind. Applying KWB to "Bulls and Cows", we can obtain a strategy which is optimal in the worst case and which could approach the optimal one, by increasing the parameter k, in the expected case.

Instead of using the decision tree model to determine a lower bound, PPBB takes advantage of several key features to make the verification algorithm more efficient. These features include *constrained adversary strategies*, *backtracking mechanisms*, and *lower-bound estimations*. PPBB has been applied to "Bulls and Cows" and obtained a meaningful lower bound. Therefore, we are able to determine the exact bound on the number of guesses required for the game in the worst case. The game trees and strategies developed in this study have been implemented as interactive programs, which are available on the website [16].

The proposed concepts and algorithms could be applied to related problems. We hope this paper will prompt researchers to study other related problems.

Acknowledgements

The authors thank anonymous referees for their valuable comments. The research is supported in part by a Grant NSC93-2213-E-003–001, ROC.

References

[1] L. Bento, L. Pereira, A. Rosa, Mastermind by evolutionary algorithms, in: Proceedings of the International Symposium on Applied Computing, 1999, pp. 307–311.

[2] J.L. Bernier, C.I. Herraiz, J.J. Merelo, S. Olmeda, A. Prieto, Solving mastermind using gas and simulated annealing: A case of dynamic constraint optimization, in: Proceedings PPSN, Parallel Problem Solving from Nature IV, in Computer Science, 1141, 1996, pp. 554–563.

[3] S.T. Chen, S.S. Lin, Optimal algorithms for $2 \times n$ AB games – A graph-partition approach, Journal of Information Science and Engineering 20 (1) (2004) 105–126.

[4] S.T. Chen, S.S. Lin, Optimal algorithms for $2 \times n$ mastermind games – A graph-partition approach, Computer Journal 47 (5) (2004) 602–611.

[5] S.T. Chen, S.S. Lin, Novel algorithms for deductive games, in: Proceedings of the 2004 International Computer Symposium, Taipei, Taiwan, December 15–17, 2004, pp. 517–522.

[6] Z. Chen, C. Cunha, Finding a hidden code by asking questions, COCOON'96, Computing and Combinatorics (1996) 50–55.

[7] M.M. Flood, Sequential search strategies with Mastermind variants—Part 1, Journal of Recreational Mathematics 20 (2) (1988) 105–126.

[8] R.W. Irving, Towards an optimum Mastermind strategy, Journal of Recreational Mathematics 11 (2) (1978–79) 81–87.

[9] G. Kabatianski, V. Lebedev, V. Lebedev, The Mastermind game and the rigidity of the hamming space, in: Proceedings of the International Symposium on Information Theory IEEE, 2000, pp. 375–375.

[10] T. Kalisker, D. Camens, Solving mastermind using genetic algorithms, GECCO 2003, LNCS 2724 (2003) 1590–1591.

[11] D.E. Knuth, The computer as Mastermind, Journal of Recreational Mathematics 9 (1) (1976) 1–6.

[12] D.E. Knuth, The art of computer programming – sorting and searching, vol. 3, second ed., 1998.

[13] K.-I. Ko, S.-C. Teng, On the number of queries necessary to identify a permutation, Journal of Algorithms 7 (1986) 449–462.

[14] B. Kooi, Yet another mastermind strategy, ICGA Journal 28 (1) (2005) 13–20.

[15] K. Koyama, T.W. Lai, An optimal Mastermind strategy, Journal of Recreational Mathematics 25 (1993) 251–256.

[16] S.S. Lin, 2005, Web site: <http://csie.ntnu.edu.tw/~linss/deductivegame>.

[17] J.J. Merelo, J. Carpio, P. Castillo, V.M. Rivas, G. Romero, (GeNeura Team), Finding a needle in a haystack using hints and evolutionary computation: The case of Genetic Mastermind, in: Genetic and Evolutionary Computation Conference late breaking papers books, 1999, pp. 184–192.

[18] E. Neuwirth, Some strategies for Mastermind, Zeitschrift fur Operations Research 26 (1982) 257–278.

[19] J.R. Roche, The value of adaptive questions in generalized Mastermind, in: Proceedings of the International Symposium on Information Theory IEEE, 1997, pp. 135–135.

[20] K.H. Rosen, Discrete Mathematics and its Applications, fifth ed., McGraw-Hill, 2003.

[21] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, 2003.

[22] R. Sedgewick, Algorithms, second ed., Addison-Wesley, 1988.